Using the Infinispan REST Server

Table of Contents

Chapter 1. Infinispan REST Endpoint

Infinispan 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 Infinispan 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.

When running the Docker image, configure authentication with the `APP_USER` and `APP_PASS` command line arguments.

Reference

- Adding Users to Property Realms
- Configuring Endpoint Authentication Mechanisms

1.2. Supported Protocols

The Infinispan 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.

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

Infinispan 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 Infinispan.

The following example configures the storage format for entries:

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

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

1.3.1. Supported Formats

You can write and read data in different formats and Infinispan 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`

Infinispan 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. Infinispan 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 Infinispan 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 Infinispan to first return content in JSON format (higher priority 0.8). If it is not possible to convert the storage format to JSON, Infinispan attempts the next format of `text/plain` (second highest priority 0.7). Finally, Infinispan 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. Infinispan 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 Infinispan 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>
```

Infinispan 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 Infinispan 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 Infinispan REST API

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

By default Infinispan 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 Infinispan caches and perform operations on data.

2.1.1. Creating Caches

Create named caches across Infinispan clusters with `POST` requests that include XML or JSON configuration in the payload.

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

Table 1. Headers

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

Cache Configuration

You can provide cache configuration in XML or JSON format.

XML

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

JSON

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

JSON format

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

2.1.2. Verifying Caches

Check if caches are available in Infinispan clusters with `HEAD` requests.

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

2.1.3. Creating Caches with Templates

Create caches from Infinispan templates with `POST` requests and the `?template=` parameter.

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

See Listing Available Cache Templates.

2.1.4. Retrieving Cache Configuration

Retrieve Infinispan cache configurations with `GET` requests.

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

Table 2. Headers

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

2.1.5. Converting Cache Configurations to JSON

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

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

2.1.6. Retrieving All Cache Details

Invoke a `GET` request to retrieve all details for Infinispan caches.

```
GET /rest/v2/caches/{name}
```

Infinispan provides a JSON response such as the following:

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


- `stats` current stats of the cache.
- `size` the estimated size for the cache.
- `configuration` the cache configuration.

- `rehash_in_progress` true when a rehashing is in progress.
- `indexing_in_progress` true when indexing is in progress.
- `bounded` when expiration is enabled.
- `indexed` true if the cache is indexed.
- `persistent` true if the cache is persisted.
- `transactional` true if the cache is transactional.
- `secured` true if the cache is secured.
- `has_remote_backup` true if the cache has remote backups.

2.1.7. Adding Entries

Add entries to caches with `POST` requests.

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

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

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

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

Table 3. Headers

Header	Required or Optional	Parameter
`Key-Content-Type`	OPTIONAL	Sets the content type for the key in the request. See Key-Content-Type for more information.
`Content-Type`	OPTIONAL	Sets the MediaType of the value for the key.
`timeToLiveSeconds`	OPTIONAL	Sets the number of seconds before the entry is automatically deleted. If you do not set this parameter, Infinispan uses the default value from the configuration. If you set a negative value, the entry is never deleted.

Header	Required or Optional	Parameter
`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, Infinispan 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.

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

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

If *only* `maxIdleTimeSeconds` has a value of `0`, Infinispan 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`, Infinispan uses:

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

2.1.8. Replacing Entries

Replace entries in caches with `PUT` requests.

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

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

2.1.9. Retrieving Data By Keys

Retrieve data for specific keys with `GET` requests.

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

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

Browsers can also access caches directly, for example as a content delivery network (CDN). Infinispan 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 4. Headers

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

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

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

The preceding request returns custom headers:

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

2.1.10. Checking if Entries Exist

Verify that specific entries exists with `HEAD` requests.

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

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

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

Table 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.

2.1.11. Deleting Entries

Remove entries from caches with `DELETE` requests.

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

Table 6. Headers

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

2.1.12. Deleting Caches

Remove caches from Infinispan clusters with `DELETE` requests.

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

2.1.13. Retrieving All Keys from Caches

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

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

Table 7. 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`.
`batch`	OPTIONAL	Specifies the internal batch size when retrieving the keys. The default value is `1000`.

2.1.14. Retrieving All Entries from Caches

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

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

Table 8. 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`.

Infinispan provides a JSON response such as the following:

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

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

2.1.15. Clearing Caches

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

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

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

2.1.16. Getting Cache Size

Retrieve the size of caches across the entire cluster with `GET` requests and the `?action=size`

parameter.

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

2.1.17. Getting Cache Statistics

Obtain runtime statistics for caches with `GET` requests.

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

2.1.18. Querying Caches

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

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

Infinispan responds with query hits such as the following:

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

- `total_results` displays the total number of results from the query.
- `hits` is an array of matches from the query.
- `hit` is an object that matches the query.

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

Table 9. Request Parameters

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

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

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

The following example shows a query in the request body:

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

2.1.19. Re-indexing Data

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

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

Values for the `mode` parameter are as follows:

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

2.1.20. Purging Indexes

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

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

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

2.1.21. Retrieving Query and Index Statistics

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

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

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

Table 10. Request Parameters

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

Infinispan provides a JSON response such as the following:

