# **Dell EMC Avamar**

Version 18.1

# **REST API Getting Started Guide**

302-004-675 REV 01



Copyright © 2014-2018 Dell Inc. or its subsidiaries. All rights reserved.

#### Published July 2018

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

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS-IS." DELL MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. USE, COPYING, AND DISTRIBUTION OF ANY DELL SOFTWARE DESCRIBED IN THIS PUBLICATION REQUIRES AN APPLICABLE SOFTWARE LICENSE.

Dell, EMC, and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be the property of their respective owners. Published in the USA.

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

# CONTENTS

Figures		5
Tables		7
Preface		9
Chapter 1	Introduction	13
	Description	14
	New with Avamar release 18.1	14
	Purpose	14
	Deployment	15
	Documentation conventions	15
Chanter 2	Installation	17
	Requirements	18
	Installing the software	
	Testing the installation	
	Checking the installed version	
	Upgrading from Avamar release 7.5 Service Pack 1 or earlier	
	Migrating the Avamar REST API database for Avamar 7.5.0	
	Uninstalling	22
	Manually stopping	22
	Manually starting	23
	Changing the provider credentials	
	Changing the Avamar REST API server port numbers	24
Chapter 3	Architecture	27
	Architecture of the Avamar REST API	
Chapter 4	Concepts	31
	Design goal	
	Core concepts	32
Chapter 5	Fundamentals	35
•	Representational state transfer	
	Session log in and log out	36
	API call types	
	Client allocation strategy	39
	Changing the built-in client allocation strategy	40
Chapter 6	Advanced API Calls	41
	Browse operations	42
	Browsing a client	

	Browsing a backup	42
	Browse response	43
	Dataset creation	45
	Elements in the DatasetItem element	
	Creating a dataset	
	Setting backups to go to a Data Domain storage system	
	VMware	53
	VMware vCenter	54
	VMware virtual machines	55
	Proxy appliance for VMware	62
	On-demand virtual machine backups	70
	Virtual machine browse operations	74
	Virtual machine restore operations	76
Chapter 7	Troubleshooting	81
onupter /	Troubleshooting an Avamar REST API installation test failure	82
	Troubleshooting Insufficient Java Heap Storage Space in REST API 82	Server
	Troubleshooting a failed request	83
Appendix A	Known Problems and Limitations	85
	Replication without policy fails to Avamar server version 71 x	86
	Backup of nonactivated client remains in RUNNING state	86
	Checking client activation status	
Index		89

# **FIGURES**

1	Components of the Avamar REST API architecture	28
2	Geographically selected resource pools	33
3	Tenant hierarchy	34
4	Resource share flexibility	34

FIGURES

# TABLES

1	Typographical conventions	10
2	Variables used in this documentation	16
3	Avamar REST API requirements	
4	Descriptions of endpoint to endpoint communication	28
5	Avamar REST API object locations	28
6	Descriptions of the elements in the DatasetItem element	48
7	Required elements in a BackupRequest for an individual virtual machine	70
8	Elements in a request for an image level restore	77
9	Elements in a request for a file level restore	78

TABLES

# Preface

As part of an effort to improve the product lines, revisions of the software and hardware are periodically released. Therefore, some functions that are described in this document might not be supported by all versions of the software or hardware currently in use. The product release notes provide the most up-to-date information on product features.

Contact the technical support professional when a product does not function correctly or does not function as described in this document.

#### Note

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

#### Purpose

This document provides information to use 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.

Revision	Date	Description
01	July 7, 2018	GA release of Avamar 18.1

### **Related documentation**

The following publications provide additional information:

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

#### Special notice conventions used in this document

These conventions are used for special notices.

#### **DANGER**

Indicates a hazardous situation which, if not avoided, results in death or serious injury.

#### **A**WARNING

Indicates a hazardous situation which, if not avoided, could result in death or serious injury.

### **A**CAUTION

Indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

#### NOTICE

Addresses practices that are not related to personal injury.

#### Note

Presents information that is important, but not hazard-related.

#### **Typographical conventions**

These type style conventions are used in this document.

