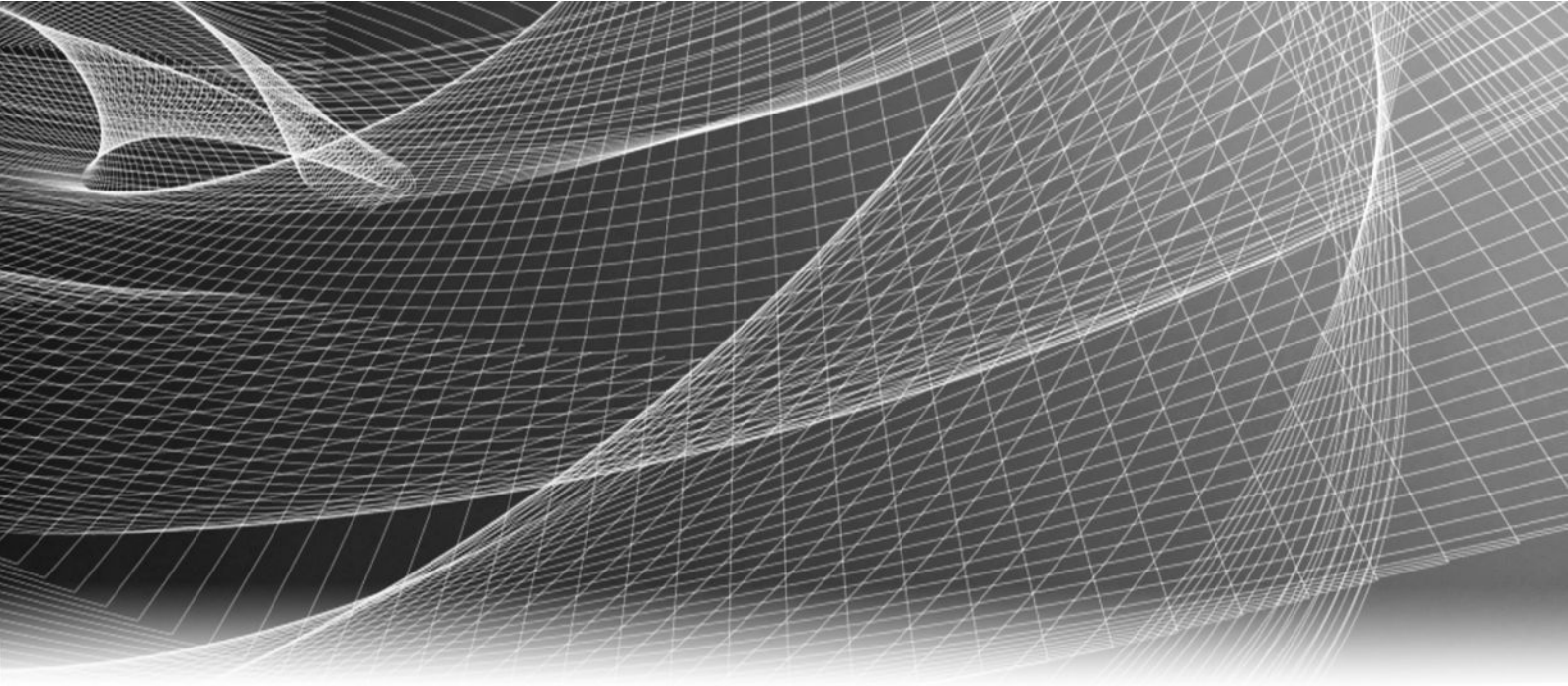


EMC® Avamar®
Avamar REST API

Version 7.3

Getting Started Guide

302-002-857
REV 01

EMC²

Published April, 2016

EMC Corporation
Hopkinton, Massachusetts 01748-9103
1-508-435-1000 In North America 1-866-464-7381
www.EMC.com

CONTENTS

FIGURES

TABLES

Preface

As part of an effort to improve its product lines, EMC periodically releases revisions of its software and hardware. Some versions of the software or hardware currently in use do not support every function that this document describes. The product release notes provide the most up-to-date information on product features.

If a product does not function correctly or does not function as described in this document contact an EMC technical support professional.

Note

This document was accurate at publication time. Go to EMC Online Support (https://support.EMC.com) to find the latest version of this document.

Purpose

This document provides information to get started using the Avamar REST API.

Audience

This document is intended for system programmers who are responsible for accessing Avamar system resources through the Avamar REST API.

Revision history

The following table presents the revision history of this document.

Table 1 Revision history

Revision	Date	Description
01	April 2016	GA release of Avamar 7.3

Related documentation

The following EMC publications provide additional information:

- HTML-formatted Avamar REST API API specification
- *EMC Avamar Administration Guide*
- *EMC Avamar Management Console Command Line Interface (MCCLI) Programmer Guide*

Special notice conventions used in this document

EMC uses the following conventions to alert the reader to particular information.

NOTICE

The Notice convention emphasizes important information about the current topic.

Note

The Note convention addresses specific information that is related to the current topic.

Typographical conventions

In this document, EMC uses the typographical conventions that are shown in the following table.

Table 2 Typographical conventions

Convention	Example	Description
Bold typeface	Click **More Options**.	Use for names of interface elements, such as names of windows, dialog boxes, buttons, fields, tab names, key names, and menu paths (what a user specifically selects or clicks).
Italic typeface	*EMC Avamar Administration Guide*	Use for full titles of publications that are referenced in text.

Monospace font	`Event Type = INFORMATION`
	`Event Severity = OK`
	`Event Summary = New group created`
		Use for:
		• System code
		• System output, such as an error message or script
		• Pathnames, file names, prompts, and syntax
		• Commands and options

Monospace font with italic typeface	Type ***Avamar_server***, where *Avamar_server* is the DNS name or IP address of the Avamar server.	Use for variables.
Monospace font with bold typeface	Type **yes**.	Use for user input.
Square brackets	**[--domain=*String*(/)] --name=*String***	Square brackets enclose optional values.
Vertical bar	**[--domain=*String*(/)] | --name=*String***	Vertical bar indicates alternate selections - the bar means “or”.
Braces	**{[--domain=*String*(/)] | --name=*String*}**	Braces enclose content that the user must specify.
Ellipses	`valid hfs ...`	Ellipses indicate nonessential information that is omitted from the example.

Where to get help

The Avamar support page provides access to licensing information, product documentation, advisories, and downloads, as well as how-to and troubleshooting information. This information may enable you to resolve a product issue before you contact EMC Customer Support.

To access the Avamar support page:

1. Go to https://support.EMC.com/products.
2. Type a product name in the **Find a Product** box.
3. Select the product from the list that appears.
4. Click the arrow next to the **Find a Product** box.
5. (Optional) Add the product to the **My Products** list by clicking **Add to my products** in the upper right corner of the **Support by Product** page.

Documentation

The Avamar product documentation provides a comprehensive set of feature overview, operational task, and technical reference information. Review the following documents to supplement the information in product administration and user guides:

- Release notes provide an overview of new features and known limitations for a release.
- Technical notes provide technical details about specific product features, including step-by-step tasks, where necessary.
- White papers provide an in-depth technical perspective of a product or products as applied to critical business issues or requirements.

Knowledgebase

The EMC Knowledgebase contains applicable solutions that you can search for either by solution number (for example, esgxxxxxx) or by keyword.

To search the EMC Knowledgebase:

1. Click **Search** at the top of the page.
2. Type either the solution number or keywords in the search box.
3. (Optional) Limit the search to specific products by typing a product name in the **Scope by product** box and then selecting the product from the list that appears.
4. Select **Knowledgebase** from the **Scope by resource** list.
5. (Optional) Specify advanced options by clicking **Advanced options** and specifying values in the available fields.
6. Click **Search**.

Online communities

Go to EMC Community Network at http://community.EMC.com for peer contacts, conversations, and content on product support and solutions. Interactively engage online with customers, partners, and certified professionals for all EMC products.

Live chat

To engage EMC Customer Support by using live interactive chat, click **Join Live Chat** on the **Service Center** panel of the Avamar support page.

Service Requests

For in-depth help from EMC Customer Support, submit a service request by clicking **Create Service Requests** on the **Service Center** panel of the Avamar support page.

Note

To open a service request, you must have a valid support agreement. Contact an EMC sales representative for details about obtaining a valid support agreement or with questions about an account.

To review an open service request, click the **Service Center** link on the **Service Center** panel, and then click **View and manage service requests**.

Enhancing support

EMC recommends that you enable ConnectEMC and Email Home on all Avamar systems:

- ConnectEMC automatically generates service requests for high priority events.
- Email Home sends configuration, capacity, and general system information to EMC Customer Support.

Comments and suggestions

Comments and suggestions help EMC to continue to improve the accuracy, organization, and overall quality of the user publications. Send comments and suggestions about this document to DPAD.Doc.Feedback@emc.com.

Please include the following information:

- Product name and version
- Document name, part number, and revision (for example, 01)
- Page numbers
- Other details to help address documentation issues

CHAPTER 1

Introduction

This chapter includes the following topics:

Description

The Avamar REST API provides an API to develop applications and tools that interact with Avamar systems. The Avamar REST API uses client-server communication based on the representational state transfer (REST) API architecture model.

Programming interface

Using the Avamar REST API you can write code that manages multiple Avamar systems simply and efficiently. The Avamar REST API abstracts Avamar systems and domains into logical entities. By doing this, the Avamar REST API enhances your ability to write code that manages your Avamar systems and your customer's requirements.

While the Avamar REST API handles management tasks that can also be handled through Avamar Administrator and through the Avamar MCCLI, the Avamar REST API is not intended to replace those tools. Instead the Avamar REST API contributes a different perspective and a unique model for Avamar system management.

REST architecture

The Avamar REST API uses the representational state transfer (REST) architectural style. The REST architectural style permits the Avamar REST API to provide a platform independent and language independent interface for managing multiple Avamar systems.

Purpose

The Avamar REST API simplifies the creation of custom web portals for customers who deliver data protection services to end users. The Avamar REST API provides a granular and responsive interface that can be easily integrated with modern web applications. The Avamar REST API also provides a new and less-complex model for managing multiple Avamar systems as a single logical entity.

Custom web portals

The Avamar REST API expands and improves on the available methods for providing Avamar data protection features as a service. By using the Avamar REST API, you can create custom web portals that interact with your Avamar systems through a REST programing interface.

Simplified management of multiple Avamar systems

The Avamar REST API is designed to simplify the task of managing multiple Avamar systems from a central location. The Avamar REST API provides methods to group Avamar systems into logical entities and allows you to perform operations on them in parallel. This reduces the complexity required to manage multiple Avamar systems.

The Avamar REST API is designed to help with the challenges of capacity management in large Avamar environments. The Avamar REST API has built in intelligence to determine the best Avamar system to add new clients to in order to optimize storage capacity.

Deployment

EMC provides Avamar REST API in an RPM Package Manager (RPM) file that is separate from the Avamar server software.

RPM file

You obtain Avamar REST API through an EMC sales representative. You receive an RPM file that contains:

- Avamar REST API server
- Avamar REST API documentation

Install the Avamar REST API on an EMC Backup and Recovery Manager server, or on an Avamar utility node or single-node Avamar server.

When installed on an EMC Backup and Recovery Manager server, the Avamar REST API server uses the available messaging subsystem to provide real-time synchronization of data protection resource information.

When installed on an Avamar utility node or single-node Avamar server, the Avamar REST API server relies on GET calls to obtain data protection resource information.

After installing the Avamar REST API, access the HTML-formatted API specification at:

```
http://RESTAPISERVER:8580/rest-api-doc/
```

where *RESTAPISERVER* is the IP address, or resolvable hostname, of the computer that hosts the Avamar REST API server.

Documentation conventions

This documentation uses several conventions to increase the readability of the descriptions and examples. The conventions consist of an abbreviated URL and a set of standard variable names.

Abbreviated URL

In the Avamar REST API, the URL that is the target of a GET, POST, PUT, or DELETE request method is often lengthy. Since there is a common segment to every URL used with the Avamar REST API, that segment is presumed in the URL references that appears in this documentation. For example, consider the following GET and URL description:

```
GET https://RESTAPISERVER:8543/rest-api/client/4702406e-d989-4058-
a57b-c66ece0c4f37/detail/job
```

This GET and URL description appears as the following abbreviated description in this documentation:

```
GET /client/4702406e-d989-4058-a57b-c66ece0c4f37/detail/job
```

As this example demonstrates, the URL segment `https://RESTAPISERVER:8543/rest-api` is presumed and removed. This abbreviation convention minimizes the instances of the URL inelegantly wrapping to a new line. This URL abbreviation convention is also the same as the abbreviation convention used for the URL designations in the HTML-formatted API specification provided with the Avamar REST API software.

This abbreviation convention is not used for examples where the full text is set out. In those examples the full URL appears.

Standard variable names

To minimize the repetition of variable definitions in this documentation, the following variables are defined here and used in accord with the listed definition throughout this documentation.

Table 3 Variables used in this documentation

Variable name	Definition
FULL_PATH	Full path to a location in a file system or backup.
BACKUP_URI	Uniform resource identifier assigned to a backup.

Table 3 Variables used in this documentation (continued)

Variable name	Definition
CLIENT_URI	Uniform resource identifier assigned to a client computer.
FOLDER_URI	Uniform resource identifier assigned to a folder.
HVM_URI	Uniform resource identifier assigned to a hypervisor manager, such as a VMware vCenter.
PLUG-IN_URI	Uniform resource identifier assigned to a plug-in.
PLUG-IN_URL	Uniform resource locator used by the Avamar REST API server to reference a plug-in.
POLICY_URI	Uniform resource identifier assigned to a policy.
PROVIDER_URI	Uniform resource identifier assigned to the provider.
PROXY_NAME	Fully qualified domain name of a proxy appliance for VMware.
PROXY_URL	Uniform resource locator used by the Avamar REST API server to reference a proxy appliance.
RESTAPISERVER	IP address or resolvable hostname of the computer that hosts the Avamar REST API server.
RETENTION_URI	Uniform resource identifier assigned to a retention policy.
TASK_URI	Uniform resource identifier assigned to a task.
TENANT_URI	Uniform resource identifier assigned to a tenant.
USERNAME	Username for an account that has permission to su to root.

The definition for any variable that is not defined in this table is provided where the variable occurs in the documentation.

CHAPTER 2

Installation

This chapter includes the following topics:

Requirements

The Avamar REST API server must be installed on a host computer that is running specific data protection server software. Also, some functionality requires a minimum Avamar server version on the Avamar systems that the Avamar REST API server manages.

Table 4 Avamar REST API requirements

Category	Requirement
Installation host server software	Either of the following: • Backup and Recovery Manager server, release 1.2 • Avamar server, release 7.0 or later
Virtual machine: • Configuration • Backup operations • Restore operations	All managed Avamar systems must be running Avamar server version 7.1 or later
Replication destination configuration	All managed Avamar systems must be running Avamar server version 7.1 or later

Installing

Install the Avamar REST API server on a Backup and Recovery Manager server or on an Avamar server.

