



# Red Hat Data Grid 8.2

## Data Grid REST API

Configure and interact with the Data Grid REST API



# Red Hat Data Grid 8.2 Data Grid REST API

---

Configure and interact with the Data Grid REST API

## Legal Notice

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

Access data, monitor and maintain clusters, perform administrative operations through the Data Grid REST API.

# Table of Contents

<b>RED HAT DATA GRID</b> .....	<b>5</b>
<b>DATA GRID DOCUMENTATION</b> .....	<b>6</b>
<b>DATA GRID DOWNLOADS</b> .....	<b>7</b>
<b>MAKING OPEN SOURCE MORE INCLUSIVE</b> .....	<b>8</b>
<b>CHAPTER 1. DATA GRID REST ENDPOINT</b> .....	<b>9</b>
1.1. REST AUTHENTICATION	9
1.2. SUPPORTED PROTOCOLS	9
1.3. DATA FORMATS AND THE REST API	9
1.3.1. Supported Formats	9
1.3.2. Accept Headers	10
1.3.3. Names with Special Characters	10
1.3.4. Key-Content-Type Headers	10
1.3.5. JSON/Protostream Conversion	11
1.4. CROSS-ORIGIN RESOURCE SHARING (CORS) REQUESTS	12
1.4.1. Allowing all CORS permissions for some origins	12
<b>CHAPTER 2. INTERACTING WITH THE DATA GRID REST API</b> .....	<b>14</b>
2.1. CREATING AND MANAGING CACHES	14
2.1.1. Creating Caches	14
2.1.1.1. Cache Configuration	14
JSON format	15
2.1.2. Verifying Caches	15
2.1.3. Creating Caches with Templates	15
2.1.4. Retrieving Cache Configuration	15
2.1.5. Converting Cache Configurations to JSON	15
2.1.6. Retrieving All Cache Details	16
2.1.7. Adding Entries	17
2.1.8. Replacing Entries	18
2.1.9. Retrieving Data By Keys	19
2.1.10. Checking if Entries Exist	20
2.1.11. Deleting Entries	20
2.1.12. Deleting Caches	21
2.1.13. Retrieving All Keys from Caches	21
2.1.14. Retrieving All Entries from Caches	21
2.1.15. Clearing Caches	23
2.1.16. Getting Cache Size	23
2.1.17. Getting Cache Statistics	23
2.1.18. Querying Caches	23
2.1.19. Re-indexing Data	24
2.1.20. Purging Indexes	25
2.1.21. Retrieving Query and Index Statistics	25
2.1.22. Clearing Search Statistics	27
2.1.23. Retrieving Index Statistics (Deprecated)	27
2.1.24. Retrieving Query Statistics (Deprecated)	27
2.1.25. Clearing Query Statistics (Deprecated)	28
2.1.26. Listing Caches	28
2.1.27. Cross-Site Operations with Caches	28
2.1.27.1. Getting Status of All Backup Locations	28

---

2.1.27.2. Getting Status of Specific Backup Locations	29
2.1.27.3. Taking Backup Locations Offline	29
2.1.27.4. Bringing Backup Locations Online	30
2.1.27.5. Pushing State to Backup Locations	30
2.1.27.6. Canceling State Transfer	30
2.1.27.7. Getting State Transfer Status	30
2.1.27.8. Clearing State Transfer Status	30
2.1.27.9. Modifying Take Offline Conditions	31
2.1.27.10. Canceling State Transfer from Receiving Sites	31
2.1.28. Rolling Upgrades	31
2.1.28.1. Synchronizing Data	31
2.1.28.2. Disconnecting Source Clusters	31
2.2. CREATING AND MANAGING COUNTERS	32
2.2.1. Creating Counters	32
2.2.2. Deleting Counters	32
2.2.3. Retrieving Counter Configuration	32
2.2.4. Getting Counter Values	32
2.2.5. Resetting Counters	33
2.2.6. Incrementing Counters	33
2.2.7. Adding Deltas to Counters	33
2.2.8. Decrementing Counter Values	33
2.2.9. Performing compareAndSet Operations on Strong Counters	34
2.2.10. Performing compareAndSwap Operations on Strong Counters	34
2.2.11. Listing Counters	34
2.3. WORKING WITH PROTOBUF SCHEMAS	34
2.3.1. Creating Protobuf Schemas	34
2.3.2. Reading Protobuf Schemas	35
2.3.3. Updating Protobuf Schemas	35
2.3.4. Deleting Protobuf Schemas	35
2.3.5. Listing Protobuf Schemas	35
2.4. WORKING WITH CACHE MANAGERS	36
2.4.1. Getting Basic Cache Manager Information	36
2.4.2. Getting Cluster Health	38
2.4.3. Getting Cache Manager Health Status	39
2.4.4. Checking REST Endpoint Availability	39
2.4.5. Obtaining Global Configuration for Cache Managers	39
2.4.6. Obtaining Configuration for All Caches	40
2.4.7. Listing Available Cache Templates	40
2.4.8. (Experimental) Obtaining Cache Status and Information	41
2.4.9. Getting Cache Manager Statistics	41
2.4.10. Backing Up Data Grid Cache Managers	43
2.4.11. Listing Backups	44
2.4.12. Checking Backup Availability	44
2.4.13. Downloading Backup Archives	45
2.4.14. Deleting Backup Archives	45
2.4.15. Restoring Data Grid Resources from Backup Archives	45
2.4.15.1. Restoring from Backup Archives on Data Grid Server	45
2.4.15.2. Restoring from Local Backup Archives	46
2.4.16. Listing Restores	47
2.4.17. Checking Restore Progress	47
2.4.18. Deleting Restore Metadata	47
2.4.19. Cross-Site Operations with Cache Managers	48
2.4.19.1. Getting Status of Backup Locations	48

---

2.4.19.2. Taking Backup Locations Offline	48
2.4.19.3. Bringing Backup Locations Online	49
2.4.19.4. Retrieving the State Transfer Mode	49
2.4.19.5. Setting the State Transfer Mode	49
2.4.19.6. Starting State Transfer	49
2.4.19.7. Canceling State Transfer	49
2.5. WORKING WITH DATA GRID SERVERS	49
2.5.1. Retrieving Basic Server Information	49
2.5.2. Getting Cache Managers	50
2.5.3. Adding Caches to Ignore Lists	50
2.5.4. Removing Caches from Ignore Lists	50
2.5.5. Confirming Ignored Caches	50
2.5.6. Obtaining Server Configuration	50
2.5.7. Getting Environment Variables	51
2.5.8. Getting JVM Memory Details	51
2.5.9. Getting JVM Thread Dumps	52
2.5.10. Getting Diagnostic Reports for Data Grid Servers	52
2.5.11. Stopping Data Grid Servers	52
2.6. WORKING WITH DATA GRID CLUSTERS	52
2.6.1. Stopping Data Grid Clusters	52
2.6.2. Stopping Specific Data Grid Servers in Clusters	52
2.6.3. Backing Up Data Grid Clusters	53
2.6.4. Listing Backups	53
2.6.5. Checking Backup Availability	53
2.6.6. Downloading Backup Archives	53
2.6.7. Deleting Backup Archives	53
2.6.8. Restoring Data Grid Cluster Resources	54
2.6.8.1. Restoring from Backup Archives on Data Grid Server	54
2.6.8.2. Restoring from Local Backup Archives	55
2.6.9. Listing Restores	55
2.6.10. Checking Restore Progress	56
2.6.11. Deleting Restore Metadata	56
2.7. DATA GRID SERVER LOGGING CONFIGURATION	56
2.7.1. Listing the logging appenders	56
2.7.2. Listing the loggers	57
2.7.3. Creating/modifying a logger	57
2.7.4. Removing a logger	57
2.8. USING SERVER TASKS	58
2.8.1. Retrieving Server Tasks Information	58
2.8.2. Executing Tasks	59
2.8.3. Uploading Script Tasks	59
2.9. WORKING WITH DATA GRID SECURITY	59
2.9.1. Retrieving the ACL of a user	59
2.9.2. Flushing the ACL cache	60
2.9.3. Retrieving the available roles	60
2.9.4. Retrieving the roles for a principal	60
2.9.5. Granting roles to a principal	60
2.9.6. Denying roles to a principal	60





# RED HAT DATA GRID

Data Grid is a high-performance, distributed in-memory data store.

## **Schemaless data structure**

Flexibility to store different objects as key-value pairs.

## **Grid-based data storage**

Designed to distribute and replicate data across clusters.

## **Elastic scaling**

Dynamically adjust the number of nodes to meet demand without service disruption.

## **Data interoperability**

Store, retrieve, and query data in the grid from different endpoints.

## DATA GRID DOCUMENTATION

Documentation for Data Grid is available on the Red Hat customer portal.

- [Data Grid 8.2 Documentation](#)
- [Data Grid 8.2 Component Details](#)
- [Supported Configurations for Data Grid 8.2](#)
- [Data Grid 8 Feature Support](#)
- [Data Grid Deprecated Features and Functionality](#)

## DATA GRID DOWNLOADS

Access the [Data Grid Software Downloads](#) on the Red Hat customer portal.



### NOTE

You must have a Red Hat account to access and download Data Grid software.

## MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

# CHAPTER 1. DATA GRID REST ENDPOINT

Data Grid servers provide [RESTful](#) HTTP access to data through a REST endpoint built on [Netty](#).

## 1.1. REST AUTHENTICATION

Configure authentication to the REST endpoint with the Data Grid command line interface (CLI) and the `user` command. The CLI lets you create and manage users, passwords, and authorization roles for accessing the REST endpoint.

### Reference

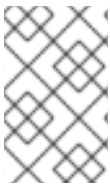
- [Adding Users to Property Realms](#)
- [Configuring Endpoint Authentication Mechanisms](#)

## 1.2. SUPPORTED PROTOCOLS

The Data Grid REST endpoint supports **HTTP/1.1** and **HTTP/2** protocols.

You can do either of the following to use **HTTP/2**:

- Perform an [HTTP/1.1 upgrade](#).
- Negotiate the communication protocol using a [TLS/ALPN extension](#).



### NOTE

TLS/ALPN with JDK8 requires additional client configuration. Refer to the appropriate documentation for your REST client. In most cases you need to use either the Jetty ALPN Agent or OpenSSL bindings.

## 1.3. DATA FORMATS AND THE REST API

Data Grid caches store data in formats that you can define with a [MediaType](#).

