Red Hat Data Grid 8.4

Data Grid REST API

Configure and interact with the Data Grid REST API

Last Updated: 2024-03-26

Red Hat Data Grid 8.4 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 8.4 Documentation
- Data Grid 8.4 Component Details
- Supported Configurations for Data Grid 8.4
- 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

- Creating Data Grid users
- 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>
  <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}
```

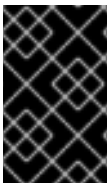
Table 2.1. Headers

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

2.1.1.1. Cache configuration

You can create declarative cache configuration in XML, JSON, and YAML format.

All declarative caches must conform to the Data Grid schema. Configuration in JSON format must follow the structure of an XML configuration, elements correspond to objects and attributes correspond to fields.



IMPORTANT

Data Grid restricts characters to a maximum of **255** for a cache name or a cache template name. If you exceed this character limit, Data Grid throws an exception. Write succinct cache names and cache template names.

IMPORTANT

A file system might set a limitation for the length of a file name, so ensure that a cache's name does not exceed this limitation. If a cache name exceeds a file system's naming limitation, general operations or initialing operations towards that cache might fail. Write succinct file names.

Distributed caches

XML

```
<distributed-cache owners="2"
                   segments="256"
                   capacity-factor="1.0"
                   l1-lifespan="5000"
                   mode="SYNC"
                   statistics="true">
  <encoding media-type="application/x-protostream"/>
  <locking isolation="REPEATABLE_READ"/>
  <transaction mode="FULL_XA"
               locking="OPTIMISTIC"/>
  <expiration lifespan="5000"
              max-idle="1000" />
  <memory max-count="1000000"
          when-full="REMOVE"/>
  <indexing enabled="true"
            storage="local-heap">
    <index-reader refresh-interval="1000"/>
    <indexed-entities>
      <indexed-entity>org.infinispan.Person</indexed-entity>
    </indexed-entities>
  </indexing>
  <partition-handling when-split="ALLOW_READ_WRITES"
                      merge-policy="PREFERRED_NON_NULL"/>
  <persistence passivation="false">
    <!-- Persistent storage configuration. -->
  </persistence>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "mode": "SYNC",
    "owners": "2",
    "segments": "256",
    "capacity-factor": "1.0",
    "l1-lifespan": "5000",
    "statistics": "true",
    "encoding": {
      "media-type": "application/x-protostream"
    },
    "locking": {
      "isolation": "REPEATABLE_READ"
```

```
    },
    "transaction": {
      "mode": "FULL_XA",
      "locking": "OPTIMISTIC"
    },
    "expiration" : {
      "lifespan" : "5000",
      "max-idle" : "1000"
    },
    "memory": {
      "max-count": "1000000",
      "when-full": "REMOVE"
    },
    "indexing" : {
          "enabled" : true,
          "storage" : "local-heap",
          "index-reader" : {
            "refresh-interval" : "1000"
          },
          "indexed-entities": [
            "org.infinispan.Person"
          ]
    },
    "partition-handling" : {
      "when-split" : "ALLOW_READ_WRITES",
      "merge-policy" : "PREFERRED_NON_NULL"
    },
    "persistence" : {
      "passivation" : false
    }
  }
}
```

YAML

```
distributedCache:
  mode: "SYNC"
  owners: "2"
  segments: "256"
  capacityFactor: "1.0"
  l1Lifespan: "5000"
  statistics: "true"
  encoding:
    mediaType: "application/x-protostream"
  locking:
    isolation: "REPEATABLE_READ"
  transaction:
    mode: "FULL_XA"
    locking: "OPTIMISTIC"
  expiration:
    lifespan: "5000"
    maxIdle: "1000"
  memory:
    maxCount: "1000000"
    whenFull: "REMOVE"
```

```
  indexing:
    enabled: "true"
    storage: "local-heap"
    indexReader:
      refreshInterval: "1000"
    indexedEntities:
      - "org.infinispan.Person"
  partitionHandling:
    whenSplit: "ALLOW_READ_WRITES"
    mergePolicy: "PREFERRED_NON_NULL"
  persistence:
    passivation: "false"
    # Persistent storage configuration.
```

Replicated caches

XML

```
<replicated-cache segments="256"
                  mode="SYNC"
                  statistics="true">
  <encoding media-type="application/x-protostream"/>
  <locking isolation="REPEATABLE_READ"/>
  <transaction mode="FULL_XA"
               locking="OPTIMISTIC"/>
  <expiration lifespan="5000"
              max-idle="1000" />
  <memory max-count="1000000"
          when-full="REMOVE"/>
  <indexing enabled="true"
            storage="local-heap">
    <index-reader refresh-interval="1000"/>
    <indexed-entities>
      <indexed-entity>org.infinispan.Person</indexed-entity>
    </indexed-entities>
  </indexing>
  <partition-handling when-split="ALLOW_READ_WRITES"
                      merge-policy="PREFERRED_NON_NULL"/>
  <persistence passivation="false">
    <!-- Persistent storage configuration. -->
  </persistence>
</replicated-cache>
```

JSON

```
{
  "replicated-cache": {
    "mode": "SYNC",
    "segments": "256",
    "statistics": "true",
    "encoding": {
      "media-type": "application/x-protostream"
    },
    "locking": {
```