```
{
  "query": {
    "indexed_local": {
      "count": 1,
      "average": 12344.2,
      "max": 122324,
      "slowest": "FROM Entity WHERE field > 4"
    },
    "indexed_distributed": {
      "count": 0,
      "average": 0.0,
      "max": -1,
      "slowest": "FROM Entity WHERE field > 4"
    },
    "hybrid": {
      "count": 0,
      "average": 0.0,
      "max": -1,
      "slowest": "FROM Entity WHERE field > 4 AND desc = 'value'"
    },
    "non_indexed": {
      "count": 0,
      "average": 0.0,
      "max": -1,
      "slowest": "FROM Entity WHERE desc = 'value'"
    },
    "entity_load": {
      "count": 123,
      "average": 10.0,
      "max": 120
    }
  },
  "index": {
    "types": {
      "org.infinispan.same.test.Entity": {
        "count": 5660001,
        "size": 0
      },
      "org.infinispan.same.test.AnotherEntity": {
        "count": 40,
        "size": 345560
      }
    },
    "reindexing": false
  }
}
```


In the `query` section:

- `indexed_local` Provides details about indexed queries.

- `indexed_distributed` Provides details about distributed indexed queries.
- `hybrid` Provides details about queries that used the index only partially.
- `non_indexed` Provides details about queries that didn't use the index.
- `entity_load` Provides details about cache operations to fetch objects after indexed queries execution.

All time related statistics are in nanoseconds.

In the `index` section:

- `types` Provide details about each indexed type (class name or protobuf message) that is configured in the cache.
  - `count` The number of entities indexed for the type.
  - `size` Usage in bytes of the type.
- `reindexing` If the value is `true`, the `Indexer` is running in the cache.

2.1.22. Clearing Search Statistics

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

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

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

2.1.23. Retrieving Index Statistics (Deprecated)

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

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

Infinispan provides a JSON response such as the following:

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

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

2.1.24. Retrieving Query Statistics (Deprecated)

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

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

Infinispan provides a JSON response such as the following:

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

- `search_query_execution_count` Provides the number of queries that have been run.
- `search_query_total_time` Provides the total time spent on queries.
- `search_query_execution_max_time` Provides the maximum time taken for a query.
- `search_query_execution_avg_time` Provides the average query time.
- `object_loading_total_time` Provides the total time spent loading objects from the cache after query execution.
- `object_loading_execution_max_time` Provides the maximum time spent loading objects execution.
- `object_loading_execution_avg_time` Provides the average time spent loading objects execution.
- `objects_loaded_count` Provides the count of objects loaded.
- `search_query_execution_max_time_query_string` Provides the slowest query executed.

2.1.25. Clearing Query Statistics (Deprecated)

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

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

2.1.26. Listing Caches

List all available caches in Infinispan clusters with `GET` requests.

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

2.1.27. Cross-Site Operations with Caches

Perform cross-site replication operations with the Infinispan REST API.

See Cross Site replication for more details about this feature.

Getting Status of All Backup Locations

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

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

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

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

Table 11. Returned Status

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

Getting Status of Specific Backup Locations

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

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

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

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

Table 12. 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.

Taking Backup Locations Offline

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

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

Bringing Backup Locations Online

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

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

Pushing State to Backup Locations

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

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

Canceling State Transfer

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

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

Getting State Transfer Status

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

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

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

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

Table 13. 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.

Clearing State Transfer Status

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

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

Modifying Take Offline Conditions

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

Procedure

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

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

   The Infinispan response includes `after_failures` and `min_wait` fields as follows:

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

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

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

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

Canceling State Transfer from Receiving Sites

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

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

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

2.1.28. Rolling Upgrades

Perform rolling upgrades of cache data between Infinispan clusters

Synchronizing Data

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

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

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

Disconnecting Source Clusters

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

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

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

2.2. Creating and Managing Counters

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

2.2.1. Creating Counters

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

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

Example Weak Counter

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

Example Strong Counter

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

2.2.2. Deleting Counters

Remove specific counters with `DELETE` requests.

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

2.2.3. Retrieving Counter Configuration

Retrieve configuration for specific counters with `GET` requests.

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

Infinispan 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 14. 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
```

`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}
```

`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
```

`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}
```

Infinispan atomically sets the value to `{update}` if the current value is `{expect}`. If the operation is successful, Infinispan 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}
```

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

2.2.11. Listing Counters

Retrieve a list of counters in Infinispan clusters with `GET` requests.