See the [Cache Encoding and Marshalling](#) for more information about MediaTypes and encoding data with Data Grid.

The following example configures the storage format for entries:

```
<distributed-cache name="mycache">
  <encoding>
    <key media-type="application/x-java-object"/>
    <value media-type="application/xml; charset=UTF-8"/>
  </encoding>
</distributed-cache>
```

If you do not configure a MediaType, Data Grid defaults to **application/octet-stream** for both keys and values. However, if the cache is indexed, Data Grid defaults to **application/x-protostream**.

### 1.3.1. Supported Formats

You can write and read data in different formats and Data Grid can convert between those formats when required.

The following "standard" formats are interchangeable:

- **application/x-java-object**
- **application/octet-stream**
- **application/x-www-form-urlencoded**
- **text/plain**

You can also convert the preceding data formats into the following formats:

- **application/xml**
- **application/json**
- **application/x-jboss-marshalling**
- **application/x-protostream**
- **application/x-java-serialized**

Data Grid also lets you convert between **application/x-protostream** and **application/json**.

All calls to the REST API can provide headers describing the content written or the required format of the content when reading. Data Grid supports the standard HTTP/1.1 headers "Content-Type" and "Accept" that are applied for values, plus the "Key-Content-Type" with similar effect for keys.

### 1.3.2. Accept Headers

The Data Grid REST endpoint is compliant with the [RFC-2616](#) Accept header and negotiates the correct MediaType based on the conversions supported.

For example, send the following header when reading data:

```
Accept: text/plain;q=0.7, application/json;q=0.8, */*;q=0.6
```

The preceding header causes Data Grid to first return content in JSON format (higher priority 0.8). If it is not possible to convert the storage format to JSON, Data Grid attempts the next format of **text/plain** (second highest priority 0.7). Finally, Data Grid falls back to **\*/\***, which picks a suitable format based on the cache configuration.

### 1.3.3. Names with Special Characters

The creation of any REST resource requires a name that is part of the URL, and in case this name contains any special characters as defined in [Section 2.2 of the RFC 3986 spec](#), it is necessary to encode it with the [Percent encoding](#) mechanism.

### 1.3.4. Key-Content-Type Headers

Most REST API calls have the Key included in the URL. Data Grid assumes the Key is a *java.lang.String* when handling those calls, but you can use a specific header *Key-Content-Type* for keys in different formats.

### Key-Content-Type Header Examples

- Specifying a byte[] Key as a Base64 string:

API call:

```
PUT /my-cache/AQIDBDM=
```

Headers:

### Key-Content-Type: application/octet-stream

- Specifying a byte[] Key as a hexadecimal string:

API call:

```
GET /my-cache/0x01CA03042F
```

Headers:

```
Key-Content-Type: application/octet-stream; encoding=hex
```

- Specifying a double Key:

API call:

```
POST /my-cache/3.141456
```

Headers:

```
Key-Content-Type: application/x-java-object;type=java.lang.Double
```

The *type* parameter for **application/x-java-object** is restricted to:

- Primitive wrapper types
- java.lang.String
- Bytes, making **application/x-java-object;type=Bytes** equivalent to **application/octet-stream;encoding=hex**.

### 1.3.5. JSON/Protostream Conversion

When caches are indexed, or specifically configured to store `_application/x-protostream`, you can send and receive JSON documents that are automatically converted to and from Protobuf.

You must register a Protobuf schema for the conversion to work.

To register protobuf schemas via REST, invoke a POST or PUT in the `__protobuf_metadata` cache as in the following example:

```
curl -u user:password -X POST --data-binary @./schema.proto
http://127.0.0.1:11222/rest/v2/caches/__protobuf_metadata/schema.proto
```

When writing JSON documents, a special field `_type` must be present in the document to identity the Protobuf *Message* that corresponds to the document.

Person.proto

```
message Person {
  required string name = 1;
  required int32 age = 2;
}
```

Person.json

```
{
  "_type": "Person",
  "name": "user1",
  "age": 32
}
```

## 1.4. CROSS-ORIGIN RESOURCE SHARING (CORS) REQUESTS

The Data Grid REST connector supports [CORS](#), including preflight and rules based on the request origin.

The following shows an example REST connector configuration with CORS rules:

```
<rest-connector name="rest1" socket-binding="rest" cache-container="default">
  <cors-rules>
    <cors-rule name="restrict host1"
      allow-credentials="false">
      <allowed-origins>http://host1,https://host1</allowed-origins>
      <allowed-methods>GET</allowed-methods>
    </cors-rule>
    <cors-rule name="allow ALL"
      allow-credentials="true"
      max-age-seconds="2000">
      <allowed-origins>*</allowed-origins>
      <allowed-methods>GET,OPTIONS,POST,PUT,DELETE</allowed-methods>
      <allowed-headers>Key-Content-Type</allowed-headers>
    </cors-rule>
  </cors-rules>
</rest-connector>
```

Data Grid evaluates CORS rules sequentially based on the "Origin" header set by the browser.

In the preceding example, if the origin is either "http://host1" or "https://host1", then the rule "restrict host1" applies. If the origin is different, then the next rule is tested.

Because the "allow ALL" rule permits all origins, any script that has an origin other than "http://host1" or "https://host1" can perform the allowed methods and use the supplied headers.

For information about configuring CORS rules, see the [Data Grid Server Configuration Schema](#).

### 1.4.1. Allowing all CORS permissions for some origins



The VM property **infinispan.server.rest.cors-allow** can be used when starting the server to allow all permissions to one or more origins. Example:

```
./bin/server.sh -Dinfinispan.server.rest.cors-allow=http://192.168.1.78:11222,http://host.mydomain.com
```

All origins specified using this method will take precedence over the configured rules.

## CHAPTER 2. INTERACTING WITH THE DATA GRID REST API

The Data Grid REST API lets you monitor, maintain, and manage Data Grid deployments and provides access to your data.



### NOTE

By default Data Grid REST API operations return **200 (OK)** when successful. However, when some operations are processed successfully, they return an HTTP status code such as **204** or **202** instead of **200**.

## 2.1. CREATING AND MANAGING CACHES

Create and manage Data Grid caches and perform operations on data.

### 2.1.1. Creating Caches

Create named caches across Data Grid clusters with **POST** requests that include XML or JSON configuration in the payload.

```
POST /rest/v2/caches/{cacheName}
```

Table 2.1. Headers

Header	Required or Optional	Parameter
<b>Content-Type</b>	REQUIRED	Sets the <a href="#">MediaType</a> for the Data Grid configuration payload; either <b>application/xml</b> or <b>application/json</b> .
<b>Flags</b>	OPTIONAL	Used to set <a href="#">AdminFlags</a>

#### 2.1.1.1. Cache Configuration

You can provide cache configuration in XML or JSON format.

##### XML

```
<distributed-cache name="myCache" mode="SYNC">
  <encoding media-type="application/x-protostream"/>
  <memory max-count="1000000" when-full="REMOVE"/>
</distributed-cache>
```

##### JSON

```
{
  "distributed-cache": {
    "name": "myCache",
    "mode": "SYNC",
```

```

"encoding": {
  "media-type": "application/x-protostream"
},
"memory": {
  "max-count": 1000000,
  "when-full": "REMOVE"
}
}
}

```

### JSON format

Cache configuration in JSON format must follow the structure of an XML configuration. \* XML elements become JSON objects. \* XML attributes become JSON fields.

## 2.1.2. Verifying Caches

Check if caches are available in Data Grid clusters with **HEAD** requests.

```
HEAD /rest/v2/caches/{cacheName}
```

## 2.1.3. Creating Caches with Templates

Create caches from Data Grid templates with **POST** requests and the **?template=** parameter.

```
POST /rest/v2/caches/{cacheName}?template={templateName}
```

### TIP

See [Listing Available Cache Templates](#).

## 2.1.4. Retrieving Cache Configuration

Retrieve Data Grid cache configurations with **GET** requests.

```
GET /rest/v2/caches/{name}?action=config
```

Table 2.2. Headers

Header	Required or Optional	Parameter
<b>Accept</b>	OPTIONAL	Sets the required format to return content. Supported formats are <b>application/xml</b> and <b>application/json</b> . The default is <b>application/json</b> . See <a href="#">Accept</a> for more information.

## 2.1.5. Converting Cache Configurations to JSON

Invoke a **POST** request with valid XML configuration and the **?action=toJSON** parameter. Data Grid responds with the equivalent JSON representation of the configuration.

POST /rest/v2/caches?action=toJSON

## 2.1.6. Retrieving All Cache Details

Invoke a **GET** request to retrieve all details for Data Grid caches.

GET /rest/v2/caches/{name}

Data Grid provides a JSON response such as the following:

```
{
  "stats": {
    "time_since_start": -1,
    "time_since_reset": -1,
    "hits": -1,
    "current_number_of_entries": -1,
    "current_number_of_entries_in_memory": -1,
    "total_number_of_entries": -1,
    "stores": -1,
    "off_heap_memory_used": -1,
    "data_memory_used": -1,
    "retrievals": -1,
    "misses": -1,
    "remove_hits": -1,
    "remove_misses": -1,
    "evictions": -1,
    "average_read_time": -1,
    "average_read_time_nanos": -1,
    "average_write_time": -1,
    "average_write_time_nanos": -1,
    "average_remove_time": -1,
    "average_remove_time_nanos": -1,
    "required_minimum_number_of_nodes": -1
  },
  "size": 0,
  "configuration": {
    "distributed-cache": {
      "mode": "SYNC",
      "transaction": {
        "stop-timeout": 0,
        "mode": "NONE"
      }
    }
  },
  "rehash_in_progress": false,
  "bounded": false,
  "indexed": false,
  "persistent": false,
  "transactional": false,
  "secured": false,
  "has_remote_backup": false,
  "indexing_in_progress": false,
  "statistics": false
}
```

- **stats** current stats of the cache.
- **size** the estimated size for the cache.
- **configuration** the cache configuration.
- **rehash\_in\_progress** true when a rehashing is in progress.
- **indexing\_in\_progress** true when indexing is in progress.
- **bounded** when expiration is enabled.
- **indexed** true if the cache is indexed.
- **persistent** true if the cache is persisted.
- **transactional** true if the cache is transactional.
- **secured** true if the cache is secured.
- **has\_remote\_backup** true if the cache has remote backups.