#### Table 1 Typographical conventions

Bold	Used for names of interface elements, such as names of windows, dialog boxes, buttons, fields, tab names, key names, and menu paths (what the user specifically selects or clicks)
Italic	Used for full titles of publications that are referenced in text
Monospace	Used for:
	System code
	System output, such as an error message or script
	<ul> <li>Pathnames, filenames, prompts, and syntax</li> </ul>
	Commands and options
Monospace italic	Used for variables
Monospace bold	Used for user input
[]	Square brackets enclose optional values
I	Vertical bar indicates alternate selections - the bar means "or"
{}	Braces enclose content that the user must specify, such as $\boldsymbol{x}$ or $\boldsymbol{y}$ or $\boldsymbol{z}$
	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 resolve a product issue before contacting 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 by Name** box.
- 3. Select the product from the list that appears.
- 4. Click the arrow next to the Find a Product by Name box.
- 5. (Optional) Add the product to the **My Products** list by clicking **Add to My Saved 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. To supplement the information in product administration and user guides, review the following documents:

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

To search the 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 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 products.

#### Live chat

To engage 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 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 a 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

It is recommended to 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 Customer Support.

#### **Comments and suggestions**

Comments and suggestions help 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	14
•	Purpose	14
•	Deployment	15
•	Documentation conventions	15

# 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 which is based on the representational state transfer (REST) API architecture model.

#### **Programming interface**

When using the Avamar REST API, write the code that can manage multiple Avamar systems simply and efficiently. The Avamar REST API abstracts Avamar systems and domains into logical entities. By performing this step, the Avamar REST API enhances the ability to write code that manages the Avamar systems and customer's requirements.

While the Avamar REST API handles management tasks through Avamar Administrator and 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.

### New with Avamar release 18.1

With Avamar release 18.1, the Avamar REST API is provided as a stand-alone server that can be deployed on either a physical or virtual SLES 12, RHEL 7.4, or RHEL 7.5 server with JRE 1.8 installed, or an Avamar utility node.

## 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, create the custom web portals to interact with the Avamar systems through a REST programming 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 Avamar systems into logical entities which allow you to perform operations on them in parallel. This step reduces the complexity that is required to manage multiple Avamar systems.

The Avamar REST API is designed to solve the issues on 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 optimize storage capacity.

## Deployment

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

### **RPM file**

Obtain Avamar REST API through a sales representative. Receive an RPM file that contains:

- Avamar REST API server
- Avamar REST API documentation

Install the Avamar REST API on a SLES 12, RHEL 7.4, or RHEL 7.5 server with JRE 1.8 installed, or an Avamar utility node.

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**

The 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, the 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-d989-4058a57b-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

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

The examples where the full text is set out do not use this abbreviation convention. The full URL appears on those examples.

#### 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 2 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 that is assigned to a backup.
CLIENT_URI	Uniform resource identifier that is assigned to a client computer.
FOLDER_URI	Uniform resource identifier that is assigned to a folder.
HVM_URI	Uniform resource identifier that is assigned to a hypervisor manager, such as a VMware vCenter.
PLUG-IN_URI	Uniform resource identifier that is assigned to a plug-in.
PLUG-IN_URL	Uniform resource locator that is used by the Avamar REST API server to reference a plug- in.
POLICY_URI	Uniform resource identifier that is assigned to a policy.
PROVIDER_URI	Uniform resource identifier that is assigned to the provider.
PROXY_NAME	Fully qualified domain name of a proxy appliance for VMware.
PROXY_URL	Uniform resource locator that is 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 that is assigned to a retention policy.
TASK_URI	Uniform resource identifier that is assigned to a task.
TENANT_URI	Uniform resource identifier that is 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	
Installing the software	
• Upgrading from Avamar release 7.5 Service	Pack 1 or earlier 21
Uninstalling	
Manually stopping	
Manually starting	
Changing the provider credentials	23
Changing the Avamar REST API server port	numbers24

## Requirements

The following table lists the requirements for the Avamar REST API.