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

2.3. Working with Protobuf Schemas

Create and manage Protobuf schemas, `.proto` files, via the Infinispan REST API.

2.3.1. Creating Protobuf Schemas

Create Protobuf schemas across Infinispan 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, Infinispan returns HTTP `409 (Conflict)`. If the schema is not valid, either because of syntax errors, or because some of its dependencies are missing, Infinispan stores the schema and returns the error in the response body.

Infinispan 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 Infinispan 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 Infinispan with `GET` requests.

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

2.3.3. Updating Protobuf Schemas

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

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

If the schema is not valid, either because of syntax errors, or because some of its dependencies are missing, Infinispan 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 Infinispan cannot successfully validate the schema, it returns errors.

2.3.4. Deleting Protobuf Schemas

Remove Protobuf schemas from Infinispan 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/
```

Infinispan 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 Infinispan cannot successfully validate the schema, it returns errors.

2.4. Working with Cache Managers

Interact with Infinispan 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}
```

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

```
{
    "version":"xx.x.x-FINAL",
    "name":"default",
    "coordinator":true,
    "cache_configuration_names":[
        "___protobuf_metadata",
        "cache2",
        "CacheManagerResourceTest",
        "cache1"
    ],
    "cluster_name":"ISPN",
    "physical_addresses":"[127.0.0.1:35770]",
    "coordinator_address":"CacheManagerResourceTest-NodeA-49696",
    "cache_manager_status":"RUNNING",
    "created_cache_count":"3",
    "running_cache_count":"3",
    "node_address":"CacheManagerResourceTest-NodeA-49696",
    "cluster_members":[
        "CacheManagerResourceTest-NodeA-49696",
        "CacheManagerResourceTest-NodeB-28120"
    ],
    "cluster_members_physical_addresses":[
        "127.0.0.1:35770",
        "127.0.0.1:60031"
    ],
    "cluster_size":2,
    "defined_caches":[
        {
            "name":"CacheManagerResourceTest",
            "started":true
        },
        {
            "name":"cache1",
            "started":true
        },
        {
            "name":"___protobuf_metadata",
            "started":true
        },
        {
            "name":"cache2",
            "started":true
        }
    ],
    "local_site": "LON",
    "sites_view": [
        "LON",
        "NYC"
    ]
}
```

- `version` contains the Infinispan version
- `name` contains the name of the cache manager as defined in the configuration
- `coordinator` is true if the cache manager is the coordinator of the cluster
- `cache_configuration_names` contains an array of all caches configurations defined in the cache manager
- `cluster_name` contains the name of the cluster as defined in the configuration
- `physical_addresses` contains the physical network addresses associated with the cache manager
- `coordinator_address` contains the physical network addresses of the coordinator of the cluster
- `cache_manager_status` the lifecycle status of the cache manager. For possible values, check the `org.infinispan.lifecycle.ComponentStatus` documentation
- `created_cache_count` number of created caches, excludes all internal and private caches
- `running_cache_count` number of created caches that are running
- `node_address` contains the logical address of the cache manager
- `cluster_members` and `cluster_members_physical_addresses` an array of logical and physical addresses of the members of the cluster
- `cluster_size` number of members in the cluster
- `defined_caches` A list of all caches defined in the cache manager, excluding private caches but including internal caches that are accessible
- `local_site` The name of the local site.
  If cross-site replication is not configured, Infinispan returns "local".
- `sites_view` The list of sites that participate in cross-site replication.
  If cross-site replication is not configured, Infinispan returns an empty list.

2.4.2. Getting Cluster Health

Retrieve health information for Infinispan clusters with `GET` requests.

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

Infinispan 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
```

Infinispan responds with one of the following in `text/plain` format:

- `HEALTHY`
- `HEALTHY_REBALANCING`
- `DEGRADED`
- `FAILED`

2.4.4. Checking REST Endpoint Availability

Verify Infinispan server REST endpoint availability with `HEAD` requests.

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

If you receive a successful response code then the Infinispan 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 15. 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.

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
```

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

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

2.4.7. Listing Available Cache Templates

Retrieve all available Infinispan cache templates with GET requests.

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

See Creating Caches with Templates.