### 2.1.7. Adding Entries

Add entries to caches with **POST** requests.

```
POST /rest/v2/caches/{cacheName}/{cacheKey}
```

The preceding request places the payload, or request body, in the **cacheName** cache with the **cacheKey** key. The request replaces any data that already exists and updates the **Time-To-Live** and **Last-Modified** values, if they apply.

If the entry is created successfully, the service returns **204 (No Content)**.

If a value already exists for the specified key, the **POST** request returns **409 (Conflict)** and does not modify the value. To update values, you should use **PUT** requests. See [Replacing Entries](#).

Table 2.3. Headers

Header	Required or Optional	Parameter
<b>Key-Content-Type</b>	OPTIONAL	Sets the content type for the key in the request. See <a href="#">Key-Content-Type</a> for more information.
<b>Content-Type</b>	OPTIONAL	Sets the <a href="#">MediaType</a> of the value for the key.

Header	Required or Optional	Parameter
<b>timeToLiveSeconds</b>	OPTIONAL	Sets the number of seconds before the entry is automatically deleted. If you do not set this parameter, Data Grid uses the default value from the configuration. If you set a negative value, the entry is never deleted.
<b>maxIdleTimeSeconds</b>	OPTIONAL	Sets the number of seconds that entries can be idle. If a read or write operation does not occur for an entry after the maximum idle time elapses, the entry is automatically deleted. If you do not set this parameter, Data Grid uses the default value from the configuration. If you set a negative value, the entry is never deleted.
<b>flags</b>	OPTIONAL	The flags used to add the entry. See <a href="#">Flag</a> for more information.

**NOTE**

The **flags** header also applies to all other operations involving data manipulation on the cache,

**NOTE**

If both **timeToLiveSeconds** and **maxIdleTimeSeconds** have a value of **0**, Data Grid uses the default **lifespan** and **maxIdle** values from the configuration.

If *only* **maxIdleTimeSeconds** has a value of **0**, Data Grid uses:

- the default **maxIdle** value from the configuration.
- the value for **timeToLiveSeconds** that you pass as a request parameter or a value of **-1** if you do not pass a value.

If *only* **timeToLiveSeconds** has a value of **0**, Data Grid uses:

- the default **lifespan** value from the configuration.
- the value for **maxIdle** that you pass as a request parameter or a value of **-1** if you do not pass a value.

## 2.1.8. Replacing Entries

Replace entries in caches with **PUT** requests.

PUT /rest/v2/caches/{cacheName}/{cacheKey}

If a value already exists for the specified key, the **PUT** request updates the value. If you do not want to modify existing values, use **POST** requests that return **409 (Conflict)** instead of modifying values. See [Adding Values](#).

### 2.1.9. Retrieving Data By Keys

Retrieve data for specific keys with **GET** requests.

GET /rest/v2/caches/{cacheName}/{cacheKey}

The server returns data from the given cache, **cacheName**, under the given key, **cacheKey**, in the response body. Responses contain **Content-Type** headers that correspond to the **MediaType** negotiation.



#### NOTE

Browsers can also access caches directly, for example as a content delivery network (CDN). Data Grid returns a unique **ETag** for each entry along with the **Last-Modified** and **Expires** header fields.

These fields provide information about the state of the data that is returned in your request. ETags allow browsers and other clients to request only data that has changed, which conserves bandwidth.

Table 2.4. Headers

Header	Required or Optional	Parameter
<b>Key-Content-Type</b>	OPTIONAL	Sets the content type for the key in the request. The default is <b>application/x-java-object; type=java.lang.String</b> . See <a href="#">Key-Content-Type</a> for more information.
<b>Accept</b>	OPTIONAL	Sets the required format to return content. See <a href="#">Accept</a> for more information.

**TIP**

Append the **extended** parameter to the query string to get additional information:

```
GET /rest/v2/caches/{cacheName}/{cacheKey}?extended
```

The preceding request returns custom headers:

- **Cluster-Primary-Owner** returns the node name that is the primary owner of the key.
- **Cluster-Node-Name** returns the JGroups node name of the server that handled the request.
- **Cluster-Physical-Address** returns the physical JGroups address of the server that handled the request.

**2.1.10. Checking if Entries Exist**

Verify that specific entries exists with **HEAD** requests.

```
HEAD /rest/v2/caches/{cacheName}/{cacheKey}
```

The preceding request returns only the header fields and the same content that you stored with the entry. For example, if you stored a String, the request returns a String. If you stored binary, base64-encoded, blobs or serialized Java objects, Data Grid does not de-serialize the content in the request.

**NOTE**

**HEAD** requests also support the **extended** parameter.

Table 2.5. Headers

Header	Required or Optional	Parameter
<b>Key-Content-Type</b>	OPTIONAL	Sets the content type for the key in the request. The default is <b>application/x-java-object; type=java.lang.String</b> . See <a href="#">Key-Content-Type</a> for more information.

**2.1.11. Deleting Entries**

Remove entries from caches with **DELETE** requests.

```
DELETE /rest/v2/caches/{cacheName}/{cacheKey}
```

Table 2.6. Headers

Header	Required or Optional	Parameter
--------	----------------------	-----------



Header	Required or Optional	Parameter
<b>Key-Content-Type</b>	OPTIONAL	Sets the content type for the key in the request. The default is <b>application/x-java-object; type=java.lang.String</b> . See <a href="#">Key-Content-Type</a> for more information.

### 2.1.12. Deleting Caches

Remove caches from Data Grid clusters with **DELETE** requests.

```
DELETE /rest/v2/caches/{cacheName}
```

### 2.1.13. Retrieving All Keys from Caches

Invoke **GET** requests to retrieve all the keys in a cache in JSON format.

```
GET /rest/v2/caches/{cacheName}?action=keys
```

Table 2.7. Request Parameters

Parameter	Required or Optional	Value
<b>limit</b>	OPTIONAL	Specifies the maximum number of keys to retrieve using an <code>InputStream</code> . A negative value retrieves all keys. The default value is <b>-1</b> .
<b>batch</b>	OPTIONAL	Specifies the internal batch size when retrieving the keys. The default value is <b>1000</b> .

### 2.1.14. Retrieving All Entries from Caches

Invoke **GET** requests to retrieve all the entries in a cache in JSON format.

```
GET /rest/v2/caches/{cacheName}?action=entries
```

Table 2.8. Request Parameters

Parameter	Required or Optional	Value
-----------	----------------------	-------

Parameter	Required or Optional	Value
<b>metadata</b>	OPTIONAL	Includes metadata for each entry in the response. The default value is <b>false</b> .
<b>limit</b>	OPTIONAL	Specifies the maximum number of keys to include in the response. A negative value retrieves all keys. The default value is <b>-1</b> .
<b>batch</b>	OPTIONAL	Specifies the internal batch size when retrieving the keys. The default value is <b>1000</b> .

Data Grid provides a JSON response such as the following:

```
[
  {
    "key":1,
    "value":"value1",
    "timeToLiveSeconds":-1,
    "maxIdleTimeSeconds":-1,
    "created":-1,
    "lastUsed":-1,
    "expireTime":-1
  },
  {
    "key":2,
    "value":"value2",
    "timeToLiveSeconds":10,
    "maxIdleTimeSeconds":45,
    "created":1607966017944,
    "lastUsed": 1607966017944,
    "expireTime":1607966027944
  }
]
```

- **key** The key for the entry.
- **value** The value of the entry.
- **timeToLiveSeconds** Based on the entry lifespan but in seconds, or **-1** if the entry never expires. It's not returned unless you set metadata="true".
- **maxIdleTimeSeconds** Maximum idle time, in seconds, or **-1** if entry never expires. It's not returned unless you set metadata="true".
- **created** Time the entry was created or or **-1** for immortal entries. It's not returned unless you set metadata="true".

- **lastUsed** Last time an operation was performed on the entry or **-1** for immortal entries. It's not returned unless you set `metadata="true"`.
- **expireTime** Time when the entry expires or **-1** for immortal entries. It's not returned unless you set `metadata="true"`.

### 2.1.15. Clearing Caches

To delete all data from a cache, invoke a **POST** request with the **?action=clear** parameter.

```
POST /rest/v2/caches/{cacheName}?action=clear
```

If the operation successfully completes, the service returns **204 (No Content)**.

### 2.1.16. Getting Cache Size

Retrieve the size of caches across the entire cluster with **GET** requests and the **?action=size** parameter.

```
GET /rest/v2/caches/{cacheName}?action=size
```

### 2.1.17. Getting Cache Statistics

Obtain runtime statistics for caches with **GET** requests.

```
GET /rest/v2/caches/{cacheName}?action=stats
```

### 2.1.18. Querying Caches

Perform Ickle queries on caches with **GET** requests and the **?action=search&query** parameter.

```
GET /rest/v2/caches/{cacheName}?action=search&query={ickle query}
```

Data Grid responds with query hits such as the following:

```
{
  "total_results" : 150,
  "hits" : [ {
    "hit" : {
      "name" : "user1",
      "age" : 35
    }
  }, {
    "hit" : {
      "name" : "user2",
      "age" : 42
    }
  }, {
    "hit" : {
      "name" : "user3",
      "age" : 12
    }
  }
]
```

```

    }
  }
}

```

- **total\_results** displays the total number of results from the query.
- **hits** is an array of matches from the query.
- **hit** is an object that matches the query.

#### TIP

Hits can contain all fields or a subset of fields if you use a **Select** clause.

Table 2.9. Request Parameters

Parameter	Required or Optional	Value
<b>query</b>	REQUIRED	Specifies the query string.
<b>max_results</b>	OPTIONAL	Sets the number of results to return. The default is <b>10</b> .
<b>offset</b>	OPTIONAL	Specifies the index of the first result to return. The default is <b>0</b> .
<b>query_mode</b>	OPTIONAL	Specifies how the Data Grid server executes the query. Values are <b>FETCH</b> and <b>BROADCAST</b> . The default is <b>FETCH</b> .

To use the body of the request instead of specifying query parameters, invoke **POST** requests as follows:

```
POST /rest/v2/caches/{cacheName}?action=search
```

The following example shows a query in the request body:

```

{
  "query":"from Entity where name:\\"user1\\"",
  "max_results":20,
  "offset":10
}

```

### 2.1.19. Re-indexing Data

Re-index all data in caches with **POST** requests and the **?action=mass-index&mode={mode}** parameter.

```
POST /v2/caches/{cacheName}/search/indexes?action=mass-index&mode={mode}
```

Values for the **mode** parameter are as follows:

- **sync** returns **204 (No Content)** only after the re-indexing operation is complete.
- **async** returns **204 (No Content)** immediately and the re-indexing operation continues running in the cluster. You can check the status with the [Index Statistics](#) REST call.