Before you begin

Select a computer for the software installation (target computer) that meets the requirements of the Avamar REST API.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ssh -l ucas *RESTAPISERVER* -p1315
   - Avamar server

     ssh -l *USERNAME RESTAPISERVER*
2. Type the following command to switch to root:

   su -
3. Copy the RPM package containing the Avamar REST API software to a temporary location on the target computer.
4. Change the working directory to the temporary location of the RPM package.
5. Type the following command to expand and install the Avamar REST API software:

   rpm -ivh *REST_API_VERSION*.rpm

where *REST_API_VERSION*.rpm is the name of the RPM package containing the Avamar REST API software.

The installer does the following:

- Installs the software
- Creates the Avamar REST API database
- Shuts down Tomcat
- Creates the Avamar REST API properties file
- Requests a new, non-default, provider password

6. Create a new provider password.

   To change the password complete the task described in Changing the provider password on page 25. Or, to change the username and password, complete the task described in Changing the provider credentials on page 26.

7. (Backup and Recovery Manager server installs only) Complete the additional configuration steps described in Configuring Backup and Recovery Manager on page 19.

8. Type the following:

   ```
   /usr/local/avamar/bin/restserver.sh --start
   ```

   Tomcat starts and the Avamar REST API server starts. It may take several minutes for Tomcat to initialize the Avamar REST API server.

Results

After Tomcat restarts and initializes the Avamar REST API server, the installation is complete.

After you finish

Use the `curl` tool to test the installation.

Configuring Backup and Recovery Manager

To use the Avamar REST API software on a Backup and Recovery Manager server, additional configuration steps are required.

Before you begin

Do the following:

- Install the Avamar REST API software on a Backup and Recovery Manager server.
- Stop the Avamar REST API server as described in Manually stopping on page 24.

When changing an Avamar REST API default property, change the value in the custom system properties file.

The custom system properties file is: `/usr/local/avamar/var/rest-api/server_data/restserver.properties`

NOTICE

Do not change the default system properties file.

Procedure

1. Log in to a secure shell (SSH) session on the Backup and Recovery Manager computer by typing the following:

```
ssh -l ucas RESTAPISERVER -p1315
```

2. Type the following command to switch to root:

   ```
   su -
   ```

3. Add the following property and value in the Avamar REST API custom system properties file:

   ```
   rabbitmq.broker.host=BRM_IP
   ```

   where *BRM_IP* is the IP address of the Backup and Recovery Manager server.

4. Save and close the custom system properties file.
5. Type the following to stop the Backup and Recovery Manager firewall daemon:

   ```
   rcSuSEfirewall2 stop
   ```

6. Open the following file for editing: `/etc/rabbitmq/rabbitmq.config`.
7. Change the tcp_listeners line to remove the IPv4 loopback address.

   Before the change:

   ```
   {tcp_listeners, [{"127.0.0.1", 5672}]},
   ```

   After the change:

   ```
   {tcp_listeners, [5672]},
   ```

8. Save and close `/etc/rabbitmq/rabbitmq.config`.
9. Type the following to restart the `rabbitmqctl` daemon:

   ```
   rabbitmqctl stop
   service rabbitmq-server start
   ```

   The `rabbitmqctl` daemon reads and applies the new configuration.

After you finish

Complete the steps in Installing on page 18 by starting the Avamar REST API server.

Testing the installation

Use a `curl` command to test the software installation.

Before you begin

Do the following:

- Install the Avamar REST API software on a target computer.
- Find a computer that has the `curl` tool and has access to the target computer.

Avamar server software includes the `curl` tool. The tool is also included with most Linux installations. This test can also be run from the command line on the target computer.

Procedure

1. Log in to an Avamar utility node, or to another computer that has the `curl` tool.

   The computer running `curl` must have network access to the target computer.

2. Type the following command:

   ```
   curl -k -D- --user USERNAME:PASSWORD -X POST https://
   TARGETCOMPUTER:8543/rest-api/login
   ```

where:

- *USERNAME* and *PASSWORD* are the provider credentials for the Avamar REST API software on a target computer. The default values are: **admin:changeme**.
- *TARGETCOMPUTER* is the resolvable hostname or IP address of the target computer.

The Avamar REST API software on a target computer sends back a `201 Created` HTTP response header and a session object.

Example 1 Testing an installation of the Avamar REST API software

In the following example, the admin user on an Avamar server with the hostname of **lava7120** uses the default provider credentials to test an installation on a computer with the hostname of **test.emc.com**.

```
admin@lava7120:~/>: curl -k -D- --user admin:changeme -X POST https://
test.emc.com:8543/rest-api/login
HTTP/1.1 201 Created
X-Concerto-Authorization:
OTQwMGE1MDYtNzIyOC00OTYxLWI4OGMtYjliNWEwNzVmNGE2
Date: Wed, 29 Jul 2015 14:08:02 GMT
Content-Type: application/xml; version=1.0
Transfer-Encoding: chunked
Connection: close
Server: Avamar

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><Session
xmlns="http://www.emc.com/concerto/v1.0" href="https://test.emc.com:
8543/rest-api/session" type="application/xml,application/json"><User
name="admin"/><AccessPoint href="https://test.emc.com:8543/rest-api/
admin/provider/6c86013b-ae5a-49b5-913f-23ab737cb7a4" id="6c86013b-
ae5a-49b5-913f-23ab737cb7a4" name="Root"/></
Session>admin@lava7120:~/>:
```

After you finish

If the `201 Created` HTTP response header does not appear, refer to Troubleshooting on page 81.

Checking the installed version

Use an RPM command to determine the installed version of the Avamar REST API software.

Before you begin

Select a target computer that has the Avamar REST API software installed.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ssh -l ucas *RESTAPISERVER* -p1315
   - Avamar server

     ssh -l *USERNAME RESTAPISERVER*
2. Type the following command to switch to root:

   su -

3. Type the following command to display the version of Avamar REST API software that is installed on the target computer:

   ```
   rpm -q rest-api
   ```

Results

The installed version number appears.

Upgrading

To upgrade the Avamar REST API software to a new version and retain the data in the Avamar REST API database, use the RPM upgrade command.

Before you begin

Select a target computer that has an earlier release of the Avamar REST API software installed.

Procedure

1. Copy the RPM package containing the new version of the Avamar REST API software to a temporary location on the target computer.
2. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ```
     ssh -l ucas RESTAPISERVER -p1315
     ```

   - Avamar server

     ```
     ssh -l USERNAME RESTAPISERVER
     ```

3. Type the following command to switch to root:

   ```
   su -
   ```

4. Change the working directory to the temporary location of the RPM package.
5. Type the following command to upgrade the Avamar REST API software:

   ```
   rpm -Uvh REST_API_VERSION.rpm
   ```

   where *REST_API_VERSION*.rpm is the name of the RPM package containing the new version of the Avamar REST API software.

   The installer expands and installs the software. Next, the installer restarts the Apache Tomcat web server (Tomcat).

   Note

   After the installer restarts Tomcat, it may take several minutes for Tomcat to initialize the Avamar REST API server.

Results

After Tomcat restarts and initializes the Avamar REST API server, you have completed the upgrade.

After you finish

Use the `curl` tool to test the installation.

Uninstalling

Use the RPM erase command, the PostgreSQL clean command, and the OS remove command to remove the Avamar REST API software and associated data.

Before you begin

Select a target computer that has the Avamar REST API software installed.

Remove the Avamar REST API software and associated data to stop using Avamar REST API on the computer, or to prepare for a clean install of the Avamar REST API software.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ssh -l ucas *RESTAPISERVER* -p1315
   - Avamar server

     ssh -l *USERNAME RESTAPISERVER*
2. Type the following command to switch to root:

   su -
3. Type the following RPM query command:

   rpm -qa | grep rest-api

   The system displays the package name of the installed version of the Avamar REST API software.
4. Type the following RPM erase command:

   rpm -e *RESTAPI_RPM_PACKAGE*

   where *RESTAPI_RPM_PACKAGE* is the package name of the installed version of the Avamar REST API software.

   The RPM erase command removes the Avamar REST API software.
5. Switch the active user to the Avamar admin account by typing the following command:

   su admin -
6. Type the following PostgreSQL command:

   psql -p5558 rest-api -c "show data_directory"

   PostgreSQL displays the full path of the data directory for the Avamar REST API database.
7. Type the following PostgreSQL command:

   pg_ctl restart -D *FULL_PATH_OF_DATA_DIRECTORY* -m i

   Where *FULL_PATH_OF_DATA_DIRECTORY* is the full path of the data directory determined in the previous step.

   The default path for the data directory is `/usr/local/avamar/var/avi/server_data/postgres/data`.

   PostgreSQL closes all connections to the data directory.

8. Type the following PostgreSQL command:

   ```
   dropdb -Uadmin -p5558 rest-api
   ```

   This command removes the Avamar REST API database. The command does not remove Avamar REST API objects that were created on associated Avamar servers.

   PostgreSQL removes the Avamar REST API database.

9. Type the following:

   ```
   rm –rf /usr/local/avamar/var/rest-api/
   rm –rf /usr/local/avamar/lib/rest-api/
   rm –rf /usr/local/avamar-tomcat/webapps/rest-api*
   rm –rf /usr/local/avamar-tomcat/work/Catalina/localhost/rest-api
   ```

   The recursive rm commands remove the Avamar REST API files and folders from the computer.

Results

The commands in this task remove the Avamar REST API software and associated data.

Manually stopping

Shutdown the Avamar REST API server from the command line.

Before you begin

Install the Avamar REST API software on an Avamar server or on a Backup and Recovery Manager server.

The Avamar REST API installer configures the Avamar REST API server to stop and start automatically when the host computer stops and starts. Complete this task to stop the Avamar REST API server manually.

NOTICE

Stopping the Avamar REST API server also stops the associated Tomcat instance. Other user interfaces on the Avamar system share the Tomcat instance. To avoid the unavailability of resources, always restart the Tomcat instance quickly.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ```
     ssh -l ucas RESTAPISERVER -p1315
     ```

   - Avamar server

     ```
     ssh -l USERNAME RESTAPISERVER
     ```

2. Type the following command to switch to root:

   ```
   su -
   ```

3. Type the following command:

   ```
   /usr/local/avamar/bin/restserver.sh --stop
   ```

Results

The Avamar REST API server shuts down.

Manually starting

Start the Avamar REST API server from the command line.

Before you begin

Install the Avamar REST API software on an Avamar server or on a Backup and Recovery Manager server.

The Avamar REST API installer configures the Avamar REST API server to stop and start automatically when the host computer stops and starts. Complete this task to start the Avamar REST API server manually.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ssh -l ucas *RESTAPISERVER* -p1315
   - Avamar server

     ssh -l *USERNAME RESTAPISERVER*
2. Type the following command to switch to root:

   su -
3. Type the following command:

   /usr/local/avamar/bin/restserver.sh --start

Results

The Avamar REST API server starts.

Changing the provider password

Change the provider password that is sent when starting a session with the Avamar REST API server.

Before you begin

Install the Avamar REST API software on a Backup and Recovery Manager server or on an Avamar system.

When changing an Avamar REST API default property, change the value in the custom system properties file.

The custom system properties file is: `/usr/local/avamar/var/rest-api/server_data/restserver.properties`

NOTICE

Do not change the default system properties file.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:

- Backup and Recovery Manager server

  ssh -l ucas *RESTAPISERVER* -p1315

- Avamar server

  ssh -l *USERNAME RESTAPISERVER*

2. Type the following command to switch to root:

   su -

3. Type the following command:

   /usr/local/avamar/bin/restserver.sh --stop

   The Avamar REST API server shuts down.

4. Find the following property in the custom system properties file:

   `provider.password`

5. Replace the encrypted value of property with a plain-text password.

   The encrypted value consists of all the characters after the equal sign.

   NOTICE

   Do not encode or encrypt the new password value. When you restart the Avamar REST API server, the Avamar REST API server automatically encodes and encrypts the new password value.

   For example, in the key/value pair `provider.password=ENC(ZTfFCpttmkSeS/yoTv4bXZlYkFUXQc1F)`, replace `ENC(ZTfFCpttmkSeS/yoTv4bXZlYkFUXQc1F)` with a plain-text password.

6. Save and close the file.
7. Type the following command:

   /usr/local/avamar/bin/restserver.sh --start

Results

The Avamar REST API server starts up and, after a short time, the Avamar REST API server encodes and encrypts the password.

Changing the provider credentials

Change the provider username and password that are required to start a session with the Avamar REST API server.

Before you begin

Install the Avamar REST API software on a Backup and Recovery Manager server or on an Avamar system.

When changing an Avamar REST API default property, change the value in the custom system properties file.

The custom system properties file is: `/usr/local/avamar/var/rest-api/server_data/restserver.properties`

NOTICE

Do not change the default system properties file.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ssh -l ucas *RESTAPISERVER* -p1315
   - Avamar server

     ssh -l *USERNAME RESTAPISERVER*
2. Type the following command to switch to root:

   su -
3. Type the following command:

   /usr/local/avamar/bin/restserver.sh --stop

   The Avamar REST API server shuts down.
4. Find the following property in the custom system properties file:

   `provider.user=admin`
5. Change the property's value, as shown here:

   provider.user=*CUSTOM_USERNAME*
   where *CUSTOM_USERNAME* is the custom username.
6. Find the following property in the custom system properties file:

   `provider.password`
7. Replace the encrypted value of property with a plain-text password.

   The encrypted value consists of all the characters after the equal sign.

   NOTICE

   Do not encode or encrypt the new password value. When you restart the Avamar REST API server, the Avamar REST API server automatically encodes and encrypts the new password value.

   For example, in the key/value pair `provider.password=ENC(ZTfFCpttmkSeS/yoTv4bXZlYkFUXQc1F)`, replace `ENC(ZTfFCpttmkSeS/yoTv4bXZlYkFUXQc1F)` with a plain-text password.
8. Save and close the file.
9. Type the following command:

   /usr/local/avamar/bin/restserver.sh --start

Results

The Avamar REST API server requires the custom credentials when starting new sessions.

CHAPTER 3

Architecture

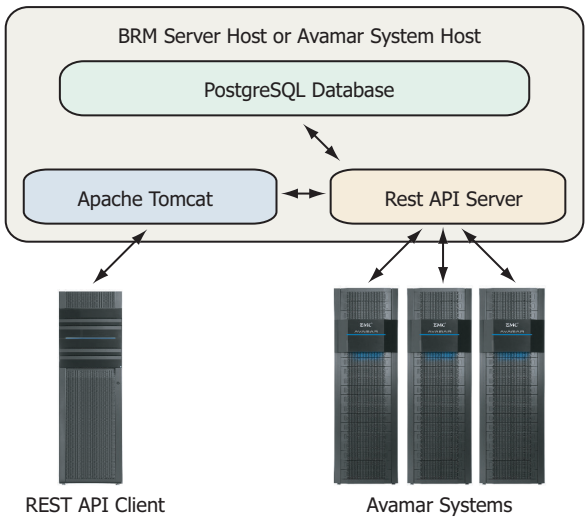
This chapter includes the following topics:

Architecture of the Avamar REST API

The Avamar REST API uses several components of the host EMC Backup and Recovery Manager server or of the host Avamar system.