Table 3 Avamar REST API requirements

Category	Requirement	
Installation host	Either of the following:	
	• A SLES 12 server (physical or virtual) with JRE 1.8 installed	
	<ul> <li>A RHEL 7.4 or 7.5 server (physical or virtual) with JRE 1.8 installed</li> </ul>	
	An Avamar utility node	
If performing the following operations on a virtual machine:	All managed Avamar systems must be running Avamar server version 7.1 or later	
Configuration		
Backup operations		
Restore operations		
If performing replication destination configuration	All managed Avamar systems must be running Avamar server version 7.1 or later	

## Installing the software

### Procedure

- 1. Log into the target computer.
- 2. To switch to root, type the following command:

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. To expand and install the Avamar REST API software, type the following command:

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 and properties file
- Requests a new, non-default, provider password
- 6. Create a provider password.

To change the username and password, complete the task that is described in Changing the provider credentials on page 23.

7. The Avamar REST API server runs on the default (8580/8543) ports.

To change the default ports, complete the task that is described in Changing the Avamar REST API server port numbers on page 24.

8. Type the following:

On SLES 12, RHEL 7.4, or RHEL 7.5: systemctl start concerto.service

On Avamar utility node: service concerto start

Jetty starts and the Avamar REST API server starts.

#### Note

When the Avamar REST API server starts for the first time, it may take up to 10 minutes to initialize and configure the system and its database. Subsequent restarts of the Avamar REST API server take less time.

#### Results

After Jetty starts and initializes the Avamar REST API server, the installation is complete.

#### After you finish

Use the  $\ensuremath{\texttt{curl}}$  tool to test the installation.

### Testing the installation

Use a  ${\tt curl}$  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- -X GET https://TARGETCOMPUTER:8543/rest-api/versions
```

This step performs a simple test to verify that the Avamar REST API is running.

3. Next, test logging into the system by typing 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 the login to 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.example.com.

```
admin@lava7120:~/>: curl -k -D- --user admin:changeme -X POST https://
test.example.com:8543/rest-api/login
HTTP/1.1 201 Created
X-Concerto-Authorization:
OTQwMGE1MDYtNzIyOC000TYxLWI40GMtYjliNWEwNzVmNGE2
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.example.com/concerto/v1.0" href="https://
test.example.com:8543/rest-api/session" type="application/
xml,application/json"><User name="admin"/><AccessPoint href="https://</pre>
test.example.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 into the target computer.
- 2. To switch to root, type the following command:

su -

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 from Avamar release 7.5 Service Pack 1 or earlier

To upgrade the Avamar REST API software from release Avamar 7.5 Service Pack 1 or earlier, install the Avamar REST API software on a host that meets the requirements of the Avamar REST API.

#### Note

You can upgrade from 7.5.1 or earlier versions, however migration is only needed if upgrading from 7.5.0 or earlier. Avamar REST API server 7.5.1 and 18.1 share the same database, therefore the database migration process is not required.

For Avamar 7.5.0, after migrating existing Avamar REST API data to the new version of the Avamar REST API, all existing behaviors, such as existing APIs and data types, remain completely compatible with previous versions. No change or rewrite is required for existing applications.

## Migrating the Avamar REST API database for Avamar 7.5.0

During upgrade of the Avamar REST API software from Avamar release 7.5.0 or earlier, data must be migrated from the Avamar REST API database.

#### Before you begin

Install the Avamar REST API software on a host that meets the requirements that are listed in Requirements on page 18.

In this procedure, the Avamar server that is runningAvamar REST API Avamar release 7.5.0 or earlier is referred to as the migration source. The host that is running the new version of the Avamar REST API software is referred to as the migration target.

#### Procedure

- 1. Copy the export script, /opt/concerto/bin/export-restapi-db.sh, from the migration target to a temporary location on the migration source.
- 2. Log into the migration source.
- 3. To switch to root, type the following command:

su -

- 4. Change the working directory to the temporary location of the export script.
- 5. To run the export script, type the following command:

./export-restapi-db.sh

The script creates an export file on the migration source. The export file is named *VERSION-HOSTNAME-DBDUMP.gz*, where *VERSION* is the version of the Avamar software and *HOSTNAME* is the hostname of the migration source. When the script has completed, it lists file's name and location.