### 2.1.20. Purging Indexes

Delete all indexes from caches with **POST** requests and the **?action=clear** parameter.

```
POST /v2/caches/{cacheName}/search/indexes?action=clear
```

If the operation successfully completes, the service returns **204 (No Content)**.

### 2.1.21. Retrieving Query and Index Statistics

Obtain information about queries and indexes in caches with **GET** requests.



#### NOTE

Statistics must be enabled in the cache otherwise the results will be empty.

```
GET /v2/caches/{cacheName}/search/stats
```

Table 2.10. Request Parameters

Parameter	Required or Optional	Value
<b>scope</b>	OPTIONAL	Use <b>cluster</b> to retrieve consolidated statistics for all members of the cluster. When omitted, Data Grid returns statistics for the local queries and indexes.

Data Grid provides a JSON response such as the following:

```
{
  "query": {
    "indexed_local": {
      "count": 1,
      "average": 12344.2,
      "max": 122324,
      "slowest": "FROM Entity WHERE field > 4"
    },
    "indexed_distributed": {
      "count": 0,
      "average": 0.0,
      "max": -1,
      "slowest": "FROM Entity WHERE field > 4"
    }
  },
}
```

```

"hybrid": {
  "count": 0,
  "average": 0.0,
  "max": -1,
  "slowest": "FROM Entity WHERE field > 4 AND desc = 'value'"
},
"non_indexed": {
  "count": 0,
  "average": 0.0,
  "max": -1,
  "slowest": "FROM Entity WHERE desc = 'value'"
},
"entity_load": {
  "count": 123,
  "average": 10.0,
  "max": 120
}
},
"index": {
  "types": {
    "org.infinispan.same.test.Entity": {
      "count": 5660001,
      "size": 0
    },
    "org.infinispan.same.test.AnotherEntity": {
      "count": 40,
      "size": 345560
    }
  },
  "reindexing": false
}
}

```

In the **query** section:

- **indexed\_local** Provides details about indexed queries.
- **indexed\_distributed** Provides details about distributed indexed queries.
- **hybrid** Provides details about queries that used the index only partially.
- **non\_indexed** Provides details about queries that didn't use the index.
- **entity\_load** Provides details about cache operations to fetch objects after indexed queries execution.



#### NOTE

All time related statistics are in nanoseconds.

In the **index** section:

- **types** Provide details about each indexed type (class name or protobuf message) that is configured in the cache.
  - **count** The number of entities indexed for the type.

- **size** Usage in bytes of the type.
- **reindexing** If the value is **true**, the **Indexer** is running in the cache.

### 2.1.22. Clearing Search Statistics

Reset runtime statistics with **POST** requests and the **?action=clear** parameter.

```
POST /v2/caches/{cacheName}/search/stats?action=clear
```

Index statistics will not be cleared, but only query execution times. Data Grid clears query statistics for the local node only.

### 2.1.23. Retrieving Index Statistics (Deprecated)

Obtain information about indexes in caches with **GET** requests.

```
GET /v2/caches/{cacheName}/search/indexes/stats
```

Data Grid provides a JSON response such as the following:

```
{
  "indexed_class_names": ["org.infinispan.sample.User"],
  "indexed_entities_count": {
    "org.infinispan.sample.User": 4
  },
  "index_sizes": {
    "cacheName_protobuf": 14551
  },
  "reindexing": false
}
```

- **indexed\_class\_names** Provides the class names of the indexes present in the cache. For Protobuf the value is always **org.infinispan.query.remote.impl.indexing.ProtobufValueWrapper**.
- **indexed\_entities\_count** Provides the number of entities indexed per class.
- **index\_sizes** Provides the size, in bytes, for each index in the cache.
- **reindexing** Indicates if a re-indexing operation was performed for the cache. If the value is **true**, the **MassIndexer** was started in the cache.

### 2.1.24. Retrieving Query Statistics (Deprecated)

Get information about the queries that have been run in caches with **GET** requests.

```
GET /v2/caches/{cacheName}/search/query/stats
```

Data Grid provides a JSON response such as the following:

```
{
  "search_query_execution_count":20,
```

```

"search_query_total_time":5,
"search_query_execution_max_time":154,
"search_query_execution_avg_time":2,
"object_loading_total_time":1,
"object_loading_execution_max_time":1,
"object_loading_execution_avg_time":1,
"objects_loaded_count":20,
"search_query_execution_max_time_query_string": "FROM entity"
}

```

- **search\_query\_execution\_count** Provides the number of queries that have been run.
- **search\_query\_total\_time** Provides the total time spent on queries.
- **search\_query\_execution\_max\_time** Provides the maximum time taken for a query.
- **search\_query\_execution\_avg\_time** Provides the average query time.
- **object\_loading\_total\_time** Provides the total time spent loading objects from the cache after query execution.
- **object\_loading\_execution\_max\_time** Provides the maximum time spent loading objects execution.
- **object\_loading\_execution\_avg\_time** Provides the average time spent loading objects execution.
- **objects\_loaded\_count** Provides the count of objects loaded.
- **search\_query\_execution\_max\_time\_query\_string** Provides the slowest query executed.

### 2.1.25. Clearing Query Statistics (Deprecated)

Reset runtime statistics with **POST** requests and the **?action=clear** parameter.

```
POST /v2/caches/{cacheName}/search/query/stats?action=clear
```

### 2.1.26. Listing Caches

List all available caches in Data Grid clusters with **GET** requests.

```
GET /rest/v2/caches/
```

### 2.1.27. Cross-Site Operations with Caches

Perform cross-site replication operations with the Data Grid REST API.

#### 2.1.27.1. Getting Status of All Backup Locations

Retrieve the status of all backup locations with **GET** requests.

```
GET /v2/caches/{cacheName}/x-site/backups/
```

Data Grid responds with the status of each backup location in JSON format, as in the following example:



```
{
  "NYC": "online",
  "LON": "offline"
}
```

Table 2.11. Returned Status

Value	Description
<b>online</b>	All nodes in the local cluster have a cross-site view with the backup location.
<b>offline</b>	No nodes in the local cluster have a cross-site view with the backup location.
<b>mixed</b>	Some nodes in the local cluster have a cross-site view with the backup location, other nodes in the local cluster do not have a cross-site view. The response indicates status for each node.

### 2.1.27.2. Getting Status of Specific Backup Locations

Retrieve the status of a backup location with **GET** requests.

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}
```

Data Grid responds with the status of each node in the site in JSON format, as in the following example:

```
{
  "NodeA": "offline",
  "NodeB": "online"
}
```

Table 2.12. Returned Status

Value	Description
<b>online</b>	The node is online.
<b>offline</b>	The node is offline.
<b>failed</b>	Not possible to retrieve status. The remote cache could be shutting down or a network error occurred during the request.

### 2.1.27.3. Taking Backup Locations Offline

Take backup locations offline with **POST** requests and the **?action=take-offline** parameter.

```
POST /v2/caches/{cacheName}/x-site/backups/{siteName}?action=take-offline
```

#### 2.1.27.4. Bringing Backup Locations Online

Bring backup locations online with the **?action=bring-online** parameter.

```
POST /v2/caches/{cacheName}/x-site/backups/{siteName}?action=bring-online
```

#### 2.1.27.5. Pushing State to Backup Locations

Push cache state to a backup location with the **?action=start-push-state** parameter.

```
POST /v2/caches/{cacheName}/x-site/backups/{siteName}?action=start-push-state
```

#### 2.1.27.6. Canceling State Transfer

Cancel state transfer operations with the **?action=cancel-push-state** parameter.

```
POST /v2/caches/{cacheName}/x-site/backups/{siteName}?action=cancel-push-state
```

#### 2.1.27.7. Getting State Transfer Status

Retrieve status of state transfer operations with the **?action=push-state-status** parameter.

```
GET /v2/caches/{cacheName}/x-site/backups?action=push-state-status
```

Data Grid responds with the status of state transfer for each backup location in JSON format, as in the following example:

```
{
  "NYC":"CANCELED",
  "LON":"OK"
}
```

Table 2.13. Returned Status

Value	Description
<b>SENDING</b>	State transfer to the backup location is in progress.
<b>OK</b>	State transfer completed successfully.
<b>ERROR</b>	An error occurred with state transfer. Check log files.
<b>CANCELLING</b>	State transfer cancellation is in progress.

#### 2.1.27.8. Clearing State Transfer Status

Clear state transfer status for sending sites with the **?action=clear-push-state-status** parameter.

```
POST /v2/caches/{cacheName}/x-site/local?action=clear-push-state-status
```

### 2.1.27.9. Modifying Take Offline Conditions

Sites go offline if certain conditions are met. Modify the take offline parameters to control when backup locations automatically go offline.

#### Procedure

1. Check configured take offline parameters with **GET** requests and the **take-offline-config** parameter.

```
GET /v2/caches/{cacheName}/x-site/backups/{siteName}/take-offline-config
```

The Data Grid response includes **after\_failures** and **min\_wait** fields as follows:

```
{
  "after_failures": 2,
  "min_wait": 1000
}
```

2. Modify take offline parameters in the body of **PUT** requests.

```
PUT /v2/caches/{cacheName}/x-site/backups/{siteName}/take-offline-config
```

If the operation successfully completes, the service returns **204 (No Content)**.

### 2.1.27.10. Canceling State Transfer from Receiving Sites

If the connection between two backup locations breaks, you can cancel state transfer on the site that is receiving the push.

Cancel state transfer from a remote site and keep the current state of the local cache with the **?action=cancel-receive-state** parameter.

```
POST /v2/caches/{cacheName}/x-site/backups/{siteName}?action=cancel-receive-state
```

## 2.1.28. Rolling Upgrades

Perform rolling upgrades of cache data between Data Grid clusters

### 2.1.28.1. Synchronizing Data

Synchronize data from a source cluster to a target cluster with **POST** requests and the **?action=sync-data** parameter:

```
POST /v2/caches/{cacheName}?action=sync-data
```

When the operation completes, Data Grid responds with the total number of entries copied to the target cluster.

### 2.1.28.2. Disconnecting Source Clusters

After you synchronize data to target clusters, disconnect from the source cluster with **POST** requests and the **?action=disconnect-source** parameter:

```
POST /v2/caches/{cacheName}?action=disconnect-source
```

If the operation successfully completes, the service returns **204 (No Content)**.

## 2.2. CREATING AND MANAGING COUNTERS

Create, delete, and modify counters via the REST API.

### 2.2.1. Creating Counters

Create counters with **POST** requests that include configuration in the payload.