```
      "isolation": "REPEATABLE_READ"
    },
    "transaction": {
      "mode": "FULL_XA",
      "locking": "OPTIMISTIC"
    },
    "expiration" : {
      "lifespan" : "5000",
      "max-idle" : "1000"
    },
    "memory": {
      "max-count": "1000000",
      "when-full": "REMOVE"
    },
    "indexing" : {
      "enabled" : true,
      "storage" : "local-heap",
      "index-reader" : {
        "refresh-interval" : "1000"
        },
      "indexed-entities": [
        "org.infinispan.Person"
      ]
    },
    "partition-handling" : {
      "when-split" : "ALLOW_READ_WRITES",
      "merge-policy" : "PREFERRED_NON_NULL"
    },
    "persistence" : {
      "passivation" : false
    }
  }
}
```

YAML

```
replicatedCache:
  mode: "SYNC"
  segments: "256"
  statistics: "true"
  encoding:
    mediaType: "application/x-protostream"
  locking:
    isolation: "REPEATABLE_READ"
  transaction:
    mode: "FULL_XA"
    locking: "OPTIMISTIC"
  expiration:
    lifespan: "5000"
    maxIdle: "1000"
  memory:
    maxCount: "1000000"
    whenFull: "REMOVE"
  indexing:
    enabled: "true"
```

```
    storage: "local-heap"
    indexReader:
      refreshInterval: "1000"
    indexedEntities:
      - "org.infinispan.Person"
  partitionHandling:
    whenSplit: "ALLOW_READ_WRITES"
    mergePolicy: "PREFERRED_NON_NULL"
  persistence:
    passivation: "false"
    # Persistent storage configuration.
```

Multiple caches

XML

```
<infinispan
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:infinispan:config:14.0 https://infinispan.org/schemas/infinispan-config-
14.0.xsd
                          urn:infinispan:server:14.0 https://infinispan.org/schemas/infinispan-server-14.0.xsd"
      xmlns="urn:infinispan:config:14.0"
      xmlns:server="urn:infinispan:server:14.0">
  <cache-container name="default"
                   statistics="true">
    <distributed-cache name="mycacheone"
                       mode="ASYNC"
                       statistics="true">
      <encoding media-type="application/x-protostream"/>
      <expiration lifespan="300000"/>
      <memory max-size="400MB"
              when-full="REMOVE"/>
    </distributed-cache>
    <distributed-cache name="mycachetwo"
                       mode="SYNC"
                       statistics="true">
      <encoding media-type="application/x-protostream"/>
      <expiration lifespan="300000"/>
      <memory max-size="400MB"
              when-full="REMOVE"/>
    </distributed-cache>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "name" : "default",
      "statistics" : "true",
      "caches" : {
        "mycacheone" : {
          "distributed-cache" : {
```

```
            "mode": "ASYNC",
            "statistics": "true",
            "encoding": {
              "media-type": "application/x-protostream"
            },
            "expiration" : {
              "lifespan" : "300000"
            },
            "memory": {
              "max-size": "400MB",
              "when-full": "REMOVE"
            }
          }
        },
        "mycachetwo" : {
          "distributed-cache" : {
            "mode": "SYNC",
            "statistics": "true",
            "encoding": {
              "media-type": "application/x-protostream"
            },
            "expiration" : {
              "lifespan" : "300000"
            },
            "memory": {
              "max-size": "400MB",
              "when-full": "REMOVE"
            }
          }
        }
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    name: "default"
    statistics: "true"
    caches:
      mycacheone:
       distributedCache:
          mode: "ASYNC"
          statistics: "true"
          encoding:
            mediaType: "application/x-protostream"
          expiration:
            lifespan: "300000"
          memory:
            maxSize: "400MB"
            whenFull: "REMOVE"
      mycachetwo:
        distributedCache:
```

```
    mode: "SYNC"
    statistics: "true"
    encoding:
      mediaType: "application/x-protostream"
    expiration:
      lifespan: "300000"
    memory:
      maxSize: "400MB"
      whenFull: "REMOVE"
```

Additional resources

- Data Grid configuration schema reference
- infinispan-config-14.0.xsd

2.1.2. Modifying Caches

Make changes to attributes in cache configurations across Data Grid clusters with **PUT** requests that include XML or JSON configuration in the payload.

NOTE

You can modify a cache only if the changes are compatible with the existing configuration.

For example you cannot use a replicated cache configuration to modify a distributed cache. Likewise if you create a cache configuration with a specific attribute, you cannot modify the configuration to use a different attribute instead. For example, attempting to modify cache configuration by specifying a value for the **max-count** attribute results in invalid configuration if the **max-size** is already set.

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

Table 2.2. 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

2.1.3. Verifying Caches

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

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

2.1.4. 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.

2.1.5. Retrieving Cache Configuration

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

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

Table 2.3. 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.

2.1.6. Converting Cache Configurations between XML, JSON and YAML

Invoke a **POST** request with valid configuration and the **?action=convert** parameter. Data Grid responds with the equivalent representation of the configuration in the type specified by the **Accept** header.