The following figure represents the architecture of a standard deployment of the Avamar REST API server.

Figure 1 Components of the Avamar REST API architecture



The following table describes the lines of communication between the components.

Table 5 Descriptions of endpoint to endpoint communication

Endpoint	Endpoint	Description
Avamar REST API client	Apache Tomcat	All communications use Hypertext Transfer Protocol Secure (HTTPS). The Apache Tomcat web server listens on port 8543. The Apache Tomcat web server also serves the online API specifications using HTTP on port 8580.
Avamar REST API server	PostgreSQL database	The Avamar REST API server connects with the PostgreSQL database on port 5558.
Avamar REST API server	MCS on each Avamar server	The Avamar REST API uses MCSDK calls to communicate with the MCS on each Avamar server. MCS listens on port 9443.

The Avamar REST API installer places objects in the locations that are shown in the following table.

Table 6 Avamar REST API object locations

Avamar REST API object	Location
Application	/usr/local/avamar-tomcat/webapps/rest-api.war
API specification	/usr/local/avamar-tomcat/webapps/rest-api-doc.war

Table 6 Avamar REST API object locations (continued)

Avamar REST API object	Location
Server log file	/usr/local/avamar/var/rest-api/server_log/restserver.log
Default system properties file	/usr/local/avamar/lib/rest-api/restserver.properties
Custom system properties file	/usr/local/avamar/var/rest-api/server_data/restserver.properties
Scripts and utilities	/usr/local/avamar/bin/

NOTICE

Do not change the default system properties file. To change the system values that are set by default, change the custom system properties file.

CHAPTER 4

Concepts

This chapter includes the following topics:

Design goal

The primary goal of the Avamar REST API is to simplify the management of large numbers of Avamar systems. To achieve this goal the Avamar REST API focuses on the consumers of the data protection resources instead of focusing on the data protection infrastructure.

Infrastructure variations

In a typical service provider environment, the service provider dedicates some of the physical infrastructure to individual customers and shares some of the infrastructure between many customers. Avamar REST API simplifies the management of these infrastructure sharing variations.

The Avamar REST API does not require you to work directly with each Avamar system. Instead, you can abstract the systems into groups that meet your business requirements and write code that addresses these logical abstractions of the systems.

The following examples demonstrate the impact of achieving this design goal.

Example: On-demand backup

As a service provider you might have one customer, or tenant, who has 10,000 clients that are backed up to 10 Avamar systems. If the tenant wants to perform an on-demand backup of one of those clients, you don't need to know the Avamar system on which the client is backed up. You just launch the backup with a simple Avamar REST API call:

```
POST /client/CLIENT_URI/action/backup
```

The Avamar REST API server determines the Avamar system responsible for the client and directs the backup operation to that system.

Example: All activities

The same tenant, with 10,000 clients spread over 10 Avamar systems, would like a list of all of the backup jobs for the clients over the past 24 hours. A single Avamar REST API call provides the basis for that list:

```
GET /tenant/TENANT_URI/job
```

This single call replaces the need to individually query each of the 10 Avamar systems.

Example: Retention policy change

The same tenant wants to make changes to a retention policy being applied to thousands of clients that are spread over all 10 of the Avamar systems. A single Avamar REST API request changes the retention policy to the parameters contained in the request's payload. The Avamar REST API server automatically propagates the retention policy change to all relevant Avamar systems. The following Avamar REST API call initiates this action:

```
PUT /retention/RETENTION_URI
```

This simple PUT request replaces the requirement of individually changing the retention policy on each of the relevant Avamar systems.

Core concepts

Understanding several core concepts helps you to work with the Avamar REST API.

Data protection resource

The Avamar REST API uses the term "data protection resource" to refer to an Avamar system. When you configure the Avamar REST API server you provide information about the Avamar systems that the Avamar REST API server manages as data protection resources.

In defining a data protection resource, specify user credentials for an account with administrative access to that Avamar system. As a recommended best practice, you can create a dedicated administrative account on each Avamar system, such as **rest-api**. Then use that account when you configure the Avamar system as a data protection resource. Uniquely identifying the account makes it easy for you to determine the Avamar system on which operations originated.

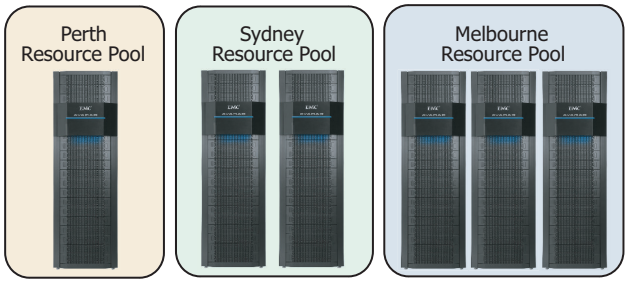
Resource pool

The term "resource pool" refers to a logical entity that represents a group of one or more data protection resources. A data protection resource is only available to use with the Avamar REST API after you assign it to a resource pool. When you configure the Avamar REST API server you specify which data protection resources are in each of the resource pools.

One method of assigning data protection resources to a resource pool uses the geographical location of the Avamar systems.

For example, consider a provider that has one Avamar system at a Perth data center, two Avamar systems at a Sydney data center, and three Avamar systems at a Melbourne data center. The provider might create three resource pools as shown here.

Figure 2 Geographically selected resource pools



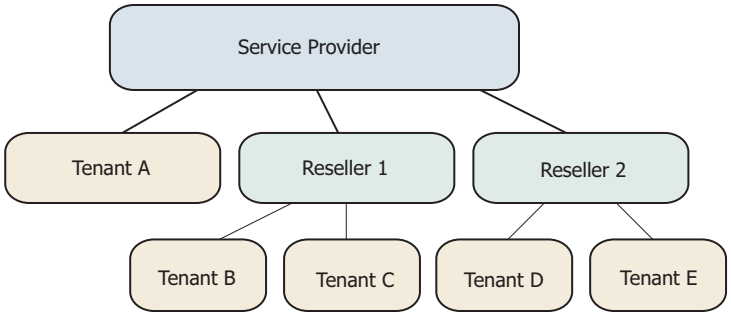
Tenant

A "tenant" represents a consumer of the data protection resources. A tenant folder is the top level container for the tenant. The tenant folder has metadata for the tenant, such as the tenant's account name, and also contains the resource shares and folders that are assigned to the tenant.

A service provider deployment usually has a separate tenant that is defined for each of the provider's customers and makes resource assignments that are based on the business requirements of each customer. In contrast, an enterprise deployment might allow a single tenant to have access to all the provider's data protection resources.

The Avamar REST API allows a tenant to have subtenants. A tenant with subtenants can create a tenant hierarchy and a service provider can have tenants who are resellers of the provider's resources. Those resellers can, in turn, manage their own customers as tenants. An example of this hierarchy appears in the following diagram.

Figure 3 Tenant hierarchy



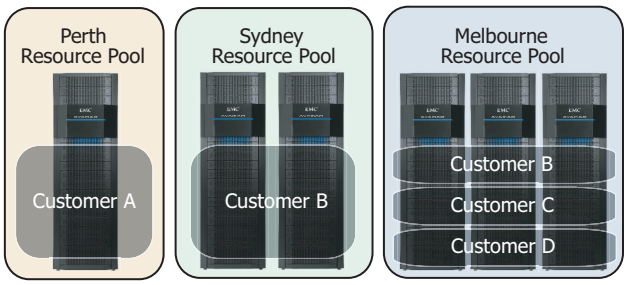
Resource share

A "resource share" is a logical entity that associates a tenant with data protection resources in a resource pool. The resource share can be associated with as few as one data protection resource in the resource pool or as many as all the data protection resources in the resource pool.

Through resource shares, you can assign several tenants to share a large resource pool, or dedicate a resource pool to a single tenant. Tenants can also be assigned multiple resource shares to meet their business requirements.

The following diagram depicts an example of the flexibility that is provided by resource shares for the customers: A, B, C, and D.

Figure 4 Resource share flexibility



When you create a resource share you specify the capacity of the resource pool that is available to the tenant. The capacity setting is not a hard quota. It is a measure that you can use to quantify the tenant's usage across multiple Avamar systems.

Folder

A "folder" is assigned to a resource share and provides a mapping between the tenant, the resource share, and a domain on each Avamar system in the resource share. A folder can only have one resource share assigned.

When you create a folder the Avamar REST API server creates a domain of the same name on each of the Avamar systems in the associated resource share.

You can create folders within folders that meet business requirements. These child folders can be associated with a separate resource share. Child folders that are not directly associated with a resource share inherit the resource share of the parent folder.

CHAPTER 5

Fundamentals

This chapter includes the following topics:

Representational state transfer

Avamar REST API uses the representational state transfer (REST) architectural style. REST is a stateless, client-server, API design model.

Client code interacts with the Avamar REST API server through standard HTTP request methods:

- GET
  Obtain information about an object,
- POST
  Create a new object.
- PUT
  Update an object.
- DELETE
  Delete an object.

To perform an operation, client code directs a request method at a URL and, for most API calls, sends a message body containing data associated with the request.

Every object on the Avamar REST API server is identified with a uniform resource identifier (URI), which is a unique ID for the object. In most cases, the URL used by a request includes an object's URI.

For example, for a client object with the following URI:

```
4702406e-d989-4058-a57b-c66ece0c4f37
```

You can get detailed information about the backup jobs that ran on the client that is represented by that client object by issuing the following request:

```
GET /client/4702406e-d989-4058-a57b-c66ece0c4f37/detail/job
```

The Avamar REST API server responds with an HTTP status code and, in most cases, with a payload that contains a detailed response to the request.

The HTTP status codes sent by the Avamar REST API server comply with HTTP/1.1 (RFC 2616, section 6 and section 10).

The Avamar REST API server supports sending the request body and the response body in either XML format or JSON format. The client making the request controls the body format.

Session log in and log out

To begin using the Avamar REST API your code starts a session with the Avamar REST API server. The session starts when your code logs into the Avamar REST API server and concludes when your code logs out.

Secure connection

Your code starts the session authentication process by initiating a SSL/TLS connection on port 8543 between the computer hosting your code and the Avamar REST API server. When SSL/TLS negotiation is finished and trust is established, your code's host computer and the Avamar REST API server share an encrypted channel for the session.

Log in

To start a session, your code uses the POST request method to send an HTTP authorization request to the Avamar REST API server at a fixed entry point. When the authorization request is successful, the Avamar REST API server sends back a custom HTTP response header and a session object.

Entry point

Your code sends the initial POST to a fixed entry point. The entry point URL is:

```
https://RESTAPISERVER:8543/rest-api/login
```

Where *RESTAPISERVER* is the IP address, or resolvable hostname, of the Avamar REST API server.

Request headers

The HTTP request headers your code sends to start a session are:

- `Accept`
- `Authorization`

The `Accept` header specifies a format to use for the response body. Your code can specify either XML or JSON, as follows:

- `Accept: application/xml`
- `Accept: application/json`

The `Authorization` header provides the authentication information for the session. To create the authentication information take the base64 encoded value of the concatenation of the user ID, a colon character, and the associated password. For example, the base64 encoded value of `admin:changeme` is `YWRtaW46Y2hhbmdlbWU=` and your code sends the following HTTP request header:

```
Authorization: Basic YWRtaW46Y2hhbmdlbWU=
```

X-Concerto-Authorization response header

The Avamar REST API server responds with the custom HTTP header `X-Concerto-Authorization` that contains the session's authentication token. Your code must include `X-Concerto-Authorization` and the token in every request sent during the session. The following is an example of this header:

```
X-Concerto-Authorization:
NzE2NjkzMDYtYjA2Yi00Y2IxLWI4MTMtMDIzNzQxMjM0OWZk
```

Session object

The Avamar REST API server sends a response payload that consists of a session object. The Avamar REST API server formats the payload as either XML or JSON, depending on the value of the Accept request header. The session object contains a user name and an access point.

The following is an example of a session object in XML format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Session xmlns="http://www.emc.com/concerto/v1.0"
href="https://lava7120:8543/rest-api/session" type="application/
xml,application/json">
<User name="admin"/>
<AccessPoint href="https://lava7120:8543/rest-api/admin/provider/
67c6200b-7fd2-48c2-ba64-5c5c92f21bb6"
id="67c6200b-7fd2-48c2-ba64-5c5c92f21bb6" name="Root"/>
</Session>
```

The following is an example of a session object in JSON format:

```
{
  "accessPoint" : [ {
    "href" : "https://lava7120:8543/rest-api/admin/provider/
67c6200b-7fd2-48c2-ba64-5c5c92f21bb6",
    "id" : "67c6200b-7fd2-48c2-ba64-5c5c92f21bb6",
    "name" : "Root"
  } ],
  "href" : "https://lava7120:8543/rest-api/session",
  "type" : "application/xml,application/json",
  "user" : {
```

```
    "name" : "admin"
  }
}
```

Log out

To end the session, your code sends a POST request to the log out URL, using the following format:

```
POST: https://RESTAPISERVER:8543/rest-api/logout
X-Concerto-Authorization: SESSIONTOKEN
```

Where:

- *RESTAPISERVER* is the IP address or resolvable hostname of the Avamar REST API host computer.
- *SESSIONTOKEN* is the session's authentication token.

The Avamar REST API server sends back the following response and closes the session:

```
HTTP 204 No Content
Connection: close
```

API call types

The Avamar REST API uses two types of API calls: synchronous and asynchronous. The Avamar REST API determines the type of API call used for an operation based on the nature of that operation.

Synchronous API calls

A synchronous API call blocks further code execution until the operation that the API call started completes. The Avamar REST API uses synchronous API calls for operations that are quick to perform. For example, the Avamar REST API uses a synchronous API call to add a policy to a data protection resource.

For a synchronous API call, the Avamar REST API server returns the result of the requested operation in the HTTP response. The HTTP response that the Avamar REST API server sends includes a response code that indicates the operation's final status. When an operation fails, the Avamar REST API server's response includes a response body with a `message` element. The `message` element describes the reason for the operation's failure.

Asynchronous API calls

The Avamar REST API server uses asynchronous API calls for operations that normally take longer to run. By using these calls the Avamar REST API avoids blocking the requestor from performing other tasks while the operation is in process.

For example, the Avamar REST API uses an asynchronous API call to initiate a client backup because a backup can take a significant amount of time. Instead of forcing the requestor to wait while the backup runs, the Avamar REST API uses an asynchronous API call. This permits the requestor to perform other tasks and to intermittently check on the status of the backup task.

The response to an asynchronous API call is almost immediate. The Avamar REST API server responds with a `task` element. For example, in response to an asynchronous API call to create a client, the Avamar REST API server might send the following JSON-formatted `task` element in the response body:

```
{
  "cancelable" : false,
  "cancelled" : false,
  "href" : "https://localhost:8543/rest-api/task/13fba299-150e-4d61-
a2ff-2bd7e926cf39",
  "id" : "13fba299-150e-4d61-a2ff-2bd7e926cf39",
  "name" : "task",
```

```
  "operation" : "Creating client under a user folder",
  "progress" : 0,
  "queueTime" : "2014-05-09T14:30:05.802Z",
  "startTime" : "2014-05-09T14:30:05.802Z",
  "state" : "QUEUED",
  "type" : "application/xml,application/json"
}
```

The `task` element includes an `href` element. The value of the `href` element is the URL that the requestor can use to reference the task.

Normally, the initial response sent by the Avamar REST API server includes a state element with the value of `QUEUED`. This indicates that the Avamar REST API server has the operation in the queue and that processing of the operation is pending.

A requestor can check the status of an asynchronous API call by performing the following API call:

```
GET /task/TASK_URI
```

The Avamar REST API server sends a response that is similar to the initial response. In subsequent responses, the `state` element of the response body moves through the following states:

- `QUEUED`
- `RUNNING`
- `SUCCESS`, `ERROR`, `CANCELED`, or `ABORTED`

Client allocation strategy

The Avamar REST API server automatically allocates new clients to Avamar systems based on the client allocation strategy you select. Select a built-in strategy to suit your business requirements.

The Avamar REST API provides the following built-in allocation strategies:

- BALANCED
- FREE_SPACE

The default allocation strategy is BALANCED. You can change the configuration to use the FREE_SPACE strategy.

BALANCED client allocation strategy
When configured to use the BALANCED client allocation strategy, the Avamar REST API server adds new clients to the Avamar system that has the least amount of client data. The strategy seeks to balance the amount of client data across all Avamar systems in a resource pool. You might select this strategy when your resource pools contain Avamar systems with similar capacities.

FREE_SPACE client allocation strategy
When configured to use the FREE_SPACE client allocation strategy, the Avamar REST API server adds new clients to the Avamar system that has the most free space. The strategy seeks to balance the amount of free space across all Avamar systems in a resource pool. You might select this strategy when your resource pools contain Avamar systems with different capacities.

Changing the built-in client allocation strategy

Change the client allocation strategy from the default built-in strategy to the alternative built-in strategy. The default built-in strategy is the BALANCED strategy. The alternative built-in strategy is the FREE_SPACE strategy.

When changing an Avamar REST API default property, change the value in the custom system properties file.

The custom system properties file is: `/usr/local/avamar/var/rest-api/server_data/restserver.properties`

NOTICE

Do not change the default system properties file.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ssh -l ucas *RESTAPISERVER* -p1315
   - Avamar server

     ssh -l *USERNAME RESTAPISERVER*
2. Type the following command to switch to root:

   su -
3. Type the following command:

   /usr/local/avamar/bin/restserver.sh --stop

   The Avamar REST API server shuts down.
4. Find the following property in the custom system properties file:

   `internalClientPlacement.strategy=BALANCED`
5. Change the property's value, as shown here:

   `internalClientPlacement.strategy=FREE_SPACE`
6. Save and close the file.
7. Type the following command:

   /usr/local/avamar/bin/restserver.sh --start

Results

The Avamar REST API server uses the FREE_SPACE strategy to allocate new clients to the Avamar systems.

CHAPTER 6

Advanced API Calls

This chapter includes the following topics:

Browse operations

The Avamar REST API provides two browse operations: client browse and backup browse.

When the Avamar REST API server receives either a client browse request or a backup browse request, the Avamar REST API server sends a response that includes a `BrowseContent` element in the response body. This element contains information about one or more objects located on the browse target. The objects can be files, directories, or custom objects.

Browsing a client

The Avamar REST API provides the ability to browse file systems or applications on a backup client. Browse the client to view available data and make data protection decisions.

Before you begin

Select a client and obtain the client's URI.

Browsing a client causes the associated Avamar system to initiate a connection with the client and return information about the client's current state.

Procedure

1. Obtain a list of plug-ins by running the following API call:

   GET /admin/provider/*PROVIDER_URI*/plugin

   The Avamar REST API server responds with a list of all plug-ins.
2. From the list of plug-ins, obtain the URI of a plug-in used with the client.
3. Browse the client by running the following API call and including a ClientBrowseRequest object:

   ```
   POST /client/CLIENT_URI/action/browse
   <ClientBrowseRequest  xmlns="http://www.emc.com/concerto/v1.0">
   <Plugin href="https://RESTAPISERVER:8543/rest-api/plugin/PLUG-
   IN_URI"/>
   <Path>/FULL_PATH</Path>
   </ClientBrowseRequest>
   ```

   The `plugin` element and the `path` element are required.

Results

The Avamar REST API server sends a browse response that includes a `BrowseContent` element in the response body.

After you finish

Process the browse response.

Browsing a backup

The Avamar REST API provides the ability to browse the contents of the backups taken by any of the managed Avamar systems.

Before you begin

Select a client and obtain the client's URI.

Procedure

1. Obtain a list of the client's backups by running the following API call:

   ```
   GET /client/CLIENT_URI/backup
   ```

   The Avamar REST API server returns a list of available backups for the client.
2. From the list of available backups, select the URI of a backup.
3. Browse the backup by using one of the following methods:
   - Browse the top-level of a regular backup

     ```
     POST /backup/BACKUP_URI/action/browse
     ```
   - Browse a specific path in the backup

     ```
     POST /backup/BACKUP_URI/action/browse
     <BackupBrowseRequest  xmlns="http://www.emc.com/concerto/v1.0">
     <Path>/FULL_PATH</Path>
     </BackupBrowseRequest>
     ```

Results

The Avamar REST API server sends a browse response that includes a `BrowseContent` element in the response body.

After you finish

Process the browse response.

Browse response

In response to a browse request, either a client browse requestor a backup browse request, the Avamar REST API server sends a browse response. The browse response includes a `BrowseContent` element in the response body.

The browse response includes the requested browse details in the `BrowseContent` element. In the `BrowseContent` element, the bulk of the requested information is contained in the `Metadata` element.

Metadata element

The `Metadata` element consists of a `kv` array. Each element in the `kv` array represents a file, a directory, or a custom object. Each element contains a series of key/value pairs containing information about the object represented by the element.

The contents of the `Metadata` element that the Avamar REST API server sends depend on the client, file system, and application being browsed. Browsing a file system returns different metadata from browsing a database. Browsing a Windows file system returns different metadata from browsing a Linux file system.

The following example shows a portion of a `kv` array showing a single element representing one file. Note the `name` element that contains the name of the file and the `metadatatype` element that provides the type of object represented by this `kv` array element.

```
"metadata" : [ {
    "kv" : [ {
      "k" : "links",
      "v" : {
        "value" : "1",
        "vtype" : "number"
      }
    }, {
      "k" : "inode",
      "v" : {
```

```
        "value" : "251",
        "vtype" : "number"
      }
    }, {
      "k" : "internal",
      "v" : {
        "value" : "0",
        "vtype" : "number"
      }
    }, {
      "k" : "date",
      "v" : {
        "value" : "2014-04-23 18:21:57",
        "vtype" : "dateTime"
      }
    }, {
      "k" : "size",
      "v" : {
        "value" : "100182",
        "vtype" : "number"
      }
    }, {
      "k" : "group",
      "v" : {
        "value" : "root",
        "vtype" : "string"
      }
    }, {
      "k" : "fstype",
      "v" : {
        "value" : "ext3",
        "vtype" : "string"
      }
    }, {
      "k" : "protection",
      "v" : {
        "value" : "-rwxr-xr-x",
        "vtype" : "string"
      }
    }, {
      "k" : "user",
      "v" : {
        "value" : "root",
        "vtype" : "string"
      }
    } ],
    "metadataType" : "file",
    "name" : "myfile"
  }
```


Headers element

The `BrowseContent` element also contains a `Headers` element. The `Headers` element maps extended labels to the key names found in `kv` array elements. This mapping is particularly useful when browsing applications with complex metadata.

The following example shows the contents of the `Headers` element after browsing a Linux file system.

```
metadataType : The type of object being described e.g. 'file'
name:   Name of file
user:  The user who owns this file e.g. root
group:  The group that owns this file
protection:   The unix permissions on this file e.g. -rwxr-xr-x
date:  The date the file was created
size: The size of the file
links: The number of links to this file
inode: The inode number of this file
date:  The date the file was created
```

```
size: The size of the file
fstype:  The filesystem type
```

Dataset creation

In the Avamar REST API, a dataset provides the information that is required to back up data, replicate data, or validate data. Create datasets to access the data you want to protect and to meet your business requirements.

You create a dataset by running the following API call:

```
POST /folder/FOLDER_URI/dataset
```

The POST must include a request body with a `Dataset` element. The `Dataset` element contains a `Mode` element and at least one `DatasetItem` element.

The dataset is created in the folder identified by *FOLDER_URI*.

Example 2 Dataset request body with one DatasetItem element

The following example shows a dataset request body with one `DatasetItem` element. The `DatasetItem` element references the "Windows File System" plug-in and a `DatasetTarget` value of `C:\`.

In XML format:

```
<Dataset  xmlns="http://www.emc.com/concerto/v1.0" name="TestDS">
    <Description></Description>
    <DatasetItem name ="Windows File System">
        <Plugin href="https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"/>
        <DatasetTarget name="target"><Value>C:\</Value></
DatasetTarget>
    </DatasetItem>
    <Mode>backup</Mode>
</Dataset>
```

In JSON format:

```
{
   "name": "TestDS",
   "Description": null,
   "DatasetItem": {
      "name": "Windows File System",
      "Plugin": {
         "href": "https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"
      },
      "DatasetTarget": {
         "name": "target",
         "Value": "C:\\"
      }
   },
   "Mode": "backup"
}
```

Mode element

The value of the `Mode` element determines the dataset's purpose. The `Mode` element must have one of the following values:

- Backup

- Replication
- Validation

DatasetItem element

Specify a `DatasetItem` element for every plug-in handled by the dataset. For example, to create a dataset that protects both Linux clients and Windows clients, include in the `Dataset` element a `DatasetItem` element for the "Linux File System" plug-in and a `DatasetItem` element for the "Windows File System" plug-in.

The `DatasetItem` element includes a required element and optional elements. The `Plugin` element is required.

Example 3 Dataset request body with two DatasetItem elements

The following example shows a dataset request body with two `DatasetItem` elements. The first `DatasetItem` element references the "Windows File System" plug-in and a `DatasetTarget` value of `C:\`.

The second `DatasetItem` element references the "Linux File System" plug-in and a `DatasetTarget` value of `All`.

When you specify `All` as the `DatasetTarget` value, the `DatasetItem` element targets all of the data in the file system.

In XML format:

```
<Dataset  xmlns="http://www.emc.com/concerto/v1.0" name="My New
Dataset">
    <Description></Description>
    <DatasetItem name ="Windows File System">
        <Plugin href="https://localhost:8543/rest-api/plugin/
03ffdd6d-6353-4246-8e4f-228ea7f62917"/>
        <DatasetTarget name="target"><Value>C:\</Value></
DatasetTarget>
    </DatasetItem>
    <DatasetItem name ="Linux File System">
        <Plugin href="https://localhost:8543/rest-api/plugin/87f7c7df-
fe00-4d62-87a7-8be7e5993ca3"/>
        <DatasetTarget name="target"><Value>ALL</Value></
DatasetTarget>
    </DatasetItem>
    <Mode>backup</Mode>
</Dataset>
```

In JSON format:

```
{
   "name": "My New Dataset",
   "Description": null,
   "DatasetItem": [
      {
         "name": "Windows File System",
         "Plugin": {
            "href": "https://localhost:8543/rest-api/plugin/
03ffdd6d-6353-4246-8e4f-228ea7f62917"
         },
         "DatasetTarget": {
            "name": "target",
            "Value": "C:\\"
         }
      },
      {
```

Example 3 Dataset request body with two DatasetItem elements (continued)

```
         "name": "Linux File System",
         "Plugin": {
            "href": "https://localhost:8543/rest-api/plugin/87f7c7df-
fe00-4d62-87a7-8be7e5993ca3"
         },
         "DatasetTarget": {
            "name": "target",
            "Value": "ALL"
         }
      }
   ],
   "Mode": "backup"
}
```

DatasetOption element

Optionally, use the `DatasetOption` elements within the `DatasetItem` element to specify options for the identified plug-in. Specify an option by including the unique `PluginOptionName` value for that option.

Obtain a list of the options for a plug-in, with the associated `PluginOptionName` values, by running the following API call:

```
GET /plugin/PLUG-IN_URI
```

Example 4 Dataset request body with DatasetOption elements

The following example shows a dataset request body with one `DatasetItem` element. The `DatasetItem` element references the "Windows File System" plug-in and a `DatasetTarget` value of `All`.

Notice that the `DatasetItem` element uses the name `MYDSITEMNAME` instead of `Windows File System`. You can use any text string for the name of the `DatasetItem` element as long as it is only used for one `DatasetItem` in the `Dataset` element. The `Plugin` element reference determines the associated plug-in.

In the example, two `DatasetOption` elements appear. The first `DatasetOption` element sets the verbose flag to `true`. The second `DatasetOption` element sets the backup label to `mylabel`.

In XML format:

```
<Dataset  xmlns="http://www.emc.com/concerto/v1.0" name="My Dataset">
    <Description></Description>
    <DatasetItem name ="MYDSITEMNAME">
        <Plugin href="https://localhost:8543/rest-api/plugin/
03ffdd6d-6353-4246-8e4f-228ea7f62917"/>
        <DatasetTarget name="target"><Value>ALL</Value></
DatasetTarget>
        <DatasetOption name="verbose"><Value>true</Value></
DatasetOption>
        <DatasetOption name="label"><Value>mylabel</Value></
DatasetOption>
    </DatasetItem>
    <Mode>backup</Mode>
</Dataset>
```

In JSON format:

Example 4 Dataset request body with DatasetOption elements (continued)