```
POST /rest/v2/counters/{counterName}
```

#### Example Weak Counter

```
{
  "weak-counter":{
    "initial-value":5,
    "storage":"PERSISTENT",
    "concurrency-level":1
  }
}
```

#### Example Strong Counter

```
{
  "strong-counter":{
    "initial-value":3,
    "storage":"PERSISTENT",
    "upper-bound":5
  }
}
```

### 2.2.2. Deleting Counters

Remove specific counters with **DELETE** requests.

```
DELETE /rest/v2/counters/{counterName}
```

### 2.2.3. Retrieving Counter Configuration

Retrieve configuration for specific counters with **GET** requests.

```
GET /rest/v2/counters/{counterName}/config
```

Data Grid responds with the counter configuration in JSON format.

### 2.2.4. Getting Counter Values

Retrieve counter values with **GET** requests.

```
GET /rest/v2/counters/{counterName}
```

Table 2.14. Headers

Header	Required or Optional	Parameter
<b>Accept</b>	OPTIONAL	The required format to return the content. Supported formats are <i>application/json</i> and <i>text/plain</i> . JSON is assumed if no header is provided.

### 2.2.5. Resetting Counters

Restore the initial value of counters without **POST** requests and the **?action=reset** parameter.

```
POST /rest/v2/counters/{counterName}?action=reset
```

If the operation successfully completes, the service returns **204 (No Content)**.

### 2.2.6. Incrementing Counters

Increment counter values with **POST** request and the **?action=increment** parameter.

```
POST /rest/v2/counters/{counterName}?action=increment
```



#### NOTE

**WEAK** counters never respond after operations and return **204 (No Content)**.

**STRONG** counters return **200 (OK)** and the current value after each operation.

### 2.2.7. Adding Deltas to Counters

Add arbitrary values to counters with **POST** requests that include the **?action=add** and **delta** parameters.

```
POST /rest/v2/counters/{counterName}?action=add&delta={delta}
```



#### NOTE

**WEAK** counters never respond after operations and return **204 (No Content)**.

**STRONG** counters return **200 (OK)** and the current value after each operation.

### 2.2.8. Decrementing Counter Values

Decrement counter values with **POST** requests and the **?action=decrement** parameter.

```
POST /rest/v2/counters/{counterName}?action=decrement
```

**NOTE**

**WEAK** counters never respond after operations and return **204 (No Content)**.

**STRONG** counters return **200 (OK)** and the current value after each operation.

### 2.2.9. Performing compareAndSet Operations on Strong Counters

Atomically set values for strong counters with **GET** requests and the **compareAndSet** parameter.

```
POST /rest/v2/counters/{counterName}?action=compareAndSet&expect={expect}&update={update}
```

Data Grid atomically sets the value to **{update}** if the current value is **{expect}**. If the operation is successful, Data Grid returns **true**.

### 2.2.10. Performing compareAndSwap Operations on Strong Counters

Atomically set values for strong counters with **GET** requests and the **compareAndSwap** parameter.

```
POST /rest/v2/counters/{counterName}?action=compareAndSwap&expect={expect}&update={update}
```

Data Grid atomically sets the value to **{update}** if the current value is **{expect}**. If the operation is successful, Data Grid returns the previous value in the payload.

### 2.2.11. Listing Counters

Retrieve a list of counters in Data Grid clusters with **GET** requests.

```
GET /rest/v2/counters/
```

## 2.3. WORKING WITH PROTOBUF SCHEMAS

Create and manage Protobuf schemas, **.proto** files, via the Data Grid REST API.

### 2.3.1. Creating Protobuf Schemas

Create Protobuf schemas across Data Grid clusters with **POST** requests that include the content of a protobuf file in the payload.

```
POST /rest/v2/schemas/{schemaName}
```

If the schema already exists, Data Grid returns HTTP **409 (Conflict)**. If the schema is not valid, either because of syntax errors, or because some of its dependencies are missing, Data Grid stores the schema and returns the error in the response body.

Data Grid responds with the schema name and any errors.

```
{
  "name" : "users.proto",
  "error" : {
    "message": "Schema users.proto has errors",
  }
}
```

```
"cause": "java.lang.IllegalStateException:Syntax error in error.proto at 3:8: unexpected label:
messoge"
}
}
```

- **name** is the name of the Protobuf schema.
- **error** is **null** for valid Protobuf schemas. If Data Grid cannot successfully validate the schema, it returns errors.

If the operation successfully completes, the service returns **201 (Created)**.

### 2.3.2. Reading Protobuf Schemas

Retrieve Protobuf schema from Data Grid with **GET** requests.

```
GET /rest/v2/schemas/{schemaName}
```

### 2.3.3. Updating Protobuf Schemas

Modify Protobuf schemas with **PUT** requests that include the content of a protobuf file in the payload.

```
PUT /rest/v2/schemas/{schemaName}
```

If the schema is not valid, either because of syntax errors, or because some of its dependencies are missing, Data Grid updates the schema and returns the error in the response body.

```
{
  "name" : "users.proto",
  "error" : {
    "message": "Schema users.proto has errors",
    "cause": "java.lang.IllegalStateException:Syntax error in error.proto at 3:8: unexpected label:
messoge"
  }
}
```

- **name** is the name of the Protobuf schema.
- **error** is **null** for valid Protobuf schemas. If Data Grid cannot successfully validate the schema, it returns errors.

### 2.3.4. Deleting Protobuf Schemas

Remove Protobuf schemas from Data Grid clusters with **DELETE** requests.

```
DELETE /rest/v2/schemas/{schemaName}
```

If the operation successfully completes, the service returns **204 (No Content)**.

### 2.3.5. Listing Protobuf Schemas

List all available Protobuf schemas with **GET** requests.

```
GET /rest/v2/schemas/
```

Data Grid responds with a list of all schemas available on the cluster.

```
[ {
  "name" : "users.proto",
  "error" : {
    "message": "Schema users.proto has errors",
    "cause": "java.lang.IllegalStateException:Syntax error in error.proto at 3:8: unexpected label:
messoge"
  }
}, {
  "name" : "people.proto",
  "error" : null
}]
```

- **name** is the name of the Protobuf schema.
- **error** is **null** for valid Protobuf schemas. If Data Grid cannot successfully validate the schema, it returns errors.

## 2.4. WORKING WITH CACHE MANAGERS

Interact with Data Grid Cache Managers to get cluster and usage statistics.

### 2.4.1. Getting Basic Cache Manager Information

Retrieving information about Cache Managers with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}
```

Data Grid responds with information in JSON format, as in the following example:

```
{
  "version":"xx.x.x-FINAL",
  "name":"default",
  "coordinator":true,
  "cache_configuration_names":[
    "__protobuf_metadata",
    "cache2",
    "CacheManagerResourceTest",
    "cache1"
  ],
  "cluster_name":"ISPN",
  "physical_addresses":"[127.0.0.1:35770]",
  "coordinator_address":"CacheManagerResourceTest-NodeA-49696",
  "cache_manager_status":"RUNNING",
  "created_cache_count":"3",
  "running_cache_count":"3",
  "node_address":"CacheManagerResourceTest-NodeA-49696",
  "cluster_members":[
    "CacheManagerResourceTest-NodeA-49696",
    "CacheManagerResourceTest-NodeB-28120"
  ],
}
```



```

"cluster_members_physical_addresses":[
  "127.0.0.1:35770",
  "127.0.0.1:60031"
],
"cluster_size":2,
"defined_caches":[
  {
    "name":"CacheManagerResourceTest",
    "started":true
  },
  {
    "name":"cache1",
    "started":true
  },
  {
    "name":"__protobuf_metadata",
    "started":true
  },
  {
    "name":"cache2",
    "started":true
  }
],
"local_site": "LON",
"sites_view": [
  "LON",
  "NYC"
]
}

```

- **version** contains the Data Grid version
- **name** contains the name of the cache manager as defined in the configuration
- **coordinator** is true if the cache manager is the coordinator of the cluster
- **cache\_configuration\_names** contains an array of all caches configurations defined in the cache manager
- **cluster\_name** contains the name of the cluster as defined in the configuration
- **physical\_addresses** contains the physical network addresses associated with the cache manager
- **coordinator\_address** contains the physical network addresses of the coordinator of the cluster
- **cache\_manager\_status** the lifecycle status of the cache manager. For possible values, check the [org.infinispan.lifecycle.ComponentStatus](#) documentation
- **created\_cache\_count** number of created caches, excludes all internal and private caches
- **running\_cache\_count** number of created caches that are running
- **node\_address** contains the logical address of the cache manager
- **cluster\_members** and **cluster\_members\_physical\_addresses** an array of logical and physical addresses of the members of the cluster

- **cluster\_size** number of members in the cluster
- **defined\_caches** A list of all caches defined in the cache manager, excluding private caches but including internal caches that are accessible
- **local\_site** The name of the local site.  
If cross-site replication is not configured, Data Grid returns "local".
- **sites\_view** The list of sites that participate in cross-site replication.  
If cross-site replication is not configured, Data Grid returns an empty list.

## 2.4.2. Getting Cluster Health

Retrieve health information for Data Grid clusters with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/health
```

Data Grid responds with cluster health information in JSON format, as in the following example:

```
{
  "cluster_health":{
    "cluster_name":"ISPN",
    "health_status":"HEALTHY",
    "number_of_nodes":2,
    "node_names":[
      "NodeA-36229",
      "NodeB-28703"
    ]
  },
  "cache_health":[
    {
      "status":"HEALTHY",
      "cache_name":"__protobuf_metadata"
    },
    {
      "status":"HEALTHY",
      "cache_name":"cache2"
    },
    {
      "status":"HEALTHY",
      "cache_name":"mycache"
    },
    {
      "status":"HEALTHY",
      "cache_name":"cache1"
    }
  ]
}
```

- **cluster\_health** contains the health of the cluster
  - **cluster\_name** specifies the name of the cluster as defined in the configuration.
  - **health\_status** provides one of the following:

- **DEGRADED** indicates at least one of the caches is in degraded mode.
  - **HEALTHY\_REBALANCING** indicates at least one cache is in the rebalancing state.
  - **HEALTHY** indicates all cache instances in the cluster are operating as expected.
  - **FAILED** indicates the cache failed to start with the provided configuration.
- **number\_of\_nodes** displays the total number of cluster members. Returns a value of **0** for non-clustered (standalone) servers.
  - **node\_names** is an array of all cluster members. Empty for standalone servers.
- **cache\_health** contains health information per-cache
    - **status** HEALTHY, DEGRADED, HEALTHY\_REBALANCING or FAILED
    - **cache\_name** the name of the cache as defined in the configuration.

### 2.4.3. Getting Cache Manager Health Status

Retrieve the health status of Cache Managers with **GET** requests that do not require authentication.