```
POST /rest/v2/caches?action=convert
```

To convert cache configuration you must specify the input format for the configuration with the **Content-Type** header and the desired output format with the **Accept** header. For example, the following command converts the replicated cache configuration from XML to YAML:

```
curl localhost:11222/rest/v2/caches?action=convert \
--digest -u username:password \
-X POST -H "Accept: application/yaml" -H "Content-Type: application/xml" \
-d '<replicated-cache mode="SYNC" statistics="false"><encoding media-type="application/x-protostream"/><expiration lifespan="300000" /><memory max-size="400MB" when-full="REMOVE"/></replicated-cache>'
```

2.1.7. Comparing Cache Configurations

Invoke a **POST** request with a **multipart/form-data** body containing two cache configurations and the **?action=compare** parameter.

```
POST /rest/v2/caches?action=compare
```

TIP

Add the **ignoreMutable=true** parameter to ignore mutable attributes in the comparison.

Data Grid responds with **204 (No Content)** in case the configurations are equal, and **409 (Conflict)** in case they are different.

2.1.8. Retrieving All Cache Details

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

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

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,
 "rebalancing_enabled": true,
 "bounded": false,
```

```
  "indexed": false,
  "persistent": false,
  "transactional": false,
  "secured": false,
  "has_remote_backup": false,
  "indexing_in_progress": false,
  "statistics": false,
  "mode" : "DIST_SYNC",
  "storage_type": "HEAP",
  "max_size": "",
  "max_size_bytes" : -1
}
```

- **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.
- **rebalancing_enabled** is true if rebalancing is enabled. Fetching this property might fail on the server. In that case the property won't be present in the payload.
- **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.
- **key_storage** the media type of the cache keys.
- **value_storage** the media type of the cache values.

NOTE

key_storage and **value_storage** matches encoding configuration of the cache. For server caches with no encoding, Data Grid assumes **application/x-protostream** when a cache is indexed and **application/unknown** otherwise.

2.1.9. Retrieving Data Distribution of a Cache

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

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

Data Grid provides a JSON response such as the following:

```
[
  {
    "node_name": "NodeA",
    "node_addresses": [
      "127.0.0.1:44175"
    ],
    "memory_entries": 0,
    "total_entries": 0,
    "memory_used": 528512
  },
  {
    "node_name":"NodeB",
    "node_addresses": [
      "127.0.0.1:44187"
    ],
    "memory_entries": 0,
    "total_entries": 0,
    "memory_used": 528512
  }
]
```


Each element in the list represents a node. The properties are:

- **node_name** is the node name
- **node_addresses** is a list with all the node's physical addresses.
- **memory_entries** the number of entries the node holds in memory belonging to the cache.
- **total_entries** the number of entries the node has in memory and disk belonging to the cache.
- **memory_used** the value in bytes the eviction algorithm estimates the cache occupies. Returns -1 if eviction is not enabled.

2.1.10. Retrieving all mutable cache configuration attributes

Invoke a **GET** request to retrieve all mutable cache configuration attributes for Data Grid caches.

```
GET /rest/v2/caches/{name}?action=get-mutable-attributes
```

Data Grid provides a JSON response such as the following:


```
[
  "jmx-statistics.statistics",
  "locking.acquire-timeout",
  "transaction.single-phase-auto-commit",
  "expiration.max-idle",
  "transaction.stop-timeout",
  "clustering.remote-timeout",
  "expiration.lifespan",
  "expiration.interval",
  "memory.max-count",
  "memory.max-size"
]
```

Add the **full** parameter to obtain values and type information:

```
GET /rest/v2/caches/mycache?action=get-mutable-attributes&full=true
```

Data Grid provides a JSON response such as the following:


```
{
  "jmx-statistics.statistics": {
    "value": true,
    "type": "boolean"
  },
  "locking.acquire-timeout": {
    "value": 15000,
    "type": "long"
  },
  "transaction.single-phase-auto-commit": {
    "value": false,
    "type": "boolean"
  },
  "expiration.max-idle": {
    "value": -1,
    "type": "long"
  },
  "transaction.stop-timeout": {
    "value": 30000,
    "type": "long"
  },
  "clustering.remote-timeout": {
    "value": 17500,
    "type": "long"
  },
  "expiration.lifespan": {
    "value": -1,
    "type": "long"
  },
  "expiration.interval": {
    "value": 60000,
    "type": "long"
  },
  "memory.max-count": {
    "value": -1,
    "type": "long"
  },
  "memory.max-size": {
    "value": null,
    "type": "string"
  }
}
```


For attributes of type **enum**, an additional **universe** property will contain the set of possible values.

2.1.11. Updating cache configuration attributes

Invoke a **POST** request to change a mutable cache configuration attribute.

```
POST /rest/v2/caches/{name}?action=set-mutable-attributes&attribute-name=
{attributeName}&attribute-value={attributeValue}
```

2.1.12. Adding Entries

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

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

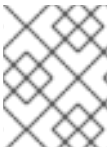
The preceding request places the payload, or request body, in the **cacheName** cache with the **cacheKey** key. The request replaces any data that already exists and updates the **Time-To-Live** and **Last-Modified** values, if they apply.

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.