```
{
   "name": "My Dataset",
   "Description": null,
   "DatasetItem": {
      "name": "MYDSITEMNAME",
      "Plugin": {
         "href": "https://localhost:8543/rest-api/plugin/
03ffdd6d-6353-4246-8e4f-228ea7f62917"
      },
      "DatasetTarget": {
         "name": "target",
         "Value": "ALL"
      },
      "DatasetOption": [
         {
            "name": "verbose",
            "Value": "true"
         },
         {
            "name": "label",
            "Value": "mylabel"
         }
      ]
   },
   "Mode": "backup"
}
```

Elements in the DatasetItem element

The `DatasetItem` element includes one required element and several optional elements.

Table 7 Descriptions of the elements in the DatasetItem element

Element	Required	Description
Plugin	Yes	URL reference to the associated plug-in.
DatasetTarget	No	Data targeted by the `DatasetItem` element. Use `All` to include all data on the file system.
DatasetExclude	No	List of paths and files to exclude from the dataset.
DatasetInclude	No	List of paths and files which are identified by the values in the `DatasetExclude` element but are not excluded.
DatasetOption	No	Plug-in specific set of options that control how the dataset behaves for that plug-in.

Creating a dataset

Create a dataset to specify the data to backup, replicate or validate. Include in the dataset any options for handling the backed up data.

This task shows the addition of one `DatasetItem` element. Add additional `DatasetItem` elements to your `Dataset` element using the same method.

Procedure

1. Draft a skeleton `Dataset` element in XML format or in JSON format.

   In XML format:

   ```
   <Dataset  xmlns="http://www.emc.com/concerto/v1.0"
   name="MYDATASET">
   </Dataset>
   ```

   In JSON format:

   ```
   {
      "name": "MYDATASET"
   }
   ```

   where *MYDATASET* is the name you assign to the dataset.

2. Add the `Description` element.

   The description element can be empty. Here it is "My dataset test."

   In XML format:

   ```
   <Dataset  xmlns="http://www.emc.com/concerto/v1.0"
   name="MYDATASET">
       <Description>My dataset test.</Description>
   </Dataset>
   ```

   In JSON format:

   ```
   {
      "name": "MYDATASET",
      "Description": "My dataset test."
   }
   ```

3. Add the `Mode` element.

   The value of the `Mode` element must be `backup`, `replication`, or `validation`.

   In XML format:

   ```
   <Dataset  xmlns="http://www.emc.com/concerto/v1.0"
   name="MYDATASET">
       <Description>My dataset test.</Description>
       <Mode>backup</Mode>
   </Dataset>
   ```

   In JSON format:

   ```
   {
      "name": "MYDATASET",
      "Description": "My dataset test."
      "Mode": "backup"
   }
   ```

4. Add a skeleton `DatasetItem` element.

   The name of the `DatasetItem` element must be unique among all `DatasetItem` elements in the `Dataset` element. Here it is MYDSITEMNAME.

   In XML format:

   ```
   <Dataset  xmlns="http://www.emc.com/concerto/v1.0"
   name="MYDATASET">
       <Description>My dataset test.</Description>
       <DatasetItem name ="MYDSITEMNAME">
       </DatasetItem>
       <Mode>backup</Mode>
   </Dataset>
   ```

   In JSON format:

```
{
   "name": "MYDATASET",
   "Description": "My dataset test.",
   "DatasetItem": {
      "name": "MYDSITEMNAME"
   },
   "Mode": "backup"
}
```

5. Add a `Plugin` element.

   The `Plugin` element is a reference to a plug-in using a URL that includes the plug-in's URI.

   In XML format:

```
<Dataset  xmlns="http://www.emc.com/concerto/v1.0"
name="MYDATASET">
    <Description>My dataset test.</Description>
    <DatasetItem name ="MYDSITEMNAME">
        <Plugin href="https://localhost:8543/rest-api/plugin/PLUG-
IN_URI"/>
    </DatasetItem>
    <Mode>backup</Mode>
</Dataset>
```

   In JSON format:

```
{
   "name": "MYDATASET",
   "Description": "My dataset test.",
   "DatasetItem": {
      "name": "MYDSITEMNAME",
      "Plugin": {
         "href": "https://localhost:8543/rest-api/plugin/PLUG-
IN_URI"
      }
   },
   "Mode": "backup"
}
```

6. Add a `DatasetTarget` element.

   The `DatasetTarget` element is the full path to a directory to protect. Alternatively, the value can be `All` to protect the entire file system.

   Within the `DatasetItem` element you can add additional `DatasetTarget` elements to protect additional directories with the associated plug-in.

   The `DatasetTarget` element's `name` value must be unique in the `Dataset` element.

   In XML format:

```
<Dataset  xmlns="http://www.emc.com/concerto/v1.0"
name="MYDATASET">
    <Description>My dataset test.</Description>
    <DatasetItem name ="MYDSITEMNAME">
        <Plugin href="https://localhost:8543/rest-api/plugin/PLUG-
IN_URI"/>
        <DatasetTarget name="mytarget1"><Value>/usr/local/</
Value></DatasetTarget>
        <DatasetTarget name="mytarget2"><Value>/home/user/</
Value></DatasetTarget>
    </DatasetItem>
    <Mode>backup</Mode>
</Dataset>
```

   In JSON format:

```
{
   "name": "MYDATASET",
```

```
   "Description": "My dataset test.",
   "DatasetItem": {
      "name": "MYDSITEMNAME",
      "Plugin": {
         "href": "https://localhost:8543/rest-api/plugin/PLUG-
IN_URI"
      },
      "DatasetTarget": [
         {
            "name": "mytarget1",
            "Value": "/usr/local/"
         },
         {
            "name": "mytarget2",
            "Value": "/home/user/"
         }
      ]
   },
   "Mode": "backup"
}
```

7. (Optional) Add one or more of the `DatasetOption` elements supported by the plug-in.

   Use the `PluginOptionName` value when referencing an option.

   In XML format:

```
<Dataset  xmlns="http://www.emc.com/concerto/v1.0"
name="MYDATASET">
    <Description>My dataset test.</Description>
    <DatasetItem name ="MYDSITEMNAME">
        <Plugin href="https://localhost:8543/rest-api/plugin/PLUG-
IN_URI"/>
        <DatasetTarget name="mytarget1"><Value>/usr/local/</
Value></DatasetTarget>
        <DatasetTarget name="mytarget2"><Value>/home/user/</
Value></DatasetTarget>
        <DatasetOption name="verbose"><Value>true</Value></
DatasetOption>
        <DatasetOption name="label"><Value>mylabel</Value></
DatasetOption>
    </DatasetItem>
    <Mode>backup</Mode>
</Dataset>
```

   In JSON format:

```
{
   "name": "MYDATASET",
   "Description": "My dataset test.",
   "DatasetItem": {
      "name": "MYDSITEMNAME",
      "Plugin": {
         "href": "https://localhost:8543/rest-api/plugin/PLUG-
IN_URI"
      },
      "DatasetTarget": [
         {
            "name": "mytarget1",
            "Value": "/usr/local/"
         },
         {
            "name": "mytarget2",
            "Value": "/home/user/"
         }
      ],
      "DatasetOption": [
         {
            "name": "verbose",
            "Value": "true"
```

```
         },
         {
            "name": "label",
            "Value": "mylabel"
         }
      ]
   },
   "Mode": "backup"
}
```

8. (Optional) Add additional `DatasetItem` elements.
9. Save the dataset locally.
10. Create a dataset by running the following API call with the `Dataset` element in the request body:

    ```
    POST /folder/FOLDER_URI/dataset
    ```

Results

The Avamar REST API server creates the dataset in the specified folder.

Setting backups to go to a Data Domain storage system

Configure backups to go to a Data Domain storage system by setting `DatasetOption` elements in a dataset at the `DatasetItem` level.

Before you begin

Do the following:

- Determine the *PLUG-IN_URI* for each plug-in that will store data on a Data Domain storage system.
- Obtain the plug-in specific list of `DatasetOption` elements for each plug-in.
- Select only plug-ins that include the `ddr` option and the `ddr-index` option in the plug-in's list of `DatasetOption` elements.

Use the `ddr` option and the `ddr-index` option to direct backed up data to a Data Domain storage system.

Procedure

1. Draft a `Dataset` element that includes a `DatasetItem` element for each of the plug-ins that will store backed up data on a Data Domain storage system.
2. In the `DatasetItem` element for each of the selected plug-ins, include the following `DatasetOption` element:

   ```
   <DatasetOption name="ddr"><Value>true</Value></DatasetOption>
   ```

3. In the `DatasetItem` element for each of the selected plug-ins, include the following `DatasetOption` element:

   ```
   <DatasetOption name="ddr-index"><Value>n</Value></DatasetOption>
   ```

   where *n* is the index number that is assigned to a Data Domain storage system.

4. Save the `Dataset` element locally.
5. Run the following API call and include the `Dataset` element in the request body of the call:

   ```
   POST /folder/FOLDER_URI/dataset
   ```

Results

The Avamar REST API server creates the dataset in the specified folder. For backups using the dataset, the Avamar REST API server directs data from the selected plug-ins to the Data Domain storage system.

Example 5 Backup a Windows file system to a Data Domain storage system

In the following example, the dataset named `TestDS` has the `DatasetItem` element named `MYDSITEMNAME`. This `DatasetItem` element uses the Windows File System plug-in, backs up all data in the file system, and stores all backed up data on the index 1 Data Domain storage system.

In XML format:

```
<Dataset  xmlns="http://www.emc.com/concerto/v1.0" name="TestDS">
    <Description></Description>
    <DatasetItem name ="MYDSITEMNAME">
        <Plugin href="https://localhost:8543/rest-api/plugin/
03ffdd6d-6353-4246-8e4f-228ea7f62917"/>
        <DatasetTarget name="target"><Value>ALL</Value></
DatasetTarget>
        <DatasetOption name="ddr"><Value>true</Value></DatasetOption>
        <DatasetOption name="ddr-index"><Value>1</Value></
DatasetOption>
    </DatasetItem>
    <Mode>backup</Mode>
</Dataset>
```

In JSON format:

```
{
   "name": "TestDS",
   "Description": null,
   "DatasetItem": {
      "name": "MYDSITEMNAME",
      "Plugin": {
         "href": "https://localhost:8543/rest-api/plugin/
03ffdd6d-6353-4246-8e4f-228ea7f62917"
      },
      "DatasetTarget": {
         "name": "target",
         "Value": "ALL"
      },
      "DatasetOption": [
         {
            "name": "ddr",
            "Value": "true"
         },
         {
            "name": "ddr-index",
            "Value": "1"
         }
      ]
   },
   "Mode": "backup"
}
```

VMware

To manage VMware resources through the Avamar REST API, first add the resources to the Avamar REST API server by providing the required information. Once added, use the

VMware-specific actions provided by the Avamar REST API to manage the VMware resources.

Resources
Add the following VMware resources through the Avamar REST API:

- VMware vCenters
- VMware virtual machines
- Avamar proxy appliance for VMware

Actions
After adding VMware resources, manage them with the following actions:

- Add datastores to proxy appliances
- Remove datastores from proxy appliances
- Add proxy appliances to backup policies
- Perform on-demand backups of virtual machines
- Perform on-demand backups through proxy appliance policy associations
- Browse VMware backups at the image level
- Browse VMware backups at the file level
- Restore virtual machine data at the image level
- Restore virtual machine data at the file level

VMware vCenter

Add a VMware vCenter to the Avamar REST API server by providing information about the vCenter and including the URI of a folder. The vCenter is added to the referenced folder.

Add a vCenter
Add a vCenter by running the following API call:

```
POST /folder/FOLDER_URI/hypervisorManager
```

The POST must include a request body with a `HypervisorManager` element that includes the following elements:

- `Hostname`
  The hostname of the vCenter.
- `Username`
  A username for an administrator's account on the vCenter.
- `Password`
  The associated password for the administrator's account.
- `HypervisorManagerType`
  The hypervisor type. For VMware vCenter, use **`vCenter`**.

Optionally, other elements may be included.

Example 6 Request body used to add a VMware vCenter

The following example shows the request body used to add a vCenter with hostname **`vccc.asl.lab.emc.com`**, administrator's account **`administrator`**, and password **`changeme`**. Also shown are two optional elements: `Port` and `Description`. The `Port` element provides the port that the vCenter listens on. Here port **`443`** (the default) is

Example 6 Request body used to add a VMware vCenter (continued)

shown. The `Description` element provides identifying information for the entry. Here that element is empty.

In XML format:

```
<HypervisorManager xmlns="http://www.emc.com/concerto/v1.0"
name="vccc.asl.lab.emc.com">
   <Description></Description>
   <Hostname>vccc.asl.lab.emc.com</Hostname>
   <Port>443</Port>
   <Username>administrator</Username>
   <Password>changeme</Password>
   <HypervisorManagerType>vCenter</HypervisorManagerType>
</HypervisorManager>
```

In JSON format:

```
{
   "name": "vccc.asl.lab.emc.com",
   "Description": null,
   "Hostname": "vccc.asl.lab.emc.com",
   "Port": "443",
   "Username": "administrator",
   "Password": "changeme",
   "HypervisorManagerType": "vCenter"
}
```

Asynchronous API call

The Avamar REST API server handles the POST request as an asynchronous API call. In response, the Avamar REST API server provides a reference to the `task` element used to track the operation. Periodically check the status of the `task` element until the operation succeeds.

Successful operation

After successfully adding the vCenter to the specified folder, the folder includes the following two subfolders:

- Container Clients
- Virtual Machines

This result is identical to the result obtained by adding a vCenter to an Avamar system through Avamar Administrator.

Information about the vCenter

After successfully adding a vCenter, use the following API call to get all of information that the Avamar REST API server has for the vCenter:

```
GET /hypervisorManager/HVM_URI
```

In response to this API call, the Avamar REST API server provides all available elements contained in the `HypervisorManager` element for the vCenter. These elements include the `VmDatastore` element, which contains a list of the datastores available in the vCenter.

VMware virtual machines

Add VMware virtual machines by using the same API call that is used to add other clients. When adding a virtual machine, use additional elements to provide the required

information about the virtual machine. Add a virtual machine to the `Virtual Machines` folder within the associated vCenter's folder.

The Avamar REST API server manages virtual machines as Clients. Enable this capability by using additional elements to provide the extra information the Avamar REST API server requires.

API call

Add a virtual machine by running the following API call:

```
POST /folder/FOLDER_URI/client
```

Where *FOLDER_URI* references the `Virtual Machines` folder that was created when the vCenter associated with the virtual machine was added to the Avamar REST API server.

The POST must include a request body with a `Client` element that includes the following elements:

- `ClientExtensionType`
  For virtual machines, use the value `VmClient`.
- `VmClientExt`
  Identify the virtual machine by specifying the `DataCenter` and `VmFolder` elements, or specifying the `VmUUID` element. Also in this element, optionally include the `ChangedBlockTracking` element.

Example 7 A Client element that identifies a virtual machine by datacenter and folder

This example shows a `Client` element that could be used to add a virtual machine named "MyVM" to the Avamar REST API server. The virtual machine is located in the vCenter's "/Client DataCenter" datacenter and the "MyVMs" folder. Changed block tracking is enabled for the virtual machine client.

In XML format:

```
<Client  xmlns="http://www.emc.com/concerto/v1.0" name="MyVM">
    <Description></Description>
    <Contact></Contact>
    <Phone></Phone>
    <Email></Email>
    <Location></Location>
    <ClientExtensionType>VmClient</ClientExtensionType>
    <VmClientExt>
        <DataCenter>/Client DataCenter</DataCenter>
        <VmFolder>MyVMs</VmFolder>
        <ChangedBlockTracking>true</ChangedBlockTracking>
    </VmClientExt>
</Client>
```

In JSON format:

```
{
   "name": "MyVM",
   "Description": null,
   "Contact": null,
   "Phone": null,
   "Email": null,
   "Location": null,
   "ClientExtensionType": "VmClient",
   "VmClientExt": {
      "DataCenter": "/Client DataCenter",
      "VmFolder": "MyVMs",
      "ChangedBlockTracking": "true"
```

Example 7 A Client element that identifies a virtual machine by datacenter and folder (continued)

```
    }
}
```

ClientExtensionType
Use the `ClientExtensionType` element to specify that the client is a virtual machine (`VmClient`) or to specify that the client is a proxy appliance for VMware (`VmProxyClient`). When adding a virtual machine, use `VmClient`.

VmClientExt
Because a vCenter does not require virtual machine names to be unique, a virtual machine's name by itself is not enough for the Avamar REST API server to identify the virtual machine. Instead, provide additional identifying information by using the `VmClientExt` element.

In a `VmClientExt` element, use the vCenter's UUID for the virtual machine to identify the virtual machine by providing the UUID in a `VmUUID` element.

Alternatively, in a `VmClientExt` element use the `DataCenter` element and the `VmFolder` element to provide the vCenter's data center and folder for the virtual machine. The Avamar REST API server uses this information, in conjunction with the `Client` element's `name` attribute to identify the virtual machine. The value provided in the `name` attribute must exactly match the vCenter's name for the virtual machine.

The `VmClientExt` element also can include the optional `ChangedBlockTracking` element. Use the `ChangedBlockTracking` element with the value `true` to optimize backup performance by enabling changed block tracking on the virtual machine client.

Asynchronous API call
The Avamar REST API server handles the POST request as an asynchronous API call. In response, the Avamar REST API server provides a reference to the `task` element used to track the operation's status. Periodically check the status of the `task` element until the operation succeeds.

Adding a virtual machine by using the UUID

Use the vCenter UUID of a virtual machine to add the virtual machine to the Avamar REST API server.

Before you begin

Do the following:

- Add the vCenter that is associated with the virtual machine to the Avamar REST API server.
- Retain the URI of the vCenter's folder.
- Obtain the UUID used by the vCenter for the virtual machine.

Procedure

1. Draft a skeleton `Client` element.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="">
   </Client>
   ```

   In JSON format:

```
{
   "name": ""
}
```

2. Add a value to the `name` attribute.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": "VM_NAME"
   }
   ```

   where *VM_NAME* is the exact name for the virtual machine in vCenter.

3. Add a `ClientExtensionType` element with the value set to `VmClient`.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
       <ClientExtensionType>VmClient</ClientExtensionType>
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": "VM_NAME",
      "ClientExtensionType": "VmClient"
   }
   ```

4. Add a `VmClientExt` element with a `VmUUID` element that includes the virtual machine's UUID.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
       <ClientExtensionType>VmClient</ClientExtensionType>
       <VmClientExt>
           <VmUUID>00 11 22 33 44 55 66 77-88 99 aa bb cc dd ee ff</
   VmUUID>
       </VmClientExt>
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": "VM_NAME",
      "ClientExtensionType": "VmClient",
      "VmClientExt": {
         "VmUUID": "00 11 22 33 44 55 66 77-88 99 aa bb cc dd ee ff"
      }
   }
   ```

   where *00 11 22 33 44 55 66 77-88 99 aa bb cc dd ee ff* is a properly formatted vCenter UUID.

5. (Optional) Add a `ChangedBlockTracking` element with the value `true` to turn on changed block tracking for the virtual machine client.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
       <ClientExtensionType>VmClient</ClientExtensionType>
       <VmClientExt>
           <VmUUID>00 11 22 33 44 55 66 77-88 99 aa bb cc dd ee ff</
   VmUUID>
           <ChangedBlockTracking>true</ChangedBlockTracking>
   ```

```
    </VmClientExt>
</Client>
```

In JSON format:

```
{
   "name": "VM_NAME",
   "ClientExtensionType": "VmClient",
   "VmClientExt": {
      "VmUUID": "00 11 22 33 44 55 66 77-88 99 aa bb cc dd ee ff",
      "ChangedBlockTracking": "true"
   }
}
```

6. (Optional) Add other optional elements to the `Client` element.
7. Save the `Client` element locally.
8. Obtain the URI of the vCenter's `Virtual Machines` folder by running the following API call:

   ```
   GET /folder/FOLDER_URI/folders
   ```

   where *FOLDER_URI* is the URI of the folder in which the vCenter was added.

   The Avamar REST API server returns a list of the folders in the vCenter's folder, including the `Virtual Machines` folder.
9. Add the virtual machine by running the following API call with the `Client` element in the request body:

   ```
   POST /folder/FOLDER_URI/client
   ```

   where *FOLDER_URI* is the URI of the `Virtual Machines` folder.

Results

The Avamar REST API server adds the virtual machine to the specified folder.

Adding a virtual machine by using the datacenter and folder

Use the vCenter's datacenter and folder for a virtual machine to add the virtual machine to the Avamar REST API server.

Before you begin

Do the following:

- Add the vCenter that is associated with the virtual machine to the Avamar REST API server.
- Retain the URI of the vCenter's folder.
- Obtain the vCenter's datacenter and folder for the virtual machine.

Procedure

1. Draft a skeleton `Client` element.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="">
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": ""
   }
   ```

2. Add a value to the `name` attribute.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": "VM_NAME"
   }
   ```

   where *VM_NAME* is the exact name for the virtual machine in vCenter.

3. Add a `ClientExtensionType` element with the value set to `VmClient`.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
       <ClientExtensionType>VmClient</ClientExtensionType>
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": "VM_NAME",
      "ClientExtensionType": "VmClient"
   }
   ```

4. Add a `VmClientExt` element with a `DataCenter` element that includes the datacenter of the virtual machine.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
       <ClientExtensionType>VmClient</ClientExtensionType>
       <VmClientExt>
           <DataCenter>/DataCenter</DataCenter>
       </VmClientExt>
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": "VM_NAME",
      "ClientExtensionType": "VmClient",
      "VmClientExt": {
         "DataCenter": "/DataCenter"
      }
   }
   ```

   where */DataCenter* is the full path to the vCenter's datacenter that contains the virtual machine.

5. Add to the `VmClientExt` element the `VmFolder` element with the name of the vCenter folder that contains the virtual machine.

   In XML format:

   ```
   <Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
       <ClientExtensionType>VmClient</ClientExtensionType>
       <VmClientExt>
           <DataCenter>/DataCenter</DataCenter>
           <VmFolder>folder</VmFolder>
       </VmClientExt>
   </Client>
   ```

   In JSON format:

   ```
   {
      "name": "VM_NAME",
   ```

```
   "ClientExtensionType": "VmClient",
   "VmClientExt": {
      "DataCenter": "/DataCenter",
      "VmFolder": "folder"
   }
}
```

where *folder* is the vCenter folder that contains the virtual machine.

6. (Optional) Add a `ChangedBlockTracking` element with the value `true` to turn on changed block tracking for the virtual machine client.

   In XML format:

```
<Client  xmlns="http://www.emc.com/concerto/v1.0" name="VM_NAME">
    <ClientExtensionType>VmClient</ClientExtensionType>
    <VmClientExt>
        <DataCenter>/DataCenter</DataCenter>
        <VmFolder>folder</VmFolder>
        <ChangedBlockTracking>true</ChangedBlockTracking>
    </VmClientExt>
</Client>
```

   In JSON format:

```
{
   "name": "VM_NAME",
   "ClientExtensionType": "VmClient",
   "VmClientExt": {
      "DataCenter": "/DataCenter",
      "VmFolder": "folder",
      "ChangedBlockTracking": "true"
   }
}
```

7. (Optional) Add other optional elements to the `Client` element.
8. Save the `Client` element locally.
9. Obtain the URI of the vCenter's `Virtual Machines` folder by running the following API call:

```
GET /folder/FOLDER_URI/folders
```

   where *FOLDER_URI* is the URI of the folder in which the vCenter was added.

   The Avamar REST API server returns a list of the folders in the vCenter's folder, including the `Virtual Machines` folder.

10. Add the virtual machine by running the following API call with the `Client` element in the request body:

```
POST /folder/FOLDER_URI/client
```

   where *FOLDER_URI* is the URI of the `Virtual Machines` folder.

Results

The Avamar REST API server adds the virtual machine to the specified folder.

Proxy appliance for VMware

Add proxy appliances for VMware by using the same API call that is used to add other clients. When adding a proxy appliance, use additional elements to provide the required information about the proxy appliance.

The Avamar REST API server manages proxy appliances as clients. Enable this capability by using additional elements to provide the extra information the Avamar REST API server

requires. Proxy appliances cannot be added to a vCenter's folder. Instead, add proxy appliances to a folder that is hierarchically higher than the associated vCenter's folder.

API call

Add a proxy appliance by running the following API call:

```
POST /folder/FOLDER_URI/client
```

Where *FOLDER_URI* references a folder that is hierarchically higher than the folder that contains the associated vCenter.

The POST must include a request body with a `Client` element that includes the `ClientExtensionType` element with the value `VmProxyClient`.

NOTICE

After a proxy appliance is added to the Avamar REST API server, register and activate the proxy appliance with one of the managed Avamar systems. The proxy appliance cannot be configured to perform virtual machine backups until it is registered and activated with an Avamar system. In vCenter, open a console session on the proxy appliance to register and activate the proxy appliance with an Avamar system.

Identifying a proxy appliance

The Avamar REST API server identifies a proxy appliance by the `name` attribute of the `Client` element that is used to add the proxy appliance. The value of this attribute is the fully qualified domain name of the proxy appliance.

ClientExtensionType

Use the `ClientExtensionType` element to specify that the client is a virtual machine (`VmClient`) or to specify that the client is a proxy appliance for VMware (`VmProxyClient`). When adding a proxy appliance, use `VmProxyClient`.

Example 8 A Client element for a proxy appliance

This example shows a `Client` element for a proxy appliance named `myproxy.mycompany.com`.

In XML format:

```
<Client  xmlns="http://www.emc.com/concerto/v1.0"
name="myproxy.mycompany.com">
    <Description></Description>
    <Contact></Contact>
    <Phone></Phone>
    <Email></Email>
    <Location></Location>
    <ClientExtensionType>VmProxyClient</ClientExtensionType>
</Client>
```

In JSON format:

```
{
   "name": "myproxy.mycompany.com",
   "Description": null,
   "Contact": null,
   "Phone": null,
   "Email": null,
   "Location": null,
   "ClientExtensionType": "VmProxyClient",
}
```

Asynchronous API call
The Avamar REST API server handles the POST request as an asynchronous API call. In response, the Avamar REST API server provides a reference to the `task` element used to track the operation's status. Periodically check the status of the `task` element until the operation succeeds.

Information about the proxy appliance
After successfully adding a proxy appliance, use the following API call to get all of information that the Avamar REST API server has for the proxy appliance:

```
GET /client/CLIENT_URI
```

In response to this API call, the Avamar REST API server provides all available elements contained in the `Client` element for the proxy appliance. These elements include the proxy appliance's mapped datastores and groups.

Adding datastores to a proxy appliance

To prepare to backup a virtual machine through a proxy appliance, add the datastore associated with the virtual machine to the proxy appliance's list of protected datastores. An Avamar system only transfers a virtual machine's backup to a proxy appliance when the proxy appliance is configured to protect the datastore associated with the virtual machine.

Before you begin

Complete the following:

- Add the proxy appliance to the Avamar REST API server.
- Register and activate the proxy appliance with one of the managed Avamar systems.

Procedure

1. Use GET to get a list of the datastores managed by a vCenter.

   ```
   GET /hypervisorManager/HVM_URI
   ```

   The reply includes the vCenter's `vmDatastore` list with the URL for each managed datastore.

2. Draft a skeleton `VmDatastoreList` element.

   In XML format:

   ```
   <VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
   </VmDatastoreList>
   ```

   In JSON format:

   ```
   {}
   ```

3. Add a `VmDatastore` element containing a `Url` element.

   In XML format:

   ```
   <VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
       <VmDatastore>
           <Url></Url>
       </VmDatastore>
   </VmDatastoreList>
   ```

   In JSON format:

   ```
   {
      "VmDatastore": {
         "Url": null
   ```

```
   }
}
```

4. In the `Url` element add the URL of a datastore that is being added.

   In XML format:

```
<VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
    <VmDatastore>
        <Url>DS_URL</Url>
    </VmDatastore>
</VmDatastoreList>
```

   In JSON format:

```
{

   "VmDatastore": {

      "Url": "DS_URL"

   }

}
```

   where *DS_URL* is the vCenter's URL for the datastore. This is the value returned in the `url` element of the vCenter's `vmDatastore` list.

5. Add an additional `VmDatastore` element and `Url` element for every datastore being added.

   In XML format:

```
<VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
    <VmDatastore>
        <Url>DS_URL</Url>
    </VmDatastore>
    <VmDatastore>
        <Url>DS_URL</Url>
    </VmDatastore>
</VmDatastoreList>
```

   In JSON format:

```
{
   "VmDatastore": [
      {
         "Url": "DS_URL"
      },
      {
         "Url": "DS_URL"
      }
   ]
}
```

   Note that the JSON format uses an array of `Url` elements.

6. (Optional) Save the `VmDatastoreList` element locally.
7. Run the following API call and include the `VmDatastoreList` element.

```
PUT /client/CLIENT_URI/hypervisorManager/HVM_URI/action/
addDatastore
```

   where *CLIENT_URI* is the URI of the proxy appliance and *HVM_URI* is the URI of the vCenter associated with the datastores being added.

Results

The Avamar REST API server adds the datastores to the proxy appliance's list of protected datastores. The API call is synchronous. The Avamar REST API server responds with the proxy appliance's configuration, including a list of protected datastores.

Example 9 VmDatastoreList element to add two datastores to a proxy appliance

The following example shows a `VmDatastoreList` element that adds two datastores to a proxy appliance's list of protected datastores.

In XML format:

```
<VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
    <VmDatastore>
        <Url>ds:///vmfs/volumes/4e010b4b-e4b3547d-da84-001ec9b2b08b/</
Url>
    </VmDatastore>
    <VmDatastore>
        <Url>ds:///vmfs/volumes/4e010b4b-e4b3547d-da84-001ec9b2b3df/</
Url>
    </VmDatastore>
</VmDatastoreList>
```

In JSON format:

```
{
   "VmDatastore": [
      {
         "Url": "ds:///vmfs/volumes/4e010b4b-e4b3547d-
da84-001ec9b2b08b/"
      },
      {
         "Url": "ds:///vmfs/volumes/4e010b4b-e4b3547d-
da84-001ec9b2b3df/"
      }
   ]
}
```

Removing datastores from a proxy appliance

When a proxy appliance is no longer required to protect a datastore, remove the datastore from the proxy appliance's list of protected datastores.