- Copy the export file from on the original host to a temporary location on the migration target.
- 7. Log into the migration target.
- 8. To switch to root, type the following command:

su -

9. Type the following command to run the import script:

/opt/concerto/bin/import-restapi-db.sh /TEMP\_LOCATION/VERSIONhostname-DBDUMP.gz

#### Results

After the import script has completed, the Avamar REST API server restarts automatically.

## Uninstalling

Use the RPM erase 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.

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

#### Procedure

- 1. Log into the target computer.
- 2. To switch to root, type the following command:

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. Type the following:

rm -rf /opt/concerto/

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.

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.

#### Procedure

- 1. Log into the target computer.
- 2. To switch to root, type the following command:

su -

3. Type the following command to stop the REST API server:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemct1 stop concerto.service

On Avamar utility node: service concerto stop

#### Results

The Avamar REST API server shuts down.

## Manually starting

Start the Avamar REST API server from the command line.

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 into the target computer.
- 2. To switch to root, type the following command:

su -

3. Type the following command to start the REST API server:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl start concerto.service

On Avamar utility node: service concerto start

#### Results

The Avamar REST API server starts.

## Changing the provider credentials

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

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

The custom system properties file is: /opt/concerto/config/ restserver.properties

#### Procedure

- 1. Log into the target computer.
- 2. To switch to root, type the following command:

su -

3. Type the following command to stop the REST API server:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl stop concerto.service

On Avamar utility node: service concerto stop

- 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 to start the REST API server:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl start concerto.service

On Avamar utility node: service concerto start

#### Results

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

## Changing the Avamar REST API server port numbers

The Avamar REST API server exposes two service endpoints (default ports 8543, the main REST API service port, and 8580, the REST API online documentation port). These default port numbers can be changed.

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

The custom system properties file is: /opt/concerto/config/ restserver.properties

### Procedure

- 1. Log into the target computer.
- 2. To switch to root, type the following command:

su -

3. Type the following command to stop the REST API server:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl stop concerto.service

On Avamar utility node: service concerto stop

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

```
server.https.port=8543
server.http.port=8580
```

- 5. Change the values for port numbers.
- 6. Save and close the file.
- 7. Type the following command to start the REST API server:

On SLES 12 server: systemctl start concerto.service

On Avamar utility node: service concerto start

NOTICE

Your firewall needs to be reconfigured to allow incoming traffic through the new port numbers and, if necessary, block the old port numbers.

### Results

The port numbers are changed and clients are expected to communicate using the new port numbers.

25

Installation

# **CHAPTER 3**

# Architecture

This chapter includes the following topics:

•	Architecture of the Avamar	REST API	28

## Architecture of the Avamar REST API

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 4 Descriptions of endpoint to endpoint communication

Endpoint	Endpoint	Description
Avamar REST API client	Jetty	All communications use Hypertext Transfer Protocol Secure (HTTPS). The Jetty web server listens on port 8543. The Jetty web server also serves the online API specifications using HTTP on port 8580.
Avamar REST API server	HSQLDB database	HSQLDBis an embedded database running within the Avamar REST API server.
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 5 Avamar REST API object locations

Avamar REST API object	Location
API specification	/opt/concerto/app/restapi-doc.jar
Server log file	/opt/concerto/logs/restserver.log
Jetty log file	/opt/concerto/logs/jetty.log

Table 5 Avamar REST API object locations (continued)

Avamar REST API object	Location
Custom system properties file	/opt/concerto/config/restserver.properties
Scripts and utilities	/opt/concerto/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.

#### The custom system properties file

All system default properties are kept within the software, while the custom system properties file, /opt/concerto/config/restserver.properites, allows you to customize and override system default property values. Please refer to online documentation at http://RESTAPISERVER:8580/rest-api-doc/ for details.

To change a particular system property value, type a name value pair in /opt/ concerto/config/restserver.properties. Some properties are dynamic, so their new values take effect without having to restart the Avamar REST API server. Others values are not dynamic, which require that the server is restarted for the new properties values to take effect. Architecture

# **CHAPTER 4**

# Concepts

This chapter includes the following topics:

٠	Design goal	62
٠	Core concepts	62

## **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 to work directly with each Avamar system. Instead, abstract the systems into groups that can meet the business requirements and write code that addresses these logical abstractions of the systems.