```
GET /rest/v2/cache-managers/{cacheManagerName}/health/status
```

Data Grid responds with one of the following in **text/plain** format:

- **HEALTHY**
- **HEALTHY\_REBALANCING**
- **DEGRADED**
- **FAILED**

### 2.4.4. Checking REST Endpoint Availability

Verify Data Grid server REST endpoint availability with **HEAD** requests.

```
HEAD /rest/v2/cache-managers/{cacheManagerName}/health
```

If you receive a successful response code then the Data Grid REST server is running and serving requests.

### 2.4.5. Obtaining Global Configuration for Cache Managers

Retrieve global configuration for Cache Managers with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/config
```

Table 2.15. Headers

Header	Required or Optional	Parameter
<b>Accept</b>	OPTIONAL	The required format to return the content. Supported formats are <i>application/json</i> and <i>application/xml</i> . JSON is assumed if no header is provided.

## Reference

[GlobalConfiguration](#)

### 2.4.6. Obtaining Configuration for All Caches

Retrieve the configuration for all caches with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/cache-configs
```

Data Grid responds with **JSON** arrays that contain each cache and cache configuration, as in the following example:

```
[
  {
    "name":"cache1",
    "configuration":{
      "distributed-cache":{
        "mode":"SYNC",
        "partition-handling":{
          "when-split":"DENY_READ_WRITES"
        },
        "statistics":true
      }
    }
  },
  {
    "name":"cache2",
    "configuration":{
      "distributed-cache":{
        "mode":"SYNC",
        "transaction":{
          "mode":"NONE"
        }
      }
    }
  }
]
```

### 2.4.7. Listing Available Cache Templates

Retrieve all available Data Grid cache templates with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/cache-configs/templates
```

## TIP

See [Creating Caches with Templates](#).

### 2.4.8. (Experimental) Obtaining Cache Status and Information

Retrieve a list of all available caches for a Cache Manager, along with cache statuses and details, with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/caches
```

Data Grid responds with JSON arrays that lists and describes each available cache, as in the following example:

```
[ {
  "status" : "RUNNING",
  "name" : "cache1",
  "type" : "local-cache",
  "simple_cache" : false,
  "transactional" : false,
  "persistent" : false,
  "bounded" : false,
  "secured" : false,
  "indexed" : true,
  "has_remote_backup" : true,
  "health" : "HEALTHY"
}, {
  "status" : "RUNNING",
  "name" : "cache2",
  "type" : "distributed-cache",
  "simple_cache" : false,
  "transactional" : true,
  "persistent" : false,
  "bounded" : false,
  "secured" : false,
  "indexed" : true,
  "has_remote_backup" : true,
  "health" : "HEALTHY"
} ]
```

### 2.4.9. Getting Cache Manager Statistics

Retrieve the statistics for Cache Managers with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/stats
```

Data Grid responds with Cache Manager statistics in JSON format, as in the following example:

```
{
  "statistics_enabled" : true,
  "read_write_ratio" : 0.0,
}
```

```

"time_since_start":1,
"time_since_reset":1,
"number_of_entries":0,
"total_number_of_entries":0,
"off_heap_memory_used":0,
"data_memory_used":0,
"misses":0,
"remove_hits":0,
"remove_misses":0,
"evictions":0,
"average_read_time":0,
"average_read_time_nanos":0,
"average_write_time":0,
"average_write_time_nanos":0,
"average_remove_time":0,
"average_remove_time_nanos":0,
"required_minimum_number_of_nodes":1,
"hits":0,
"stores":0,
"current_number_of_entries_in_memory":0,
"hit_ratio":0.0,
"retrievals":0
}

```

- **statistics\_enabled** is **true** if statistics collection is enabled for the Cache Manager.
- **read\_write\_ratio** displays the read/write ratio across all caches.
- **time\_since\_start** shows the time, in seconds, since the Cache Manager started.
- **time\_since\_reset** shows the number of seconds since the Cache Manager statistics were last reset.
- **number\_of\_entries** shows the total number of entries currently in all caches from the Cache Manager. This statistic returns entries in the local cache instances only.
- **total\_number\_of\_entries** shows the number of store operations performed across all caches for the Cache Manager.
- **off\_heap\_memory\_used** shows the amount, in **bytes[]**, of off-heap memory used by this cache container.
- **data\_memory\_used** shows the amount, in **bytes[]**, that the current eviction algorithm estimates is in use for data across all caches. Returns **0** if eviction is not enabled.
- **misses** shows the number of **get()** misses across all caches.
- **remove\_hits** shows the number of removal hits across all caches.
- **remove\_misses** shows the number of removal misses across all caches.
- **evictions** shows the number of evictions across all caches.
- **average\_read\_time** shows the average number of milliseconds taken for **get()** operations across all caches.
- **average\_read\_time\_nanos** same as **average\_read\_time** but in nanoseconds.

- **average\_remove\_time** shows the average number of milliseconds for **remove()** operations across all caches.
- **average\_remove\_time\_nanos** same as **average\_remove\_time** but in nanoseconds.
- **required\_minimum\_number\_of\_nodes** shows the required minimum number of nodes to guarantee data consistency.
- **hits** provides the number of **get()** hits across all caches.
- **stores** provides the number of **put()** operations across all caches.
- **current\_number\_of\_entries\_in\_memory** shows the total number of entries currently in all caches, excluding passivated entries.
- **hit\_ratio** provides the total percentage hit/(hit+miss) ratio for all caches.
- **retrievals** shows the total number of **get()** operations.

### 2.4.10. Backing Up Data Grid Cache Managers

Create backup archives, **application/zip**, that contain resources (caches, cache templates, counters, Protobuf schemas, server tasks, and so on) currently stored in the cache manager.

POST /rest/v2/cache-managers/{cacheManagerName}/backups/{backupName}

If a backup with the same name already exists, the service responds with **409 (Conflict)**. If the **directory** parameter is not valid, the service returns **400 (Bad Request)**. A **202** response indicates that the backup request is accepted for processing.

Optionally include a JSON payload with your request that contains parameters for the backup operation, as follows:

Table 2.16. JSON Parameters

Key	Required or Optional	Value
<b>directory</b>	OPTIONAL	Specifies a location on the server to create and store the backup archive.
<b>resources</b>	OPTIONAL	Specifies the resources to back up, in JSON format. The default is to back up all resources. If you specify one or more resources, then Data Grid backs up only those resources. See the <i>Resource Parameters</i> table for more information.

Table 2.17. Resource Parameters

Key	Required or Optional	Value
<b>caches</b>	OPTIONAL	Specifies either an array of cache names to back up or * for all caches.
<b>cache-configs</b>	OPTIONAL	Specifies either an array of cache templates to back up or * for all templates.
<b>counters</b>	OPTIONAL	Defines either an array of counter names to back up or * for all counters.
<b>proto-schemas</b>	OPTIONAL	Defines either an array of Protobuf schema names to back up or * for all schemas.
<b>tasks</b>	OPTIONAL	Specifies either an array of server tasks to back up or * for all tasks.

The following example creates a backup archive with all counters and caches named **[cache1,cache2]** in a specified directory:

```
{
  "directory": "/some/path/accessible/to/the/server",
  "resources": {
    "caches": ["cache1", "cache2"],
    "counters": ["*"]
  }
}
```

### 2.4.11. Listing Backups

Retrieve the names of all backup operations that are in progress, completed, or failed.

```
GET /rest/v2/cache-managers/{cacheManagerName}/backups
```

Data Grid responds with an Array of all backup names as in the following example:

```
["backup1", "backup2"]
```

### 2.4.12. Checking Backup Availability

Verify that a backup operation is complete.

```
HEAD /rest/v2/cache-managers/{cacheManagerName}/backups/{backupName}
```



A **200** response indicates the backup archive is available. A **202** response indicates the backup operation is in progress.

### 2.4.13. Downloading Backup Archives

Download backup archives from the server.

```
GET /rest/v2/cache-managers/{cacheManagerName}/backups/{backupName}
```

A **200** response indicates the backup archive is available. A **202** response indicates the backup operation is in progress.

### 2.4.14. Deleting Backup Archives

Remove backup archives from the server.

```
DELETE /rest/v2/cache-managers/{cacheManagerName}/backups/{backupName}
```

A **204** response indicates that the backup archive is deleted. A **202** response indicates that the backup operation is in progress but will be deleted when the operation completes.

### 2.4.15. Restoring Data Grid Resources from Backup Archives

Restore Data Grid resources from backup archives. The provided **{restoreName}** is for tracking restore progress, and is independent of the name of backup file being restored.



#### IMPORTANT

You can restore resources only if the container name in the backup archive matches **{cacheManagerName}**.

```
POST /rest/v2/cache-managers/{cacheManagerName}/restores/{restoreName}
```

A **202** response indicates that the restore request has been accepted for processing.

#### 2.4.15.1. Restoring from Backup Archives on Data Grid Server

Use the **application/json** content type with your POST request to back up from an archive that is available on the server.

Table 2.18. JSON Parameters

Key	Required or Optional	Value
<b>location</b>	REQUIRED	Specifies the path of the backup archive to restore.

Key	Required or Optional	Value
<b>resources</b>	OPTIONAL	Specifies the resources to restore, in JSON format. The default is to restore all resources. If you specify one or more resources, then Data Grid restores only those resources. See the <i>Resource Parameters</i> table for more information.