Before you begin

Complete the following:

- Add the proxy appliance to the Avamar REST API server.
- Register and activate the proxy appliance with one of the managed Avamar systems.

Procedure

1. Use GET to get a list of the datastores managed by a vCenter.

   ```
   GET /hypervisorManager/HVM_URI
   ```

   The reply includes the vCenter's `vmDatastore` list with the URL for each managed datastore.

2. Draft a skeleton `VmDatastoreList` element.

   In XML format:

```
<VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
</VmDatastoreList>
```

In JSON format:

```
{}
```

3. Add a `VmDatastore` element containing a `Url` element.

   In XML format:

   ```
   <VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
       <VmDatastore>
           <Url></Url>
       </VmDatastore>
   </VmDatastoreList>
   ```

   In JSON format:

   ```
   {
      "VmDatastore": {
         "Url": null
      }
   }
   ```

4. In the `Url` element add the URL of a datastore that is being removed.

   In XML format:

   ```
   <VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
       <VmDatastore>
           <Url>DS_URL</Url>
       </VmDatastore>
   </VmDatastoreList>
   ```

   In JSON format:

   ```
   {

      "VmDatastore": {

         "Url": "DS_URL"

      }

   }
   ```

   where *DS_URL* is the vCenter's URL for the datastore. This is the value returned in the `url` element of the vCenter's `vmDatastore` list.

5. Add an additional `VmDatastore` element and `Url` element for every datastore being removed.

   In XML format:

   ```
   <VmDatastoreList  xmlns="http://www.emc.com/concerto/v1.0">
       <VmDatastore>
           <Url>DS_URL</Url>
       </VmDatastore>
       <VmDatastore>
           <Url>DS_URL</Url>
       </VmDatastore>
   </VmDatastoreList>
   ```

   In JSON format:

   ```
   {
      "VmDatastore": [
         {
            "Url": "DS_URL"
         },
         {
   ```

```
         "Url": "DS_URL"
      }
   ]
}
```

Note that the JSON format uses an array of `Url` elements.

6. (Optional) Save the `VmDatastoreList` element locally.
7. Run the following API call and include the `VmDatastoreList` element.

```
PUT /client/CLIENT_URI/hypervisorManager/HVM_URI/action/
removeDatastore
```

where *CLIENT_URI* is the URI of the proxy appliance and *HVM_URI* is the URI of the vCenter associated with the datastores being removed.

Results

The Avamar REST API server removes the datastores from the proxy appliance's list of protected datastores. The API call is synchronous. The Avamar REST API server responds with the proxy appliance's configuration, including a list of protected datastores.

Adding proxy appliances to a backup policy

To use a proxy appliance with a backup policy, add the proxy appliance to the backup policy's list of available proxy appliances.

Before you begin

Do the following:

- Obtain the URI of a backup policy.
- Obtain the URL for each proxy appliance being added.

Running a backup of virtual machines through a backup policy requires that the backup policy lists at least one proxy appliance. Otherwise, the backup fails with a message that no proxy can be found.

Procedure

1. Draft a skeleton `ReferenceList` element.

   In XML format:

```
<ReferenceList  xmlns="http://www.emc.com/concerto/v1.0">
    <Reference />
</ReferenceList>
```

   In JSON format:

```
{
   "Reference": null
}
```

2. Add a URL for each proxy appliance being added.

   In XML format:

```
<ReferenceList  xmlns="http://www.emc.com/concerto/v1.0">
    <Reference href="PROXY_URL" />
    <Reference href="PROXY_URL" />
</ReferenceList>
```

   In JSON format:

```
{
   "Reference": [
      {
```

```
            "href": "PROXY_URL"
         },
         {
            "href": "PROXY_URL"
         }
      ]
}
```

Note that the JSON format uses an array of `Reference` elements.

3. (Optional) Save the `ReferenceList` element locally.
4. Run the following API call and include the `ReferenceList` element.

```
PUT  /policy/POLICY_URI/action/addVmProxy
```

Results

The Avamar REST API server adds the proxy appliances to the backup policy. The API call is synchronous. The Avamar REST API server responds with the backup policy's configuration, including a list of associated proxy appliances.

Example 10 ReferenceList element to add two proxy appliances to a backup policy

In the following example, the `ReferenceList` element is drafted to add proxy appliances with the following reference URLs:

- https://localhost:8543/rest-api/client/09fad7f7-3f74-4cc7-9bf8-201f7275c4bf
- https://localhost:8543/rest-api/client/09fad7f7-3f74-4cc7-9bf8-201f7275cd2g

In XML format:

```
<ReferenceList  xmlns="http://www.emc.com/concerto/v1.0">
    <Reference href="https://localhost:8543/rest-api/client/
09fad7f7-3f74-4cc7-9bf8-201f7275c4bf" />
    <Reference href="https://localhost:8543/rest-api/client/
09fad7f7-3f74-4cc7-9bf8-201f7275cd2g" />
</ReferenceList>
```

In JSON format:

```
{
   "Reference": [
      {
         "href": "https://localhost:8543/rest-api/client/
09fad7f7-3f74-4cc7-9bf8-201f7275c4bf"
      },
      {
         "href": "https://localhost:8543/rest-api/client/
09fad7f7-3f74-4cc7-9bf8-201f7275cd2g"
      }
   ]
}
```

Removing proxy appliances from a backup policy

Remove proxy appliances from a backup policy's list of available proxy appliances.

Before you begin

Do the following:

- Obtain the URI of a backup policy.

- Obtain the URL for each proxy appliance being removed.

Procedure

1. Draft a skeleton `ReferenceList` element.

   In XML format:

   ```
   <ReferenceList  xmlns="http://www.emc.com/concerto/v1.0">
       <Reference />
   </ReferenceList>
   ```

   In JSON format:

   ```
   {
      "Reference": null
   }
   ```

2. Add a URL for each proxy appliance being removed.

   In XML format:

   ```
   <ReferenceList  xmlns="http://www.emc.com/concerto/v1.0">
       <Reference href="PROXY_URL" />
       <Reference href="PROXY_URL" />
   </ReferenceList>
   ```

   In JSON format:

   ```
   {
      "Reference": [
         {
            "href": "PROXY_URL"
         },
         {
            "href": "PROXY_URL"
         }
      ]
   }
   ```

   Note that the JSON format uses an array of `Reference` elements.

3. (Optional) Save the `ReferenceList` element locally.
4. Run the following API call and include the `ReferenceList` element.

   ```
   PUT  /policy/POLICY_URI/action/removeVmProxy
   ```

Results

The Avamar REST API server removes the proxy appliances from the backup policy. The API call is synchronous. The Avamar REST API server responds with the backup policy's configuration, including a list of associated proxy appliances.

On-demand virtual machine backups

Use the Avamar REST API to perform on-demand backups of an individual virtual machine and to perform on-demand backups of groups of virtual machines through a backup policy.

On-demand backups of an individual virtual machine

Use the following API call to perform an on-demand backup of an individual virtual machine:

```
POST /client/CLIENT_URI/action/backup
```

where *CLIENT_URI* is the URI of a virtual machine client.

The POST request must include a `BackupRequest` element that contains the specific elements in the following table.

Table 8 Required elements in a BackupRequest for an individual virtual machine

Element	Description
`Plugin`	Reference to the plug-in to use in the backup. This is one of the following for a VMware virtual machine: • Windows VMware Image • Linux VMware Image
`Retention`	Reference to the retention object that will define how long the backed up data is retained.
`Datasource`	Describes the data to backup. For a full VMware image backup, set the value to `All`.

Example 11 BackupRequest element for backing up an individual virtual machine

In this example, the `Datasource` element has the value `All`, indicating a full VMware image backup.

In XML format:

```
<BackupRequest  xmlns="http://www.emc.com/concerto/v1.0" >
    <DataSource><Source>All</Source></DataSource>
    <Plugin href="https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171" />
    <Retention href="https://localhost:8543/rest-api/retention/
a416939d-ce8d-4873-abfa-4bd2e8b22624" />
</BackupRequest>
```

In JSON format:

```
{
   "DataSource": {
      "Source": "All"
   },
   "Plugin": {
      "href": "https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"
   },
   "Retention": {
      "href": "https://localhost:8543/rest-api/retention/a416939d-
ce8d-4873-abfa-4bd2e8b22624"
   }
}
```

On-demand backups through a backup policy

An on-demand backup through a backup policy requires a correctly configured backup policy. Add all relevant proxy appliances to the backup policy before using the policy to perform backup.

Use the following API call to initiate a backup based on a backup policy:

```
POST /policy/POLICY_URI/action/backup
```

where POLICY_URI is the URI of the backup policy.

This API call does not require a request body.

Asynchronous API call

The Avamar REST API server handles an on-demand backup POST request as an asynchronous API call. In response, the Avamar REST API server provides a reference to the `task` element used to track the operation's status. Periodically check the status of the `task` element until the operation succeeds.

When the `task` element's status changes to `RUNNING`, the backup has started.

Continue to track the backup through the `task` element, or use one of the backup monitoring calls available through the Avamar REST API.

Running an on-demand backup of a virtual machine

Use an API call with a `BackupRequest` element in the request body to run an on-demand backup of a virtual machine.

Before you begin

Do the following:

- Obtain the reference URL for the plug-in being used for the backup.
- Obtain the reference URL for the retention object being used for the backup.
- Obtain the URI of the virtual machine client.

Procedure

1. Draft a skeleton `BackupRequest` element.

   In XML format:

   ```
   <BackupRequest  xmlns="http://www.emc.com/concerto/v1.0" >
       <DataSource><Source></Source></DataSource>
       <Plugin href="" />
       <Retention href="" />
   </BackupRequest>
   ```

   In JSON format:

   ```
   {
      "DataSource": {
         "Source": null
      },
      "Plugin": {
         "href": ""
      },
      "Retention": {
         "href": ""
      }
   }
   ```

2. Add a value to the `Datasource` element.

   For a full VMware image backup, use `All`.

   In XML format:

   ```
   <BackupRequest  xmlns="http://www.emc.com/concerto/v1.0" >
       <DataSource><Source>ALL</Source></DataSource>
       <Plugin href="" />
       <Retention href="" />
   </BackupRequest>
   ```

   In JSON format:

   ```
   {
      "DataSource": {
         "Source": "All"
   ```

```
   },
   "Plugin": {
      "href": ""
   },
   "Retention": {
      "href": ""
   }
}
```

3. Add a plug-in reference.

   In XML format:

```
<BackupRequest  xmlns="http://www.emc.com/concerto/v1.0" >
    <DataSource><Source>All</Source></DataSource>
    <Plugin href="PLUG-IN_URL" />
    <Retention href="" />
</BackupRequest>
```

   In JSON format:

```
{
   "DataSource": {
      "Source": "All"
   },
   "Plugin": {
      "href": "PLUG-IN_URL"
   },
   "Retention": {
      "href": ""
   }
}
```

4. Add a retention object reference.

   In XML format:

```
<BackupRequest  xmlns="http://www.emc.com/concerto/v1.0" >
    <DataSource><Source>All</Source></DataSource>
    <Plugin href="PLUG-IN_URL" />
    <Retention href="RETENTION_URL" />
</BackupRequest>
```

   In JSON format:

```
{
   "DataSource": {
      "Source": "All"
   },
   "Plugin": {
      "href": "PLUG-IN_URL"
   },
   "Retention": {
      "href": "RETENTION_URL"
   }
}
```

5. (Optional) Add other elements to the `BackupRequest` element.
6. (Optional) Save the `BackupRequest` element locally.
7. Run the following API call and include the `BackupRequest` element.

```
POST /client/CLIENT_URI/action/backup
```

Results

The Avamar REST API server handles the POST request as an asynchronous API call. In response, the Avamar REST API server provides a reference to the `task` element used to

track the operation. Periodically check the status of the `task` element until the operation succeeds.

Running an on-demand backup of virtual machines through a policy

Use an API call with the URI of a virtual machine backup policy to initiate a backup of the virtual machines.

Before you begin

Correctly configure a backup policy for virtual machines, including adding all relevant proxy appliances.

Procedure

1. Use one of the Avamar REST API calls to obtain the URI of the policy being used.

   For example, to obtain a list of the policies available for a folder, run the following API call:

   ```
   GET /folder/FOLDER_URI/detail/policy
   ```

2. Run the following API call to initiate the policy-based backup:

   ```
   POST /policy/POLICY_URI/action/backup
   ```

Results

The Avamar REST API server handles the POST request as an asynchronous API call. In response, the Avamar REST API server provides a reference to the `task` element used to track the operation. Periodically check the status of the `task` element until the operation succeeds.

Virtual machine browse operations

The Avamar REST API browse action can be used to view information about the contents of virtual machine backups. Depending on the content of the request body, the browse action can be used to view information about the virtual disk images in a backup or to view information about the individual files in a backup.

Virtual machine backup browse operations use the same API call and same top level request body element as the backup browse operations for other clients. However, the content of the Avamar REST API server's response is determined by the nature of the virtual machine client and the elements contained in the request body element.

In all cases, use the following API call to browse a virtual machine backup:

```
POST /backup/BACKUP_URI/action/browse
```

Include in the request a `BackupBrowseRequest` element.

The elements in the `BackupBrowseRequest` element determine whether the Avamar REST API server responds with information about the virtual disk images in a backup or responds with information about the individual files in a backup.

Browse virtual machine images

View information about the virtual disks contained in virtual machine backups. Obtain information about the virtual machine images in a backup that can be used when restoring virtual machine images.

API call

Use the following API call to view information about the virtual disk images in a virtual machine backup:

```
POST /backup/BACKUP_URI/action/browse
```

Include the following `BackupBrowseRequest` element in the request. No additional elements are required.

In XML format:

```
<BackupBrowseRequest  xmlns="http://www.emc.com/concerto/v1.0">
</BackupBrowseRequest>
```

In JSON format:

```
{}
```

The Avamar REST API server responds with a browse response that includes `metadata` elements that contain information about the virtual disk images in the backup.

Example 12 Excerpt from image level browse response

The following example is an excerpt of a JSON-formatted browse response that shows one of the `metadata` elements contained in the Avamar REST API server's response to an image browse request.

```
"metadata" : [ {
  "metadataType" : "VMDISK",
  "name" : "Hard disk 1 - [datastore1] vm-clicore-109_ New NetWorker
Server/vm-clicore-109_ New NetWorker Server.vmdk"
} ],
"
```

Browse virtual machine files

View information about the files and directories contained in a virtual machine backup. Obtain information about the file system that can be used when restoring a virtual machine's files.