The following examples determine 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 ondemand backup of one of those clients, it is not required to know the Avamar system on which the client is backed up. 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 which is responsible for the client and directs the backup operation to that system.

#### **Example: All activities**

If the same tenant wants 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 modify 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 in working 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 configuring the Avamar REST API server, provide information about the

Avamar systems that the Avamar REST API server manages as data protection resources.

When defining a data protection resource, specify user credentials for an account with administrative access to that Avamar system. As a recommended best practice, create a dedicated administrative account on each Avamar system, such as rest-api. Use that account when configuring the Avamar system as a data protection resource. Identifying the account makes it simple 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 it is assigned to a resource pool. When the Avamar REST API server is configured, specify the data protection resources that should be in each of the resource pools.

One method of assigning data protection resources to a resource pool is to use 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, assign several tenants to share a resource pool. A single tenant can also dedicate to a resource pool. Tenants can 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 creating a resource share, 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 creating 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.

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	.36
•	Session log in and log out	.36
•	API call types	38
•	Client allocation strategy	39

## 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 that is 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 more information about the backup jobs that ran on the client where the client object represents 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 that are 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 that is sent during the session. The following is an example of this header:

X-Concerto-Authorization: NzE2NjkzMDYtYjA2Yi00Y2IxLWI4MTMtMDIzNzQxMjM00WZk

#### 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.example.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"
} ],
```

37

```
"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 which is 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" : "2014-05-09T14:30:05.8022",
"startTime" : "2014-05-09T14:30:05.8022",
"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 that is 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 which are based on the client allocation strategy you select. To suit your business requirements, select a built-in strategy.

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 the 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 the 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: /opt/concerto/config/ restserver.properties

#### Procedure

- 1. Log into the target computer.
- 2. To switch to root, type the following command:

su -

3. Type the following command to stop the REST API server:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl stop concerto.service

On Avamar utility node: service concerto stop

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 to start the REST API server:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl start concerto.service

On Avamar utility node: service concerto 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	42
•	Dataset creation	.45
•	VMware	53

# **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 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 set up 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 that is used with the client.
- Browse the client by running the following API call and including a ClientBrowseRequest object:

```
POST /client/CLIENT_URI/action/browse
<ClientBrowseRequest xmlns="http://www.example.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 that any of the managed Avamar systems take on.

#### 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.example.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, the client sends a browse request or 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 that the element represents.

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 that this kv array element represents.

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

43

```
"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 the business requirements.

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 that FOLDER\_UR/ identifies.

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.example.com/concerto/v1.0" name="TestDS">
        <Description></Description>
        <Description>
        <Descrip
```

In JSON format:

#### 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 that the dataset handles. 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 specifying All as the DatasetTarget value, the DatasetItem element targets all of the data in the file system.

#### In XML format:

```
<Dataset xmlns="http://www.example.com/concerto/v1.0" name="My New</pre>
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></</pre>
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></</pre>
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:\\"
         }
      },
      {
         "name": "Linux File System",
         "Plugin": {
             "href": "https://localhost:8543/rest-api/plugin/87f7c7df-
```

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

#### **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:

#### In JSON format:

```
"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"
```

#### Example 4 Dataset request body with DatasetOption elements (continued)

## **Elements in the DatasetItem element**

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

Table 6 Descriptions of the elements in the DatasetItem element

Element	Required	Description
Plugin	Yes	URL reference to the associated plug-in.
DatasetTarget	No	Data that the DatasetItem element targets on. 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 the values identify 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 the 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.example.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:

#### 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:

#### 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:

#### In JSON format:

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

5. Add a Plugin element.

The  ${\tt Plugin}$  element is a reference to a plug-in using a URL that includes the plug-in's URI.

## In XML format:

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:

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 that are supported by the plug-in.

Use the PluginOptionName value when referencing an option.

#### In XML format:

```
<Dataset xmlns="http://www.example.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/</</pre>
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"
```

51



- 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 browse on a Data Domain storage system by setting DatasetOption elements in a dataset at the DatasetItem level.

#### Before you begin

Perform the following:

- Determine the *PLUG-IN\_URI* for each plug-in that stores 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 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 plugins 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 <code>TestDS</code> has the <code>DatasetItem</code> element named <code>MYDSITEMNAME</code>. This <code>DatasetItem</code> 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:

#### 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"
          }
      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 that are 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 bestname of the vCor
  - 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 that is used to add a vCenter with hostname vccc.asl.lab.example.com, administrator's account administrator, and password changeme. Two optional elements are also shown: Port and Description. The Port element provides the port that the vCenter listens on. Here port 443 (the

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

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

In XML format:

In JSON format:

```
"name": "vccc.asl.lab.example.com",
"Description": null,
"Hostname": "vccc.asl.lab.example.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 that are 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 option by using additional elements to provide the extra information that the Avamar REST API server requires.

#### API call

Add a virtual machine by running the following API call:

POST /folder/FOLDER\_URI/client

FOLDER\_UR/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. This element can also 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.example.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>
<//ClientExt>
</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 with the Client element's name attribute to identify the virtual machine. The value that is provided in the name attribute must exactly match the vCenter's name for the virtual machine.

The VmClientExt element can also 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

To add the virtual machine to the Avamar REST API server, use the vCenter UUID of a virtual machine.

#### 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.example.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.example.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.example.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.example.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 correctly formatted vCenter UUID.

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

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

```
{
    "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\_UR/ 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.

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

' "name": ""

{

2. Add a value to the name attribute.

#### In XML format:

```
<Client xmlns="http://www.example.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.example.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.example.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.

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

```
<VmClientExt>

<DataCenter>/DataCenter</DataCenter>

<VmFolder>folder</VmFolder>

</VmClientExt>

</Client>
```

```
"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:

#### 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\_UR/ is the URI of the Virtual Machines folder.

61

#### 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 option by using additional elements to provide the extra information that 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.

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

#### Example 8 A Client element for a proxy appliance (continued)

```
<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 that are 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 that is 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 that is 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  ${\tt vmDatastore}$  list with the URL for each managed datastore.

2. Draft a skeleton VmDatastoreList element.

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

{ }

3. Add a VmDatastore element containing a Url element.

#### In XML format:

#### 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.example.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, which is the value that is returned in the url element of the vCenter's vmDatastore list.

5. Add another VmDatastore element and Url element for every datastore being added.

#### In XML format:

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

<VmDatastore>

<Url>DS_URL</Url>

</VmDatastore>

<Url>DS_URL</Url>

</VmDatastore>

</VmDatastore>

</VmDatastore>

</VmDatastore>
```

#### 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.
- HVM\_UR/ 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.example.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  ${\tt vmDatastore}$  list with the URL for each managed datastore.

2. Draft a skeleton VmDatastoreList element.

In XML format:

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

In JSON format:

{ }

3. Add a VmDatastore element containing a Url element.

#### In XML format:

In JSON format:

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

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

#### In XML format:

#### In JSON format:

{

```
"VmDatastore": {
"Url": "DS_URL"
```

}

where  $DS\_URL$  is the vCenter's URL for the datastore, which is the value that is returned in the url element of the vCenter's vmDatastore list.

5. Add another VmDatastore element and Url element for every datastore being removed.

#### In XML format:

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

<VmDatastore>

<Url>DS_URL</Url>

</VmDatastore>

<Url>DS_URL</Url>

</VmDatastore>

</VmDatastore>

</VmDatastore>

</VmDatastore>
```

#### In JSON format:

{

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

Note that the JSON format uses an array of  ${\tt Url}$  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.example.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.example.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

```
<ReferenceList xmlns="http://www.example.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/
```

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

```
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.example.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.example.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 7 Required elements in a BackupRequest for an individual virtual machine

Element	Description
Plugin	<ul> <li>Reference to the plug-in to use in the backup.</li> <li>This element is one of the following for a VMware virtual machine:</li> <li>Windows VMware Image</li> <li>Linux VMware Image</li> </ul>
Retention	Reference to the retention object that defines 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 <code>Datasource</code> element has the value <code>All</code>, indicating a full VMware image backup.