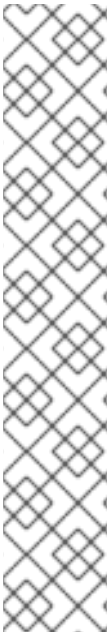
Table 2.4. Headers

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.

2.1.13. 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.

2.1.14. 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 2.5. 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.

2.1.15. 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 2.6. 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.

2.1.16. Deleting Entries

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

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

Table 2.7. 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.

2.1.17. Deleting Caches

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

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

2.1.18. 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 2.8. Request Parameters

Parameter	Required or Optional	Value
limit	OPTIONAL	Specifies the maximum number of keys to retrieve using an InputStream. A negative value retrieves all keys. The default value is **-1**.

Parameter	Required or Optional	Value
batch	OPTIONAL	Specifies the internal batch size when retrieving the keys. The default value is **1000**.

2.1.19. 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 2.9. Request Parameters

Parameter	Required or Optional	Value
metadata	OPTIONAL	Includes metadata for each entry in the response. The default value is **false**.
limit	OPTIONAL	Specifies the maximum number of keys to include in the response. A negative value retrieves all keys. The default value is **-1**.
batch	OPTIONAL	Specifies the internal batch size when retrieving the keys. The default value is **1000**.
content-negotiation	OPTIONAL	If **true**, will convert keys and values to a readable format. For caches with text encodings (e.g., text/plain, xml, json), the server returns keys and values as plain text. For caches with binary encodings, the server will return the entries as JSON if the conversion is supported, otherwise in a text hexadecimal format, e.g., **0xA123CF98**. When content-negotiation is used, the response will contain two headers: **key-content-type** and **value-content-type** to described the negotiated format.

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".

2.1.20. 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)**.

2.1.21. 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
```

2.1.22. Getting Cache Statistics

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

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

2.1.23. Listing Caches

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

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

2.1.24. Listening to cache events

Receive cache events using Server-Sent Events. The **event** value will be one of **cache-entry-created**, **cache-entry-removed**, **cache-entry-updated**, **cache-entry-expired**. The **data** value will contain the key of the entry that has fired the event in the format set by the **Accept** header.

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

Table 2.10. Headers

Header	Required or Optional	Parameter
Accept	OPTIONAL	Sets the required format to return content. Supported formats are **text/plain** and **application/json**. The default is **application/json**. See Accept for more information.

2.1.25. Enabling rebalancing

Turn on automatic rebalancing for a specific cache.

```
POST /rest/v2/caches/{cacheName}?action=enable-rebalancing
```

2.1.26. Disabling rebalancing

Turn off automatic rebalancing for a specific cache.

```
POST /rest/v2/caches/{cacheName}?action=disable-rebalancing
```

2.1.27. Getting Cache Availability

Retrieve the availability of a cache.

```
GET /rest/v2/caches/{cacheName}?action=get-availability
```

NOTE

You can get the availability of internal caches but this is subject to change in future Data Grid versions.

2.1.28. Setting Cache Availability

Change the availability of clustered caches when using either the DENY_READ_WRITES or ALLOW_READS partition handling strategy.

```
POST /rest/v2/caches/{cacheName}?action=set-availability&availability={AVAILABILITY}
```

Table 2.11. Request Parameters

Parameter	Required or Optional	Value
availability	REQUIRED	AVAILABLE or DEGRADED_MODE

- **AVAILABLE** makes caches available to all nodes in a network partition.
- **DEGRADED_MODE** prevents read and write operations on caches when network partitions occur.

NOTE

You can set the availability of internal caches but this is subject to change in future Data Grid versions.

2.1.29. Set a Stable Topology

By default, after a cluster shutdown, Data Grid waits for all nodes to join the cluster and restore the topology. However, it is possible to define the current cluster topology as stable for a specific cache using the REST operation.

```
POST /rest/v2/caches/{cacheName}?action=initialize&force={FORCE}
```

Table 2.12. Request Parameters

Parameter	Required or Optional	Value
force	OPTIONAL	true or false.

- **force** is required when the number of missing nodes in the current topology is greater or equal to the number of owners.

IMPORTANT

Manually installing a topology can lead to data loss, only perform this operation if the initial topology cannot be recreated.

2.1.30. Indexing and Querying with the REST API

Query remote caches with **GET** requests and the **?action=search&query** parameter from any HTTP client.

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

Data Grid response

```
{
  "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 2.13. Request Parameters

Parameter	Required or Optional	Value
query	REQUIRED	Specifies the query string.
offset	OPTIONAL	Specifies the index of the first result to return. The default is **0**.
max_results	OPTIONAL	Sets the number of results to return. The default is **10**.

Parameter	Required or Optional	Value
hit_count_accuracy	OPTIONAL	Limits the required accuracy of the hit count for the indexed queries to an upper-bound. The default is **10000**. You can change the default limit by setting the **query.hit-count-accuracy** cache property.
local	OPTIONAL	When **true**, the query is restricted to the data present in node that process the request. The default is **false**.

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

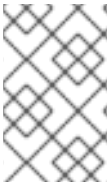
```
POST /rest/v2/caches/{cacheName}?action=search
```

Query in request body

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

2.1.30.1. Rebuilding indexes

When you delete fields or change index field definitions, you must rebuild the index to ensure the index is consistent with data in the cache.