Table 2.19. Resource Parameters

Key	Required or Optional	Value
<b>caches</b>	OPTIONAL	Specifies either an array of cache names to back up or * for all caches.
<b>cache-configs</b>	OPTIONAL	Specifies either an array of cache templates to back up or * for all templates.
<b>counters</b>	OPTIONAL	Defines either an array of counter names to back up or * for all counters.
<b>proto-schemas</b>	OPTIONAL	Defines either an array of Protobuf schema names to back up or * for all schemas.
<b>tasks</b>	OPTIONAL	Specifies either an array of server tasks to back up or * for all tasks.

The following example restores all counters from a backup archive on the server:

```
{
  "location": "/some/path/accessible/to/the/server/backup-to-restore.zip",
  "resources": {
    "counters": ["*"]
  }
}
```

#### 2.4.15.2. Restoring from Local Backup Archives

Use the **multipart/form-data** content type with your POST request to upload a local backup archive to the server.

Table 2.20. Form Data

Parameter	Content-Type	Required or Optional	Value
<b>backup</b>	<b>application/zip</b>	REQUIRED	Specifies the bytes of the backup archive to restore.
<b>resources</b>	<b>application/json, text/plain</b>	OPTIONAL	Defines a JSON object of request parameters.

### Example Request

```
Content-Type: multipart/form-data; boundary=5ec9bc07-f069-4662-a535-46069afeda32
Content-Length: 7721
```

```
--5ec9bc07-f069-4662-a535-46069afeda32
Content-Disposition: form-data; name="resources"
Content-Length: 23
```

```
{"scripts":["test.js"]}
--5ec9bc07-f069-4662-a535-46069afeda32
Content-Disposition: form-data; name="backup"; filename="testManagerRestoreParameters.zip"
Content-Type: application/zip
Content-Length: 7353
```

```
<zip-bytes>
--5ec9bc07-f069-4662-a535-46069afeda32--
```

### 2.4.16. Listing Restores

Retrieve the names of all restore requests that are in progress, completed, or failed.

```
GET /rest/v2/cache-managers/{cacheManagerName}/restores
```

Data Grid responds with an Array of all restore names as in the following example:

```
["restore1", "restore2"]
```

### 2.4.17. Checking Restore Progress

Verify that a restore operation is complete.

```
HEAD /rest/v2/cache-managers/{cacheManagerName}/restores/{restoreName}
```

A **201 (Created)** response indicates the restore operation is completed. A **202 (Accepted)** response indicates the backup operation is in progress.

### 2.4.18. Deleting Restore Metadata

Remove metadata for restore requests from the server. This action removes all metadata associated with restore requests but does not delete any restored content. If you delete the request metadata, you can use the request name to perform subsequent restore operations.

```
DELETE /rest/v2/cache-managers/{cacheManagerName}/restores/{restoreName}
```

A **204 (No Content)** response indicates that the restore metadata is deleted. A **202 (Accepted)** response indicates that the restore operation is in progress and will be deleted when the operation completes.

## 2.4.19. Cross-Site Operations with Cache Managers

Perform cross-site operations with Cache Managers to apply the operations to all caches.

### 2.4.19.1. Getting Status of Backup Locations

Retrieve the status of all backup locations from Cache Managers with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/
```

Data Grid responds with status in JSON format, as in the following example:

```
{
  "SFO-3":{
    "status":"online"
  },
  "NYC-2":{
    "status":"mixed",
    "online":[
      "CACHE_1"
    ],
    "offline":[
      "CACHE_2"
    ]
  }
}
```

Table 2.21. Returned Status

Value	Description
<b>online</b>	All nodes in the local cluster have a cross-site view with the backup location.
<b>offline</b>	No nodes in the local cluster have a cross-site view with the backup location.
<b>mixed</b>	Some nodes in the local cluster have a cross-site view with the backup location, other nodes in the local cluster do not have a cross-site view. The response indicates status for each node.

### 2.4.19.2. Taking Backup Locations Offline

Take backup locations offline with the **?action=take-offline** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=take-offline
```

### 2.4.19.3. Bringing Backup Locations Online

Bring backup locations online with the **?action=bring-online** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=bring-online
```

### 2.4.19.4. Retrieving the State Transfer Mode

Check the state transfer mode with **GET** requests.

```
GET /rest/v2/caches/{cacheName}/x-site/backups/{site}/state-transfer-mode
```

### 2.4.19.5. Setting the State Transfer Mode

Configure the state transfer mode with the **?action=set** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{site}/state-transfer-mode?action=set&mode={mode}
```

### 2.4.19.6. Starting State Transfer

Push state of all caches to remote sites with the **?action=start-push-state** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=start-push-state
```

### 2.4.19.7. Canceling State Transfer

Cancel ongoing state transfer operations with the **?action=cancel-push-state** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=cancel-push-state
```

## 2.5. WORKING WITH DATA GRID SERVERS

Monitor and manage Data Grid server instances.

### 2.5.1. Retrieving Basic Server Information

View basic information about Data Grid servers with **GET** requests.

```
GET /rest/v2/server
```

Data Grid responds with the server name, codename, and version in JSON format as in the following example:

```
{
  "version": "Infinispan 'Codename' xx.x.x.Final"
}
```

### 2.5.2. Getting Cache Managers

Retrieve lists of cache managers for Data Grid servers with **GET** requests.

```
GET /rest/v2/server/cache-managers
```

Data Grid responds with an array of the cache manager names configured for the server.



#### NOTE

Data Grid currently supports one cache manager per server only.

### 2.5.3. Adding Caches to Ignore Lists

Configure Data Grid to temporarily exclude specific caches from client requests. Send empty **POST** requests that include the names of the cache manager name and the cache.

```
POST /v2/server/ignored-caches/{cache-manager}/{cache}
```

Data Grid responds with **204 (No Content)** if the cache is successfully added to the ignore list or **404 (Not Found)** if the cache or cache manager are not found.



#### NOTE

Data Grid currently supports one cache manager per server only. For future compatibility you must provide the cache manager name in the requests.

### 2.5.4. Removing Caches from Ignore Lists

Remove caches from the ignore list with **DELETE** requests.

```
DELETE /v2/server/ignored-caches/{cache-manager}/{cache}
```

Data Grid responds with **204 (No Content)** if the cache is successfully removed from ignore list or **404 (Not Found)** if the cache or cache manager are not found.

### 2.5.5. Confirming Ignored Caches

Confirm that caches are ignored with **GET** requests.

```
GET /v2/server/ignored-caches/{cache-manager}
```

### 2.5.6. Obtaining Server Configuration

Retrieve Data Grid server configurations with **GET** requests.

```
GET /rest/v2/server/config
```

Data Grid responds with the configuration in JSON format, as follows:

```
{
  "server":{
    "interfaces":{
      "interface":{
        "name":"public",
        "inet-address":{
          "value":"127.0.0.1"
        }
      }
    },
    "socket-bindings":{
      "port-offset":0,
      "default-interface":"public",
      "socket-binding":[
        {
          "name":"memcached",
          "port":11221,
          "interface":"memcached"
        }
      ]
    },
    "security":{
      "security-realms":{
        "security-realm":{
          "name":"default"
        }
      }
    },
    "endpoints":{
      "socket-binding":"default",
      "security-realm":"default",
      "hotrod-connector":{
        "name":"hotrod"
      },
      "rest-connector":{
        "name":"rest"
      }
    }
  }
}
```

### 2.5.7. Getting Environment Variables

Retrieve all environment variables for Data Grid servers with **GET** requests.

```
GET /rest/v2/server/env
```

### 2.5.8. Getting JVM Memory Details

Retrieve JVM memory usage information for Data Grid servers with **GET** requests.

GET /rest/v2/server/memory

Data Grid responds with heap and non-heap memory statistics, direct memory usage, and information about memory pools and garbage collection in JSON format.

### 2.5.9. Getting JVM Thread Dumps

Retrieve the current thread dump for the JVM with **GET** requests.

GET /rest/v2/server/threads

Data Grid responds with the current thread dump in **text/plain** format.

### 2.5.10. Getting Diagnostic Reports for Data Grid Servers

Retrieve aggregated reports for Data Grid servers with **GET** requests.

GET /rest/v2/server/report

Data Grid responds with a **tar.gz** archive that contains an aggregated report with diagnostic information about both the Data Grid server and the host. The report provides details about CPU, memory, open files, network sockets and routing, threads, in addition to configuration and log files.

### 2.5.11. Stopping Data Grid Servers

Stop Data Grid servers with **POST** requests.

POST /rest/v2/server?action=stop

Data Grid responds with **204 (No Content)** and then stops running.

## 2.6. WORKING WITH DATA GRID CLUSTERS

Monitor and perform administrative tasks on Data Grid clusters.

### 2.6.1. Stopping Data Grid Clusters

Shut down entire Data Grid clusters with **POST** requests.

POST /rest/v2/cluster?action=stop

Data Grid responds with **204 (No Content)** and then performs an orderly shutdown of the entire cluster.

### 2.6.2. Stopping Specific Data Grid Servers in Clusters

Shut down one or more specific servers in Data Grid clusters with **GET** requests and the **?action=stop&server** parameter.

POST /rest/v2/cluster?action=stop&server={server1\_host}&server={server2\_host}

Data Grid responds with **204 (No Content)**.



### 2.6.3. Backing Up Data Grid Clusters

Create backup archives, **application/zip**, that contain resources (caches, templates, counters, Protobuf schemas, server tasks, and so on) currently stored in the cache container for the cluster.

```
POST /rest/v2/cluster/backups/{backupName}
```

Optionally include a JSON payload with your request that contains parameters for the backup operation, as follows:

Table 2.22. JSON Parameters

Key	Required or Optional	Value
<b>directory</b>	OPTIONAL	Specifies a location on the server to create and store the backup archive.

If the backup operation successfully completes, the service returns **202 (Accepted)**. If a backup with the same name already exists, the service returns **409 (Conflict)**. If the **directory** parameter is not valid, the service returns **400 (Bad Request)**.

### 2.6.4. Listing Backups

Retrieve the names of all backup operations that are in progress, completed, or failed.

```
GET /rest/v2/cluster/backups
```

Data Grid responds with an Array of all backup names as in the following example:

```
["backup1", "backup2"]
```

### 2.6.5. Checking Backup Availability

Verify that a backup operation is complete. A **200** response indicates the backup archive is available. A **202** response indicates the backup operation is in progress.

```
HEAD /rest/v2/cluster/backups/{backupName}
```

### 2.6.6. Downloading Backup Archives

Download backup archives from the server. A **200** response indicates the backup archive is available. A **202** response indicates the backup operation is in progress.

```
GET /rest/v2/cluster/backups/{backupName}
```

### 2.6.7. Deleting Backup Archives

Remove backup archives from the server. A **204** response indicates that the backup archive is deleted. A **202** response indicates that the backup operation is in progress but will be deleted when the operation completes.