```
<BackupRequest xmlns="http://www.example.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/
```

Example 11 BackupRequest element for backing up an individual virtual machine (continued)

```
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 which is 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  ${\tt task}$  element's status changes to  ${\tt RUNNING},$  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

To run an on-demand backup of a virtual machine, use an API call with a BackupRequest element in the request body.

#### 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.example.com/concerto/v1.0" > <DataSource><Source></DataSource>

```
<Plugin href="" />
<Retention href="" />
</BackupRequest>
```

```
{
    "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.example.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.example.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.
```
<BackupRequest xmlns="http://www.example.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

To run a backup of the virtual machines, use an API call with the URI of a virtual machine backup policy.

#### 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 run 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 perform the following steps:

- To view information about the virtual disk images in a backup.
- 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 that are 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 that are 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.example.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 that are 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"
```

Example 12 Excerpt from image level browse response (continued)

}], "

### Browse virtual machine files

View information about the files and directories that are 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.example.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.example.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 that is 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 on Disk#1.

In XML format:

```
<BackupBrowseRequest xmlns="http://www.example.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 that is 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 that are shown here.

In XML format:

```
<RestoreRequest xmlns="http://www.example.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 <code>DestClient</code> element and additional elements that are provided in the <code>RestoreRequest</code> 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 8 Elements in a request for an image level restore

Element	Description
Plugin	<ul> <li>Reference to the plug-in to use for the restore. For a virtual machine restore operation, this plug-in option is one of the following:</li> <li>Windows VMware Image</li> <li>Linux VMware Image</li> </ul>
DestClient	Element containing elements that provide information about the restore target.
Vmldentification	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 that is associated with the restore target.
EsxHost	Name of the ESX host that is associated with the restore target.
Datastore	Name of the datastore that is 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.example.com/concerto/v1.0" >
<Plugin href="https://localhost:8543/rest-api/plugin/960cf388-
c775-4521-918f-5d9f11f49171"/>
<DestClient>
```

#### Example 14 RestoreRequest element for an image level restore (continued)

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.example.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 9 Elements in a request for a file level restore

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

Table 9 Elements in a request for a file level restore (continued)

Element	Description
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:

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"
```

Advanced API Calls

# **CHAPTER 7**

# Troubleshooting

This chapter includes the following topics:

•	Troubleshooting an A	vamar REST AP	l installation test	failure82
---	----------------------	---------------	---------------------	-----------

- Troubleshooting Insufficient Java Heap Storage Space in REST API Server......82

# Troubleshooting an Avamar REST API installation test failure

Find and fix a problem that is 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, follow the recommendation on using a curl command. The curl tool does not display a 201 Created HTTP response header and a session object.

#### Procedure

- 1. Log into the target computer.
- 2. To switch to root, type the following command:

su -

3. Determine if the problem is with Jetty by checking the log.

View the current log located in /opt/concerto/logs/jetty.log.

- Determine if the problem is with the Avamar REST API by checking the log.
   View the current log located in /opt/concerto/logs/restserver.log.
- 5. Restart the Avamar REST API server by doing one of the following:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server, type the following command: systemctl restart concerto.service

On an Avamar utility node, type the command: service concerto stop. Once the server is stopped, type the command: service concerto start.

# Troubleshooting Insufficient Java Heap Storage Space in REST API Server

In large environments, you may encounter instances where heap space of Java application becomes insufficient when the Avamar REST API is launched in a standalone system or in an Avamar utility node. You may want to increase the heap space to perform the procedures seamlessly.

#### SYMPTOM:

While launching REST API, you are getting repeated java.lang.OutOfMemoryError: Java heap space error messages.

#### Procedure

1. Stop the Avamar REST API server by typing the following commands:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl stop concerto.service

On Avamar utility node: service concerto stop

2. Open the following file in a text editor:

/opt/concerto/bin/concertoserver.sh

3. Find the following line:

\$JAVA OPTS = " -Xms1028m -Xmx2056m "

Note

In this line, the Java heap size is 2GB.

- 4. Replace the "1028" and "2056" values in the above line with the new heap size and save the text editor.
- 5. Restart the Avamar REST API server by typing the following commands:

On a SLES 12, RHEL 7.4, or RHEL 7.5 server: systemctl start concerto.service

On Avamar utility node: service concerto start

# Troubleshooting a failed request

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

#### Before you begin

Perform 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 the new code by sending an Avamar REST API request to the Avamar REST API server. The request fails.

#### Procedure

 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.

 Examine the Avamar REST API log at: /opt/concerto/logs/ restserver.log.

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

83

Troubleshooting

# **APPENDIX A**

# **Known Problems and Limitations**

This appendix includes the following topics:

- Replication without policy fails to Avamar server version 7.1.x......86

# 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.

The Avamar systems that are 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.

The tasks that use the Avamar REST API to replicate the data for single clients to Avamar systems that are 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 an inactive client results in the task entering, and remaining in the RUNNING state.

A backup task that is initiated through the Avamar REST API stops responding 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 that is 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 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 performing this step, avoid the error that occurs when a backup is started for an inactive 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. To determine the value of the activated element, examine the Client object.
- 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

Value	Action
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.example.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://</pre>
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"
```