NOTE

Rebuilding Protobuf schema using REST, CLI, Data Grid Console or remote client might lead to inconsistencies. Remote clients might have different versions of the Protostream entity and this might lead to unreliable behavior.

Reindex all data in caches with **POST** requests and the **?action=reindex** parameter.

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

Table 2.14. Request Parameters

Parameter	Required or Optional	Value

Parameter	Required or Optional	Value
mode	OPTIONAL	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.
local	OPTIONAL	When **true**, only the data from node that process the request is re-indexed. The default is **false**, meaning all data cluster-wide is re-indexed.

2.1.30.2. Updating index schema

The update index schema operation lets you add schema changes with a minimal downtime. Instead of removing previously indexed data and recreating the index schema, Data Grid adds new fields to the existing schema.

Update the index schema of values in your cache using **POST** requests and the **?action=updateSchema** parameter.

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

2.1.30.3. Purging indexes

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

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

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

2.1.30.4. Get Indexes Metamodel

Present the full index schema metamodel of all indexes defined on this cache.

```
GET /rest/v2/caches/{cacheName}/search/indexes/metamodel
```

Data Grid response

```
[{
   "entity-name": "org.infinispan.query.test.Book",
   "java-class": "org.infinispan.query.test.Book",
```

```
"index-name": "org.infinispan.query.test.Book",
"value-fields": {
   "description": {
      "multi-valued": false,
      "multi-valued-in-root": false,
      "type": "java.lang.String",
      "projection-type": "java.lang.String",
      "argument-type": "java.lang.String",
      "searchable": true,
      "sortable": false,
      "projectable": false,
      "aggregable": false,
      "analyzer": "standard"
   },
   "name": {
      "multi-valued": false,
      "multi-valued-in-root": true,
      "type": "java.lang.String",
      "projection-type": "java.lang.String",
      "argument-type": "java.lang.String",
      "searchable": true,
      "sortable": false,
      "projectable": false,
      "aggregable": false,
      "analyzer": "standard"
   },
   "surname": {
      "multi-valued": false,
      "multi-valued-in-root": true,
      "type": "java.lang.String",
      "projection-type": "java.lang.String",
      "argument-type": "java.lang.String",
      "searchable": true,
      "sortable": false,
      "projectable": false,
      "aggregable": false
   },
   "title": {
      "multi-valued": false,
      "multi-valued-in-root": false,
      "type": "java.lang.String",
      "projection-type": "java.lang.String",
      "argument-type": "java.lang.String",
      "searchable": true,
      "sortable": false,
      "projectable": false,
      "aggregable": false
   }
},
"object-fields": {
   "authors": {
      "multi-valued": true,
      "multi-valued-in-root": true,
      "nested": true,
      "value-fields": {
         "name": {
```

```
                "multi-valued": false,
                "multi-valued-in-root": true,
                "type": "java.lang.String",
                "projection-type": "java.lang.String",
                "argument-type": "java.lang.String",
                "searchable": true,
                "sortable": false,
                "projectable": false,
                "aggregable": false,
                "analyzer": "standard"
            },
            "surname": {
                "multi-valued": false,
                "multi-valued-in-root": true,
                "type": "java.lang.String",
                "projection-type": "java.lang.String",
                "argument-type": "java.lang.String",
                "searchable": true,
                "sortable": false,
                "projectable": false,
                "aggregable": false
            }
        }
    }
}, {
    "entity-name": "org.infinispan.query.test.Author",
    "java-class": "org.infinispan.query.test.Author",
    "index-name": "org.infinispan.query.test.Author",
    "value-fields": {
        "surname": {
            "multi-valued": false,
            "multi-valued-in-root": false,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false
        },
        "name": {
            "multi-valued": false,
            "multi-valued-in-root": false,
            "type": "java.lang.String",
            "projection-type": "java.lang.String",
            "argument-type": "java.lang.String",
            "searchable": true,
            "sortable": false,
            "projectable": false,
            "aggregable": false,
            "analyzer": "standard"
        }
    }
}]
```

2.1.30.5. Retrieving Query and Index Statistics

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

NOTE

You must enable statistics in the cache configuration or results are empty.

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

Table 2.15. Request Parameters

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

Data Grid response

```
{
  "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

Time is always measured 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.

2.1.30.6. Clearing Query Statistics

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

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

Data Grid resets only query execution times for the local node only. This operation does not clear index statistics.

2.1.30.7. Retrieving Index Statistics (Deprecated)

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

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

Data Grid response

```
{
   "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.

2.1.30.8. Retrieving Query Statistics (Deprecated)

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

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

Data Grid response

```
{
   "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.

2.1.30.9. Clearing Query Statistics (Deprecated)

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

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

2.1.31. Cross-Site Operations with Caches

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

2.1.31.1. Getting status of all backup locations

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

```
GET /rest/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": {
    "status": "online"
  },
  "LON": {
    "status": "mixed",
    "online": [
      "NodeA"
    ],
    "offline": [
      "NodeB"
    ]
  }
}
```

Table 2.16. Returned Status

Value	Description

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.

2.1.31.2. Getting status of specific backup locations

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

```
GET /rest/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 2.17. 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.