DELETE /rest/v2/cluster/backups/{backupName}

## 2.6.8. Restoring Data Grid Cluster Resources

Apply resources in a backup archive to restore Data Grid clusters. The provided **{restoreName}** is for tracking restore progress, and is independent of the name of backup file being restored.



### IMPORTANT

You can restore resources only if the container name in the backup archive matches the container name for the cluster.

POST /rest/v2/cluster/restores/{restoreName}

A **202** response indicates that the restore request is accepted for processing.

### 2.6.8.1. Restoring from Backup Archives on Data Grid Server

Use the **application/json** content type with your POST request to back up from an archive that is available on the server.

Table 2.23. JSON Parameters

Key	Required or Optional	Value
<b>location</b>	REQUIRED	Specifies the path of the backup archive to restore.
<b>resources</b>	OPTIONAL	Specifies the resources to restore, in JSON format. The default is to restore all resources. If you specify one or more resources, then Data Grid restores only those resources. See the <i>Resource Parameters</i> table for more information.

Table 2.24. Resource Parameters

Key	Required or Optional	Value
<b>caches</b>	OPTIONAL	Specifies either an array of cache names to back up or * for all caches.
<b>cache-configs</b>	OPTIONAL	Specifies either an array of cache templates to back up or * for all templates.

Key	Required or Optional	Value
<b>counters</b>	OPTIONAL	Defines either an array of counter names to back up or * for all counters.
<b>proto-schemas</b>	OPTIONAL	Defines either an array of Protobuf schema names to back up or * for all schemas.
<b>tasks</b>	OPTIONAL	Specifies either an array of server tasks to back up or * for all tasks.

The following example restores all counters from a backup archive on the server:

```
{
  "location": "/some/path/accessible/to/the/server/backup-to-restore.zip",
  "resources": {
    "counters": ["*"]
  }
}
```

### 2.6.8.2. Restoring from Local Backup Archives

Use the **multipart/form-data** content type with your POST request to upload a local backup archive to the server.

Table 2.25. Form Data

Parameter	Content-Type	Required or Optional	Value
<b>backup</b>	<b>application/zip</b>	REQUIRED	Specifies the bytes of the backup archive to restore.

### Example Request

```
Content-Type: multipart/form-data; boundary=5ec9bc07-f069-4662-a535-46069afeda32
Content-Length: 7798

--5ec9bc07-f069-4662-a535-46069afeda32
Content-Disposition: form-data; name="backup"; filename="testManagerRestoreParameters.zip"
Content-Type: application/zip
Content-Length: 7353

<zip-bytes>
--5ec9bc07-f069-4662-a535-46069afeda32--
```

### 2.6.9. Listing Restores

Retrieve the names of all restore requests that are in progress, completed, or failed.

```
GET /rest/v2/cluster/restores
```

Data Grid responds with an Array of all restore names as in the following example:

```
["restore1", "restore2"]
```

### 2.6.10. Checking Restore Progress

Verify that a restore operation is complete.

```
HEAD /rest/v2/cluster/restores/{restoreName}
```

A **201 (Created)** response indicates the restore operation is completed. A **202** response indicates the backup operation is in progress.

### 2.6.11. Deleting Restore Metadata

Remove metadata for restore requests from the server. This action removes all metadata associated with restore requests but does not delete any restored content. If you delete the request metadata, you can use the request name to perform subsequent restore operations.

```
DELETE /rest/v2/cluster/restores/{restoreName}
```

A **204** response indicates that the restore metadata is deleted. A **202** response indicates that the restore operation is in progress and will be deleted when the operation completes.

## 2.7. DATA GRID SERVER LOGGING CONFIGURATION

View and modify the logging configuration on Data Grid clusters at runtime.

### 2.7.1. Listing the logging appenders

View a list of all configured appenders with **GET** requests.

```
GET /rest/v2/logging/appenders
```

Data Grid responds with a list of appenders in JSON format as in the following example:

```
{
  "STDOUT" : {
    "name" : "STDOUT"
  },
  "JSON-FILE" : {
    "name" : "JSON-FILE"
  },
  "HR-ACCESS-FILE" : {
    "name" : "HR-ACCESS-FILE"
  },
  "FILE" : {
    "name" : "FILE"
  }
}
```

```

    },
    "REST-ACCESS-FILE" : {
      "name" : "REST-ACCESS-FILE"
    }
  }
}

```

### 2.7.2. Listing the loggers

View a list of all configured loggers with **GET** requests.

```
GET /rest/v2/logging/loggers
```

Data Grid responds with a list of loggers in JSON format as in the following example:

```

[ {
  "name" : "",
  "level" : "INFO",
  "appenders" : [ "STDOUT", "FILE" ]
}, {
  "name" : "org.infinispan.HOTROD_ACCESS_LOG",
  "level" : "INFO",
  "appenders" : [ "HR-ACCESS-FILE" ]
}, {
  "name" : "com.arjuna",
  "level" : "WARN",
  "appenders" : [ ]
}, {
  "name" : "org.infinispan.REST_ACCESS_LOG",
  "level" : "INFO",
  "appenders" : [ "REST-ACCESS-FILE" ]
} ]

```

### 2.7.3. Creating/modifying a logger

Create a new logger or modify an existing one with **PUT** requests.

```
PUT /rest/v2/logging/loggers/{loggerName}?level={level}&appender={appender}&appender=
{appender}...
```

Data Grid sets the level of the logger identified by **{loggerName}** to **{level}**. Optionally, it is possible to set one or more appenders for the logger. If no appenders are specified, those specified in the root logger will be used.

If the operation successfully completes, the service returns **204 (No Content)**.

### 2.7.4. Removing a logger

Remove an existing logger with **DELETE** requests.

```
DELETE /rest/v2/logging/loggers/{loggerName}
```

Data Grid removes the logger identified by **{loggerName}**, effectively reverting to the use of the root logger configuration.

If operation processed successfully, the service returns a response code **204 (No Content)**.

## 2.8. USING SERVER TASKS

Retrieve, execute, and upload Data Grid server tasks.

### 2.8.1. Retrieving Server Tasks Information

View information about available server tasks with **GET** requests.

```
GET /rest/v2/tasks
```

Table 2.26. Request Parameters

Parameter	Required or Optional	Value
<b>type</b>	OPTIONAL	<b>user</b> : will exclude internal (admin) tasks from the results

Data Grid responds with a list of available tasks. The list includes the names of tasks, the engines that handle tasks, the named parameters for tasks, the execution modes of tasks, either **ONE\_NODE** or **ALL\_NODES**, and the allowed security role in **JSON** format, as in the following example:

```
[
  {
    "name": "SimpleTask",
    "type": "TaskEngine",
    "parameters": [
      "p1",
      "p2"
    ],
    "execution_mode": "ONE_NODE",
    "allowed_role": null
  },
  {
    "name": "RunOnAllNodesTask",
    "type": "TaskEngine",
    "parameters": [
      "p1"
    ],
    "execution_mode": "ALL_NODES",
    "allowed_role": null
  },
  {
    "name": "SecurityAwareTask",
    "type": "TaskEngine",
    "parameters": [],
    "execution_mode": "ONE_NODE",
    "allowed_role": "MyRole"
  }
]
```

## 2.8.2. Executing Tasks

Execute tasks with **POST** requests that include the task name and required parameters prefixed with **param**.

```
POST /rest/v2/tasks/SimpleTask?action=exec&param.p1=v1&param.p2=v2
```

Data Grid responds with the task result.

## 2.8.3. Uploading Script Tasks

Upload script tasks with **PUT** or **POST** requests.

Supply the script as the content payload of the request. After Data Grid uploads the script, you can execute it with **GET** requests.

```
POST /rest/v2/tasks/taskName
```

## 2.9. WORKING WITH DATA GRID SECURITY

View and modify security information.

### 2.9.1. Retrieving the ACL of a user

View information about the user's principals and access-control list.

```
GET /rest/v2/security/user/acl
```

Data Grid responds with information about the user who has performed the request. The list includes the principals of the user, and a list of resources and the permissions that user has when accessing them.

```
{
  "subject": [
    {
      "name": "deployer",
      "type": "NamePrincipal"
    }
  ],
  "global": [
    "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "CREATE", "MONITOR",
    "ALL_READ", "ALL_WRITE" ],
  "caches": {
    "__protobuf_metadata": [
      "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "CREATE", "MONITOR",
      "ALL_READ", "ALL_WRITE"
    ],
    "mycache": [
      "LIFECYCLE", "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "ADMIN",
      "CREATE", "MONITOR", "ALL_READ", "ALL_WRITE"
    ],
    "__script_cache": [
      "READ", "WRITE", "EXEC", "LISTEN", "BULK_READ", "BULK_WRITE", "CREATE", "MONITOR",
      "ALL_READ", "ALL_WRITE"
    ]
  }
}
```

```

]
}
}

```

### 2.9.2. Flushing the ACL cache

Flush the access-control list cache across the cluster.

```
POST /rest/v2/security/cache?action=flush
```

### 2.9.3. Retrieving the available roles

View all the available roles defined in the server.

```
GET /rest/v2/security/roles
```

Data Grid responds with a list of available roles. If authorization is enabled, only a user with the **ADMIN** permission can call this API.

```
["observer","application","admin","monitor","deployer"]
```

### 2.9.4. Retrieving the roles for a principal

View all the roles which map to a principal.

```
GET /rest/v2/security/roles/some_principal
```

Data Grid responds with a list of available roles for the specified principal. The principal need not exist in the realm in use.

```
["observer"]
```

### 2.9.5. Granting roles to a principal

Grant one or more new roles to a principal.

```
PUT /v2/security/roles/some_principal?action=grant&role=role1&role=role2
```

Table 2.27. Request Parameters

Parameter	Required or Optional	Value
<b>role</b>	REQUIRED	The name of a role

### 2.9.6. Denying roles to a principal

Remove one or more roles that were previously granted to a principal.

```
PUT /v2/security/roles/some_principal?action=deny&role=role1&role=role2
```



Table 2.28. Request Parameters

Parameter	Required or Optional	Value
<b>role</b>	REQUIRED	The name of a role