87

Known Problems and Limitations

# INDEX

## Α

activation status 86 allocation 39, 40 built-in strategy 40 API calls 38 architecture 28 asynchronous 38 Avamar 18 Avamar server 18 Avamar system proxy appliance 62

# В

backup 42, 43, 48, 52, 70, 71, 73, 76, 86 browse response 43 check activation 86 Data Domain storage system 52 dataset 48 on-demand 70, 71, 73 policy 70, 73 unresponsive 86 virtual machine 71, 73 backup policy proxy appliance 67, 69 BackupBrowseRequest 74, 75 BackupRequest 70, 71 **BALANCED** allocation 40 browse 42, 43, 74, 75 backup 42 client 42 response 43 virtual machine files 74, 75 virtual machine images 74 browse backup 42 browse client 42 browse operations 42

# С

ChangedBlockTracking 55, 57, 59 changing server port numbers 24 changing username 23 client 39, 40, 42, 43, 86 activation status 86 allocation 39, 40 browse response 43 ClientExtensionType 57, 59 code troubleshooting 83 core concepts 32

# D

Data Domain storage system 52 data protection resource 32

datacenter 59 DataCenter 55 dataset 45 Dataset 48 DatasetExclude 48 DatasetInclude 48 DatasetItem 45, 48, 52 DatasetExclude 48 DatasetInclude 48 DatasetOption 48 DatasetTarget 48 elements 48 Plugin 48 DatasetOption 45, 48, 52 DatasetTarget 45, 48 deployment method 15 description 14 design 32 DestClient 76, 77 documentation 15

# F

file paths 28 folder 54, 59 FREE\_SPACE allocation 40

## Н

hypervisorManager 54

## I

image backup 71 install 18, 19 testing 19

# L

login 36, 82 troubleshooting 82

# Μ

Mode 45

# 0

objects 28

# Ρ

path 15 Plugin 48 policy 67, 69, 70, 73 backup 73 proxy appliance 67, 69 ports 28 product description 14 proxy appliance 62, 63, 66, 67, 69 purpose 14

### R

ReferenceList 67, 69 resource pool 32 resource share 32 REST 36 restore 76, 77 RestoreRequest 76, 77

# S

server port numbers, changing 24 session 36 starting 23 stopping 22 synchronous 38

# Т

tenant 32 testing 19 troubleshoot code issues 83 login 82

# U

username, changing 23 UUID 57

# V

variable 15 vCenter 54, 55 virtual machine 55 version 18, 20 virtual machine 55, 57, 59, 71, 73, 76, 77 adding 57, 59 backup 71, 73 datacenter and folder 59 file level restore 76, 77 image level restore 76, 77 policy 73 restore request 76, 77 **UUID 57** VmClient 55 VmClientExt 57, 59 VmDatastoreList 63, 66 VmFolder 55 VMware proxy appliance 62, 63, 66, 67, 69 resources 53 tasks 53 VMware vCenter 54, 55 virtual machine 55