2.1.31.3. Taking backup locations offline

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

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

2.1.31.4. Bringing backup locations online

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

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

2.1.31.5. Pushing state to backup locations

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

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

2.1.31.6. Canceling state transfer

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

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

2.1.31.7. Getting state transfer status

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

```
GET /rest/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 2.18. 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.

2.1.31.8. Clearing state transfer status

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

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

2.1.31.9. 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 /rest/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 /rest/v2/caches/{cacheName}/x-site/backups/{siteName}/take-offline-config
```

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

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

2.1.32. Rolling Upgrades

Perform rolling upgrades of cache data between Data Grid clusters

2.1.32.1. Connecting Source Clusters

Connect a target cluster to the source cluster with:

```
POST /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

You must provide a **remote-store** definition in JSON format as the body:

JSON

```
{
  "remote-store": {
    "cache": "my-cache",
    "shared": true,
    "raw-values": true,
    "socket-timeout": 60000,
    "protocol-version": "2.9",
    "remote-server": [
      {
```

```
        "host": "127.0.0.2",
        "port": 12222
      }
    ],
    "connection-pool": {
      "max-active": 110,
      "exhausted-action": "CREATE_NEW"
    },
    "async-executor": {
      "properties": {
        "name": 4
      }
    },
    "security": {
      "authentication": {
        "server-name": "servername",
        "digest": {
          "username": "username",
          "password": "password",
          "realm": "realm",
          "sasl-mechanism": "DIGEST-MD5"
        }
      },
      "encryption": {
        "protocol": "TLSv1.2",
        "sni-hostname": "snihostname",
        "keystore": {
          "filename": "/path/to/keystore_client.jks",
          "password": "secret",
          "certificate-password": "secret",
          "key-alias": "hotrod",
          "type": "JKS"
        },
        "truststore": {
          "filename": "/path/to/gca.jks",
          "password": "secret",
          "type": "JKS"
        }
      }
    }
  }
}
```

Several elements are optional such as **security**, **async-executor** and **connection-pool**. The configuration must contain minimally the cache name, the **raw-values** set to **false** and the host/ip of the single port in the source cluster. For details about the **remote-store** configuration, consult the XSD Schema.

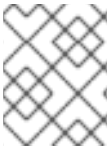
If the operation successfully completes, the service returns 204 (No Content). If the target cluster is already connected to the source cluster, it returns status 304 (Not Modified).

2.1.32.2. Obtaining Source Cluster connection details

To obtain the **remote-store** definition of a cache, use a **GET** request:

```
GET /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

If the cache was previously connected, it returns the configuration of the associated **remote-store** in JSON format and status 200 (OK), otherwise a 404 (Not Found) status.



NOTE

This is not a cluster wide operation, and it only returns the remote-store of the cache in the node where the REST invocation is handled.

2.1.32.3. Checking if a Cache is connected

To check if a cache have been connected to a remote cluster, use a **HEAD** request:

```
HEAD /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

Returns status 200 (OK) if for all nodes of the cluster, **cacheName** has a single remote store configured, and 404 (NOT_FOUND) otherwise.

2.1.32.4. Synchronizing Data

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

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

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

2.1.32.5. Disconnecting Source Clusters

After you synchronize data to target clusters, disconnect from the source cluster with **DELETE** requests:

```
DELETE /rest/v2/caches/{cacheName}/rolling-upgrade/source-connection
```

If the operation successfully completes, the service returns **204 (No Content)**. It no source was connected, it returns code 304 (Not Modified).

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 2.19. 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.

2.2.5. Resetting Counters

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

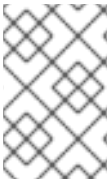
```
POST /rest/v2/counters/{counterName}?action=reset
```

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

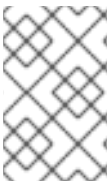
WEAK counters never respond after operations and return **204 (No Content)**.

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.

IMPORTANT

When you make changes to the existing Protobuf schema definition, you must either update or rebuild the index schema. If the changes involve modifying the existing fields, then you must rebuild the index. When you add new fields without touching existing schema, you can update the index schema instead of rebuilding it.

```
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.

2.3.6. Listing Protobuf Types

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

```
GET /rest/v2/schemas?action=types
```

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

```
["org.infinispan.Person", "org.infinispan.Phone"]
```

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:

NOTE

Information about caches with security authorization is available only to users with the specific roles and permissions assigned to them.

```
{
   "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",
  "relay_node": true,
  "relay_nodes_address": [
    "CacheManagerResourceTest-NodeA-49696"
  ],
  "sites_view": [
     "LON",
     "NYC"
  ],
  "rebalancing_enabled": 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 that are accessible to the current user
- **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".
- **relay_node** is true if the node handles RELAY messages between clusters.
- **relay_nodes_address** is an array of logical addresses for relay nodes.
- **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.
- **rebalancing_enabled** is true if rebalancing is enabled. Fetching this property might fail on the server. In that case the property won't be present in the payload.

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 2.20. Headers

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.

Table 2.21. Request parameters

Parameter	Required or Optional	Description
pretty	OPTIONAL	If **true** returns formatted content, including additional spacing and line separators which improve readability but increase payload size. The default is **false**.

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"
        }
      }
    }
  }
]
```

Table 2.22. Request parameters

Parameter	Required or Optional	Description
pretty	OPTIONAL	If **true** returns formatted content, including additional spacing and line separators which improve readability but increase payload size. The default is **false**.