API call

Use the following API call to view information about individual files in a virtual machine backup:

```
POST /backup/BACKUP_URI/action/browse
```

Include a `BackupBrowseRequest` element that contains a `GranularBrowse` element. The value of the `GranularBrowse` element determines the file system information that the Avamar REST API server provides in the response.

Top level browse information

To obtain top level browse information set the value of the `GranularBrowse` element to `true`, as shown here.

In XML format:

```
<BackupBrowseRequest  xmlns="http://www.emc.com/concerto/v1.0">
    <GranularBrowse>true</GranularBrowse>
</BackupBrowseRequest>
```

In JSON format:

```
{
   "GranularBrowse": "true"
}
```

The Avamar REST API server responds with the top-level browse information for the backup. For a virtual machine with a Windows guest operating system the top level response is a logical reference to the disk, as shown in JSON format here:

```
"metadataType" : "dir",
    "name" : "[Disk#1]"
```

File level browse information

To obtain file level browse information for an individual disk in the backup, include a `Path` element and set the value to the path of a directory on that disk. Include in the path statement the disk name that is provided by a top level browse request.

In XML format:

```
<BackupBrowseRequest  xmlns="http://www.emc.com/concerto/v1.0">
    <Path>DISK_NAME/FULL_PATH</Path>
    <GranularBrowse>true</GranularBrowse>
</BackupBrowseRequest>
```

In JSON format:

```
{
   "Path": "DISK_NAME/FULL_PATH",
   "GranularBrowse": "true"
}
```

where *DISK_NAME* is the value of the name element contained in the top level browse information of the virtual machine image.

Example 13 File level browse request for a Windows guest operating system

The following example provides the `BackupBrowseRequest` element to request file level browse information for the \Users folder on a Windows guest operating system located on Disk#1.

In XML format:

```
<BackupBrowseRequest  xmlns="http://www.emc.com/concerto/v1.0">
    <Path>[Disk#1]\Users</Path>
    <GranularBrowse>true</GranularBrowse>
</BackupBrowseRequest>
```

In JSON format:

```
{
   "Path": "[Disk#1]\\Users",
   "GranularBrowse": "true"
}
```

Virtual machine restore operations

The Avamar REST API restore action can be used to restore data to a virtual machine at the image level or at the file level. Both operations use the same API call. The difference is in the content of the request body sent with the API call.

Identifying a backup

Use the following API call to get a list of the available backups for a client:

```
POST /client/CLIENT_URI/detail/backup
```

The Avamar REST API server responds with a list of the available backups, and lists a URI for each backup. Use the listed URI to identify a backup for a restore operation.

API call

Use the following API call to initiate an image level or file level restore:

```
POST /backup/BACKUP_URI/action/restore
```

The API call must include the `RestoreRequest` element in the request body.

Request body
The API call that initiates a virtual machine restore operation includes a `RestoreRequest` element that defines the restore request. The `RestoreRequest` element always includes the elements shown here.

In XML format:

```
<RestoreRequest  xmlns="http://www.emc.com/concerto/v1.0" >
     <Plugin href="PLUG-IN_URL"/>
     <DestClient>
     </DestClient>
</RestoreRequest>
```

In JSON format:

```
{
   "Plugin": {
      "href": "PLUG-IN_URL"
   },
   "DestClient": {}
}
```

The elements within the `DestClient` element and additional elements provided in the `RestoreRequest` element determine the nature of the restore operation request.

Elements in a virtual machine RestoreRequest

Image level restore
The following table provides descriptions of the elements of a virtual machine restore request for an image level restore action.

Table 9 Elements in a request for an image level restore

Element	Description
Plugin	Reference to the plug-in to use for the restore. For a virtual machine restore operation this is one of the following: • Windows VMware Image • Linux VMware Image
DestClient	Element containing elements that provide information about the restore target.
VmIdentification	Element containing elements that provide information about an image level restore target.
Name	Name of the target virtual machine.
HypervisorManager	Reference to the vCenter associated with the restore target.
Datacenter	vCenter path for the datacenter associated with the restore target.
EsxHost	Name of the ESX host associated with the restore target.
Datastore	Name of the datastore associated with the restore target.
ChangedBlockTracking	Determines whether the full image is restored, or just the blocks that have changed since the last backup. The default is `false`, which specifies a full image restore. Set to `true` to restore just the changed blocks.

Example 14 RestoreRequest element for an image level restore

The following example contains a `RestoreRequest` element to include in the request body of an image level restore request.

In XML format:

```
<RestoreRequest  xmlns="http://www.emc.com/concerto/v1.0" >
    <Plugin href="https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"/>
    <DestClient>
        <VmIdentification>
           <Name>TestRestore2</Name>
           <HypervisorManager href="https://localhost:8543/rest-api/
hypervisorManager/4a402115-c7f9-4522-a71e-0d272b7ae38f"/>
           <Datacenter>/Client DataCenter</Datacenter>
           <EsxHost>esx-cc27.asl.lab.emc.com</EsxHost>
           <Datastore>datastore1(2)</Datastore>
           <ChangedBlockTracking>True</ChangedBlockTracking>
         </VmIdentification>
    </DestClient>
</RestoreRequest>
```

In JSON format:

```
{
   "Plugin": {
      "href": "https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"
   },
   "DestClient": {
      "VmIdentification": {
         "Name": "TestRestore2",
         "HypervisorManager": {
            "href": "https://localhost:8543/rest-api/
hypervisorManager/4a402115-c7f9-4522-a71e-0d272b7ae38f"
         },
         "Datacenter": "/Client DataCenter",
         "EsxHost": "esx-cc27.asl.lab.emc.com",
         "Datastore": "datastore1(2)",
         "ChangedBlockTracking": "True"
      }
   }
}
```

File level restore

The following table provides descriptions of the elements of a virtual machine restore request for a file level restore action.

Table 10 Elements in a request for a file level restore

Element	Description
Plugin	Reference to the plug-in to use for the restore. For a virtual machine restore operation this is one of the following: • Windows VMware Image • Linux VMware Image
BackupSource	Full path in the backup for the file that is being restored. The path statement must include the name of the disk.

Table 10 Elements in a request for a file level restore (continued)

Element	Description
DestinationPath	Full path on the target virtual machine.
DestClient	Element that contains the Client element that references the target virtual machine.
Client	Reference for the target virtual machine.
FileLevelRestore	Determines whether the Avamar REST API server performs a file level restore. To enable a file level restore, set the value to `true`.
Username	Username for an account with authorization to restore files to the target location.
Password	Password for the account with authorization to restore files to the target location.

Example 15 RestoreRequest element for a file level restore

The following example contains a `RestoreRequest` element to include in the request body of a file level restore request.

In XML format:

```
<RestoreRequest  xmlns="http://www.emc.com/concerto/v1.0" >
     <Plugin href="https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"/>
     <BackupSource>[Disk#1]\Program Files\Program Files\avs\bin
\avtar.exe  </BackupSource>
     <DestinationPath>C:\temp</DestinationPath>
     <DestClient>
        <Client href="https://localhost:8543/rest-api/client/
983bc30e-5f7f-45da-b846-1733419f7c8d" />
     </DestClient>
     <FileLevelRestore>true</FileLevelRestore>
     <Username>administrator</Username>
     <Password>changeme</Password>
</RestoreRequest>
```

In JSON format:

```
{
   "Plugin": {
      "href": "https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"
   },
   "BackupSource": "[Disk#1]\\Program Files\\Program Files\\avs\\bin\
\avtar.exe",
   "DestinationPath": "C:\\temp",
   "DestClient": {
      "Client": {
         "href": "https://localhost:8543/rest-api/client/
983bc30e-5f7f-45da-b846-1733419f7c8d"
      }
   },
   "FileLevelRestore": "true",
   "Username": "administrator",
   "Password": "changeme"
}
```

CHAPTER 7

Troubleshooting

This chapter includes the following topics:

Troubleshooting an Avamar REST API installation test failure

Find and fix a problem that you encountered when testing the Avamar REST API software installation.

Before you begin

Complete the installation of the Avamar REST API software without any RPM error messages.

SYMPTOM:

In testing the installation of the Avamar REST API software, you follow the recommendation to use a `curl` command. The curl tool does not display a `201 Created` HTTP response header and a session object.

Procedure

1. Log in to a secure shell (SSH) session on the target computer by typing the command format that is correct for the type of data protection server software:
   - Backup and Recovery Manager server

     ssh -l ucas *RESTAPISERVER* -p1315
   - Avamar server

     ssh -l *USERNAME RESTAPISERVER*
2. Type the following command to switch to root:

   su -
3. Determine if the problem is with Apache Tomcat (Tomcat) by checking the log.

   View the current log located in `/usr/local/avamar/var/rest-api/server_log/restserver.log`.
4. If you see any issues with Tomcat, restart Tomcat, by typing the following commands:

   /usr/local/avamar/bin/restserver.sh --stop
   /usr/local/avamar/bin/restserver.sh --start
5. After Tomcat restarts, retry the `curl` command.

   Tomcat normally takes several minutes to start up and initialize the Avamar REST API.
6. If the test still fails, test the connection to the PostgreSQL database by typing the following command:

   psql –Uadmin –p5558 rest-api
   If the PostgreSQL `Welcome` message appears, type **\q** to exit the connection. If the `Welcome` message does not appear, complete the next steps.
7. Restart Tomcat, by typing the following commands:

   /usr/local/avamar/bin/restserver.sh --stop
   /usr/local/avamar/bin/restserver.sh --start

   The Avamar REST API server restarts PostgreSQL and repopulates the database with the Avamar REST API schema information and tables.

8. Retry the `curl` command.

Troubleshooting a failed request

As you develop code to work with the Avamar REST API you may encounter instances where the code fails to receive the expected response. To determine what went wrong, review the HTTP response code and examine the Avamar REST API log.

Before you begin

Do the following:

- Complete the installation of the Avamar REST API software without any errors.
- Confirm that the installation is correct by successfully using the `curl` tool.

SYMPTOM:

You are testing new code by sending an Avamar REST API request to the Avamar REST API server. The request fails.

Procedure

1. Examine the HTTP status code that you receive from the Avamar REST API server.

   The HTTP status code provides an indication of how the Avamar REST API server handled the code's request.

   The status code sufficiently identified the problems in the code. If not, continue to the next step.

2. Examine the Avamar REST API log at: `/usr/local/avamar/var/rest-api/server_log/restserver.log`.

   Change the code to correct any errors that are found in the Avamar REST API log.

APPENDIX A

Known Problems and Limitations

This appendix includes the following topics:

Replication without policy fails to Avamar server version 7.1.x

Trying to replicate data from a client to an Avamar server running versions 7.1.x without using a policy fails with an HTTP status code 200.

Avamar systems running Avamar server version 7.1.x only accept replication data as part of a replication policy. A task that uses the Avamar REST API to replicate the data for a single client to an Avamar server version 7.1.x fails.

Tasks that use the Avamar REST API to replicate the data for single clients to Avamar systems running Avamar server version 7.2 and newer are not subject to this restriction. Those systems accept replication data for individual clients and for policy-based replications.

Backup of nonactivated client remains in RUNNING state

Starting a backup task for a client that is not activated results in the task entering, and remaining in, the RUNNING state.

A backup task initiated through the Avamar REST API stops responding while in the RUNNING state when each of the following is true:

- The Avamar REST API server adds the client to a folder on an Avamar system running Avamar server version 7.0 or earlier
- The backup client is registered, but not activated with the Avamar system

Under these circumstances, the task remains in the RUNNING state.

When the Avamar system is running Avamar server version 7.1 or later, the backup task instead returns an error.

To avoid these problems, always check the activation status of a client before initiating a backup.

Checking client activation status

Before starting a client backup, check the activation status of the client. By doing this you avoid the error situation that occurs when a backup is started for a nonactivated client.

Procedure

1. Request the `Client` object for the client:

   ```
   GET /client/CLIENT_URI
   ```

   The Avamar REST API server responds with the identified `Client` object.
2. Examine the `Client` object to determine the value of the `activated` element.
3. Based on the value of the `activated` element, choose whether to run or to not run the client backup.

Value	Action
true	Run the backup
false	Do not run the backup

Example 16 Client object showing activated element for nonactivated client

In XML format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><Client
xmlns="http://www.emc.com/concerto/v1.0"
name="clientsystem.widgets.com" id="7bf2c9c1-3eec-4f3c-
b366-84a9cc76a495"
statusCode="INSYNC" statusMessage="In sync with DPR's" href="https://
lava7120:8543/rest-api/client/7bf2c9c1-3eec-4f3c-b366-84a9cc76a495"
type="application/xml,application/json">
<Description></Description><DataProtectionResource href="https://
lava7120:8543/rest-api/admin/dataProtectionResource/d85dd297-
c931-43e1-b89d-9e712b16b769"
id="d85dd297-c931-43e1-b89d-9e712b16b769" name="DPR01"/><Folder
href="https://lava7120:8543/rest-api/folder/9ac64baf-2009-4f98-91d0-
c9167a1abbd1"
id="9ac64baf-2009-4f98-91d0-c9167a1abbd1" name="Widgets"/><Contact></
Contact><Phone></Phone><Email></Email><Location></
Location><Activated>false</
Activated><ActivationTS>1969-12-31T17:00:00.000-07:00</ActivationTS>
<InitializationTS>2014-04-25T11:50:43.817-06:00</
InitializationTS><LastBackupTS>1969-12-31T17:00:00.000-07:00</
LastBackupTS><LastCheckinTS>1969-12-31T17:00:00.000-07:00</
LastCheckinTS>
<ClientOS>N/A</ClientOS></Client>
```

In JSON format:

```
{
  "activated" : false,
  "activationTS" : "1969-12-31T17:00:00.000-07:00",
  "clientOS" : "N/A",
  "dataProtectionResource" : {
    "href" : "https://lava7120:8543/rest-api/admin/
dataProtectionResource/d85dd297-c931-43e1-b89d-9e712b16b769",
    "id" : "d85dd297-c931-43e1-b89d-9e712b16b769",
    "name" : "DPR01"
  },
  "folder" : {
    "href" : "https://lava7120:8543/rest-api/folder/
9ac64baf-2009-4f98-91d0-c9167a1abbd1",
    "id" : "9ac64baf-2009-4f98-91d0-c9167a1abbd1",
    "name" : "Widgets"
  },
  "href" : "https://lava7120:8543/rest-api/client/7bf2c9c1-3eec-4f3c-
b366-84a9cc76a495",
  "id" : "7bf2c9c1-3eec-4f3c-b366-84a9cc76a495",
  "initializationTS" : "2014-04-25T11:50:43.817-06:00",
  "lastBackupTS" : "1969-12-31T17:00:00.000-07:00",
  "lastCheckinTS" : "1969-12-31T17:00:00.000-07:00",
  "name" : "clientsystem.widgets.com",
  "statusCode" : "INSYNC",
  "statusMessage" : "In sync with DPR's",
  "type" : "application/xml,application/json"
}
```

INDEX