Red Hat Data Grid 8.1

Data Grid REST API

Configure and interact with the Data Grid REST API

Last Updated: 2021-06-09

Red Hat Data Grid 8.1 Data Grid REST API

Configure and interact with the Data Grid REST API

Legal Notice

Abstract

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

Table of Contents

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 8.1 Documentation
- Data Grid 8.1 Component Details
- Supported Configurations for Data Grid 8.1
- 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 Encoding section for more information about MediaTypes and encoding data with Data Grid.

The following example configures storage format for entries:

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

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.

For example, consider the following schema:

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

The corresponding JSON document is as follows:

```
{
   "_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.

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
Content-Type	REQUIRED	Sets the MediaType for the Data Grid configuration payload; either **application/xml** or **application/json**.
Flags	OPTIONAL	Used to set AdminFlags

References

- Data Grid XML Configuration
- Data Grid JSON Configuration

2.1.1.1. XML Configuration

Data Grid configuration in XML format must conform to the schema and include:

- **<infinispan>** root element.
- **<cache-container>** definition.

Example XML Configuration

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

2.1.1.2. JSON Configuration

Data Grid configuration in JSON format:

- Requires the cache definition only.
- Must follow the structure of an XML configuration.
  - XML elements become JSON objects.
  - XML attributes become JSON fields.

Example JSON Configuration

```
{
  "distributed-cache": {
    "name": "myCache",
    "mode": "SYNC",
    "encoding": {
      "media-type": "application/x-protostream"
      },
    "memory": {
      "max-count": 1000000,
      "when-full": "REMOVE"
    }
  }
}
```

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
Accept	OPTIONAL	Sets the required format to return content. Supported formats are **application/xml** and **application/json**. The default is **application/json**. See Accept 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 retreive 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 a value already exists for the specified key, the **POST** request returns an HTTP **CONFLICT** status 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
Key-Content-Type	OPTIONAL	Sets the content type for the key in the request. See Key-Content-Type for more information.
Content-Type	OPTIONAL	Sets the MediaType of the value for the key.
timeToLiveSeconds	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.
maxIdleTimeSeconds	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.
flags	OPTIONAL	The flags used to add the entry. See Flag 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}
```

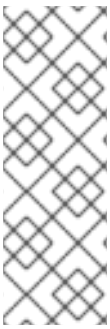
If a value already exists for the specified key, the **PUT** request updates the value. If you do not want to modify existing values, use **POST** requests that return HTTP **CONFLICT** status instead of modifying values. See Adding Values.

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
Key-Content-Type	OPTIONAL	Sets the content type for the key in the request. The default is **application/x-java-object; type=java.lang.String**. See Key-Content-Type for more information.
Accept	OPTIONAL	Sets the required format to return content. See Accept 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

Header	Required or Optional	Parameter
Key-Content-Type	OPTIONAL	Sets the content type for the key in the request. The default is **application/x-java-object; type=java.lang.String**. See Key-Content-Type 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
Key-Content-Type	OPTIONAL	Sets the content type for the key in the request. The default is **application/x-java-object; type=java.lang.String**. See Key-Content-Type 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
batch-size	OPTIONAL	Specifies the internal batch size when retrieving the keys. The default value is **1000**.

2.1.14. 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
```

2.1.15. 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
```

2.1.16. Getting Cache Statistics

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

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

2.1.17. 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 2.8. Request Parameters

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

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
}
```

2.1.18. 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 a response of **200** only after the re-indexing operation is complete.
- **async** returns a response of **200** immediately and the re-indexing operation continues running in the cluster. You can check the status with the Index Statistics REST call.

2.1.19. Purging Indexes

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

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

2.1.20. Retrieving Index Statistics

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.

2.1.21. Retrieving Query Statistics

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.

2.1.22. Clearing Query Statistics

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

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

2.1.23. Listing Caches

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

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

2.1.24. Cross-Site Operations with Caches

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

2.1.24.1. 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 2.9. Returned Status

Value	Description
online	All nodes in the local cluster have a cross-site view with the backup location.
offline	No nodes in the local cluster have a cross-site view with the backup location.
mixed	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.

2.1.24.2. 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 2.10. Returned Status

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

2.1.24.3. 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
```

2.1.24.4. 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
```

2.1.24.5. 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
```

2.1.24.6. 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
```

2.1.24.7. 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 2.11. Returned Status

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

2.1.24.8. 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
```

2.1.24.9. 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
```

2.1.24.10. 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
```

2.1.25. Rolling Upgrades

Perform rolling upgrades of cache data between Data Grid clusters

2.1.25.1. 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.

2.1.25.2. 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
```

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. Adding Values to Counters

Add values to specific counters with **POST** requests.

IMPORTANT

This method processes **plain/text** content only.

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

If the request payload is empty, the counter is incremented by one, otherwise the payload is interpreted as a signed long and added to the counter.

NOTE

WEAK counters never respond after operations.

STRONG counters return the current value after each operation.

2.2.5. Getting Counter Values

Retrieve counter values with **GET** requests.

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

Table 2.12. Headers

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

2.2.6. Resetting Counters

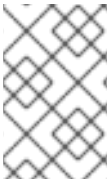
Restore the intial value of counters without **POST** requests and the **?action=reset** parameter.

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

2.2.7. 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.

STRONG counters return the current value after each operation.

2.2.8. 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

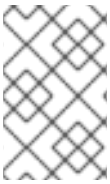
WEAK counters never respond after operations.

STRONG counters return the current value after each operation.

2.2.9. 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.

STRONG counters return the current value after each operation.

2.2.10. 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***.

2.2.11. 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.

2.2.12. 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 **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.

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}
```

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
     }
  ]

}
```


- **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

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 2.13. Headers

Header	Required or Optional	Parameter

Header	Required or Optional	Parameter
Accept	OPTIONAL	The required format to return the content. Supported formats are *application/json* and *application/xml*. 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. Cross-Site Operations with Cache Managers

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

2.4.10.1. 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 2.14. Returned Status

Value	Description
online	All nodes in the local cluster have a cross-site view with the backup location.
offline	No nodes in the local cluster have a cross-site view with the backup location.

Value	Description
mixed	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.

2.4.10.2. 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
```

2.4.10.3. 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
```

2.4.10.4. 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
```

2.4.10.5. 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
```

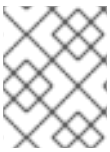
Data Grid responds with an array of the cache manager names configured for the server.

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 returns a service unavailable status (**503**) for REST client requests and a Server Error (**code 0x85**) for Hot Rod client requests.



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}
```

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 **200(OK)** 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 **200(OK)** 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 **200(OK)**.

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.

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.

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 2.15. Request Parameters

Parameter	Required or Optional	Value
type	OPTIONAL	**user**: 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 **GET** requests that include the task name and required parameters prefixed with **param**.

```
GET /rest/v2/tasks/myTask?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
```