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.

Table 2.23. Request parameters

Parameter	Required or Optional	Description
pretty	OPTIONAL	If **true** returns formatted content, including additional spacing and line separators which improve readability but increase payload size. The default is **false**.

2.4.8. 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",
  "rebalancing_enabled": true
}, {
  "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",
  "rebalancing_enabled": false
}]
```

Table 2.24. Request parameters

Parameter	Required or Optional	Description
pretty	OPTIONAL	If **true** returns formatted content, including additional spacing and line separators which improve readability but increase payload size. The default is **false**.

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. Shutdown all container caches

Shut down the Data Grid container on the server with **POST** requests.

```
POST /rest/v2/container?action=shutdown
```

Data Grid responds with **204 (No Content)** and then shutdowns all caches in the container. The servers remain running with active endpoints and clustering, however REST calls to container resources will result in a 503 Service Unavailable response.

NOTE

This method is primarily intended for use by the Data Grid Operator. The expectation is that the Server processes will be manually terminated shortly after this endpoint is invoked. Once this method has been called, it's not possible to restart the container state.

2.4.11. Enabling rebalancing for all caches

Turn on automatic rebalancing for all caches.

```
POST /rest/v2/cache-managers/{cacheManagerName}?action=enable-rebalancing
```

2.4.12. Disabling rebalancing for all caches

Turn off automatic rebalancing for all caches.

```
POST /rest/v2/cache-managers/{cacheManagerName}?action=disable-rebalancing
```

2.4.13. 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 2.25. JSON Parameters

Key	Required or Optional	Value
directory	OPTIONAL	Specifies a location on the server to create and store the backup archive.
resources	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 *Resource Parameters* table for more information.

Table 2.26. Resource Parameters

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

Key	Required or Optional	Value
process	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": "/path/accessible/to/the/server",
  "resources": {
    "caches": ["cache1", "cache2"],
    "counters": ["*"]
  }
}
```

2.4.14. 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"]
```

2.4.15. 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.

2.4.16. 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.

2.4.17. Deleting Backup Archives

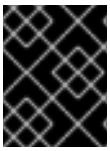
Remove backup archives from the server.

```
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.

2.4.18. 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.

2.4.18.1. 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 2.27. JSON Parameters

Key	Required or Optional	Value
location	REQUIRED	Specifies the path of the backup archive to restore.
resources	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 *Resource Parameters* table for more information.

Table 2.28. Resource Parameters

Key	Required or Optional	Value
caches	OPTIONAL	Specifies either an array of cache names to back up or * for all caches.

Key	Required or Optional	Value
cache-configs	OPTIONAL	Specifies either an array of cache templates to back up or * for all templates.
counters	OPTIONAL	Defines either an array of counter names to back up or * for all counters.
proto-schemas	OPTIONAL	Defines either an array of Protobuf schema names to back up or * for all schemas.
process	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": "/path/accessible/to/the/server/backup-to-restore.zip",
  "resources": {
    "counters": ["*"]
  }
}
```

2.4.18.2. 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 2.29. Form Data

Parameter	Content-Type	Required or Optional	Value
backup	**application/zip**	REQUIRED	Specifies the bytes of the backup archive to restore.
resources	**application/json**, **text/plain**	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--
```

2.4.19. 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"]
```

2.4.20. 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.

2.4.21. 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.

2.4.22. Listening to container configuration events

Receive events about configuration changes using Server-Sent Events. The **event** value will be one of **create-cache**, **remove-cache**, **update-cache**, **create-template**, **remove-template** or **update-template**. The **data** value will contain the declarative configuration of the entity that has been created. Remove events will only contain the name of the removed entity.

```
GET /rest/v2/container/config?action=listen
```

Table 2.30. Headers

Header	Required or Optional	Parameter
Accept	OPTIONAL	Sets the required format to return content. Supported formats are **application/yaml**, **application/json** and **application/xml**. The default is **application/yaml**. See Accept for more information.

Table 2.31. Request parameters

Parameter	Required or Optional	Description
includeCurrentState	OPTIONAL	If **true**, the results include the state of the existing configuration in addition to the changes. If set to **false**, the request returns only the changes. The default value is **false**.
pretty	OPTIONAL	If **true** returns formatted content, including additional spacing and line separators which improve readability but increase payload size. The default is **false**.

2.4.23. Listening to container events

Receive events from the container using Server-Sent Events. The emitted events come from logged information, so each event contains an identifier associated with the message. The **event** value will be **lifecycle-event**. The **data** has the logged information, which includes the **message**, **category**, **level**, **timestamp**, **owner**, **context**, and **scope**, some of which may be empty. Currently, we expose only **LIFECYCLE** events.

```
GET /rest/v2/container?action=listen
```

Table 2.32. Headers

Header	Required or Optional	Parameter
Accept	OPTIONAL	Sets the required format to return content. Supported formats are **application/yaml**, **application/json** and **application/xml**. The default is **application/yaml**. See Accept for more information.

Table 2.33. Request parameters

Parameter	Required or Optional	Description
includeCurrentState	OPTIONAL	If **true**, the results include the state of the existing configuration in addition to the changes. If set to **false**, the request returns only the changes. The default value is **false**.
pretty	OPTIONAL	If **true** returns formatted content, including additional spacing and line separators which improve readability but increase payload size. The default is **false**.

2.4.24. Cross-Site Operations with Cache Managers

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

2.4.24.1. 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"
    ],
    "mixed": [
      "CACHE_3"
    ]
  }
}
```

Table 2.34. Returned status

Value	Description
online	All nodes in the local cluster have a cross-site view with the backup location.

Value	Description
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.

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

Returns the status for a single backup location.

2.4.24.2. 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
```

2.4.24.3. 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
```

2.4.24.4. 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
```

2.4.24.5. 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}
```

2.4.24.6. 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
```

2.4.24.7. 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

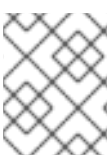
Data Grid currently supports one Cache Manager per server only.

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 /rest/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 /rest/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 /rest/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 Heap Dumps

Generate JVM heap dumps for Data Grid Servers with **POST** requests.

```
POST /rest/v2/server/memory?action=heap-dump[&live=true|false]
```

Data Grid generates a heap dump file in HPROF format in the server data directory and responds with the full path of the file in JSON format.

2.5.10. 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.

2.5.11. 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.

2.5.12. 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 2.35. JSON Parameters

Key	Required or Optional	Value
directory	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 2.36. JSON Parameters

Key	Required or Optional	Value
location	REQUIRED	Specifies the path of the backup archive to restore.
resources	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 *Resource Parameters* table for more information.

Table 2.37. Resource Parameters

Key	Required or Optional	Value
caches	OPTIONAL	Specifies either an array of cache names to back up or ***** for all caches.
cache-configs	OPTIONAL	Specifies either an array of cache templates to back up or ***** for all templates.
counters	OPTIONAL	Defines either an array of counter names to back up or ***** for all counters.
proto-schemas	OPTIONAL	Defines either an array of Protobuf schema names to back up or ***** for all schemas.
process	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": "/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 2.38. Form Data

Parameter	Content-Type	Required or Optional	Value
backup	**application/zip**	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.

2.6.12. Checking Cluster Distribution

Retrieve the distribution details about all servers in the Data Grid cluster.

```
GET /rest/v2/cluster?action=distribution
```

Returns a JSON array of each Data Grid server statistics in the cluster with the format:

```
[
  {
    "node_name": "NodeA",
    "node_addresses": [
      "127.0.0.1:39313"
    ],
    "memory_available": 466180016,
    "memory_used": 56010832
  },
  {
    "node_name": "NodeB",
    "node_addresses": [
      "127.0.0.1:47477"
    ],
    "memory_available": 467548568,
    "memory_used": 54642280
  }
]
```

Each element in the array represents an Data Grid node. If the statistics collection is disabled, information about memory usage values is -1. The properties are:

- **node_name** is the node name.
- **node_addresses** is a list with all the node's physical addresses.
- **memory_available** the node available memory in bytes.
- **memory_used** the node used memory in bytes.

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 2.39. 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 **POST** requests that include the task name, an optional cache name and required parameters prefixed with **param**.

```
POST /rest/v2/tasks/SimpleTask?action=exec&cache=mycache&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
```

2.8.4. Downloading Script Tasks

Download script tasks with **GET** requests.

```
GET /rest/v2/tasks/taskName?action=script
```

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

Table 2.40. Request Parameters

Parameter	Required or Optional	Value
role	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 /rest/v2/security/roles/some_principal?action=deny&role=role1&role=role2
```

Table 2.41. Request Parameters

Parameter	Required or Optional	Value
role	REQUIRED	The name of a role

2.10. ENABLING TRACING PROPAGATION

Tracing with Data Grid Server and REST API lets you monitor and analyze the flow of requests and track the execution path across different components.

2.10.1. Enabling tracing propagation between Data Grid Server and REST API

When you enable tracing propagation between the Data Grid Server and REST API, you must configure tracing on both the client side and the server side.

To propagate the OpenTelemetry tracing spans to the Data Grid spans, you must set the trace context on each REST invocation.

Prerequisite

- Have tracing enabled on Data Grid Server and remote client side.

Procedure

1. Extract the current tracing context using the **io.opentelemetry.api.trace.propagation.W3CTraceContextPropagator**.
   The extraction produces a context map that stores trace context information.
2. Pass the context map in the header of the REST call to ensure that the trace context is preserved.

```
HashMap<String, String> contextMap = new HashMap<>();
```

```
// Inject the request with the *current* Context, which contains our current Span.
W3CTraceContextPropagator.getInstance().inject(Context.current(), contextMap,
(carrier, key, value) -> carrier.put(key, value));

// Pass the context map in the header
RestCacheClient client = restClient.cache(CACHE_NAME);
client.put("aaa",
MediaType.TEXT_PLAIN.toString(),RestEntity.create(MediaType.TEXT_PLAIN, "bbb"),
contextMap);
```

The tracing spans that the client application generates are correlated with the dependent spans generated by the Data Grid Server.

Additional resources

- Enabling Data Grid tracing
- Hot Rod client tracing propagation