2.4.8. (Experimental) Obtaining Cache Status and Information

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

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

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

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

2.4.9. Getting Cache Manager Statistics

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

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

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

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


- `statistics_enabled` is `true` if statistics collection is enabled for the Cache Manager.
- `read_write_ratio` displays the read/write ratio across all caches.
- `time_since_start` shows the time, in seconds, since the Cache Manager started.
- `time_since_reset` shows the number of seconds since the Cache Manager statistics were last reset.
- `number_of_entries` shows the total number of entries currently in all caches from the Cache Manager. This statistic returns entries in the local cache instances only.
- `total_number_of_entries` shows the number of store operations performed across all caches for the Cache Manager.
- `off_heap_memory_used` shows the amount, in `bytes[]`, of off-heap memory used by this cache container.
- `data_memory_used` shows the amount, in `bytes[]`, that the current eviction algorithm estimates is in use for data across all caches. Returns `0` if eviction is not enabled.
- `misses` shows the number of `get()` misses across all caches.
- `remove_hits` shows the number of removal hits across all caches.
- `remove_misses` shows the number of removal misses across all caches.
- `evictions` shows the number of evictions across all caches.

- `average_read_time` shows the average number of milliseconds taken for `get()` operations across all caches.
- `average_read_time_nanos` same as `average_read_time` but in nanoseconds.
- `average_remove_time` shows the average number of milliseconds for `remove()` operations across all caches.
- `average_remove_time_nanos` same as `average_remove_time` but in nanoseconds.
- `required_minimum_number_of_nodes` shows the required minimum number of nodes to guarantee data consistency.
- `hits` provides the number of `get()` hits across all caches.
- `stores` provides the number of `put()` operations across all caches.
- `current_number_of_entries_in_memory` shows the total number of entries currently in all caches, excluding passivated entries.
- `hit_ratio` provides the total percentage hit/(hit+miss) ratio for all caches.
- `retrievals` shows the total number of `get()` operations.

2.4.10. Backing Up Infinispan 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 16. 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 Infinispan backs up only those resources. See the *Resource Parameters* table for more information.

Table 17. 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.
`tasks`	OPTIONAL	Specifies either an array of server tasks to back up or `*` for all tasks.

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

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

2.4.11. Listing Backups

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

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

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

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

2.4.12. Checking Backup Availability

Verify that a backup operation is complete.

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

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

2.4.13. Downloading Backup Archives

Download backup archives from the server.

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

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

2.4.14. Deleting Backup Archives

Remove backup archives from the server.

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

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

2.4.15. Restoring Infinispan Resources from Backup Archives

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

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.

Restoring from Backup Archives on Infinispan Server

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

Table 18. 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 Infinispan restores only those resources. See the *Resource Parameters* table for more information.

Table 19. 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.
`tasks`	OPTIONAL	Specifies either an array of server tasks to back up or * for all tasks.

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

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

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 20. 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--
```

2.4.16. Listing Restores

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

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

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

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

2.4.17. Checking Restore Progress

Verify that a restore operation is complete.

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

A `201 (Created)` response indicates the restore operation is completed. A `202 (Accepted)` response

indicates the backup operation is in progress.

2.4.18. Deleting Restore Metadata

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

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

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

2.4.19. Cross-Site Operations with Cache Managers

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

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/
```

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

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

Table 21. 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.

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
```

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
```

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
```

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}
```

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
```

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 Infinispan Servers

Monitor and manage Infinispan server instances.

2.5.1. Retrieving Basic Server Information

View basic information about Infinispan servers with `GET` requests.

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

Infinispan 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 Infinispan servers with `GET` requests.

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

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

Infinispan currently supports one cache manager per server only.

2.5.3. Adding Caches to Ignore Lists

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

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

Infinispan 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.

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

2.5.4. Removing Caches from Ignore Lists

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

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

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

2.5.5. Confirming Ignored Caches

Confirm that caches are ignored with `GET` requests.

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

2.5.6. Obtaining Server Configuration

Retrieve Infinispan server configurations with `GET` requests.

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

Infinispan 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 Infinispan servers with `GET` requests.

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

2.5.8. Getting JVM Memory Details

Retrieve JVM memory usage information for Infinispan servers with `GET` requests.

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

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

2.5.9. Getting JVM Thread Dumps

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

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

Infinispan responds with the current thread dump in `text/plain` format.

2.5.10. Getting Diagnostic Reports for Infinispan Servers

Retrieve aggregated reports for Infinispan servers with `GET` requests.

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

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

2.5.11. Stopping Infinispan Servers

Stop Infinispan servers with `POST` requests.

```
POST /rest/v2/server?action=stop
```

Infinispan responds with `204 (No Content)` and then stops running.

2.6. Working with Infinispan Clusters

Monitor and perform administrative tasks on Infinispan clusters.

2.6.1. Stopping Infinispan Clusters

Shut down entire Infinispan clusters with `POST` requests.

```
POST /rest/v2/cluster?action=stop
```

Infinispan responds with `204 (No Content)` and then performs an orderly shutdown of the entire cluster.

2.6.2. Stopping Specific Infinispan Servers in Clusters

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

```
POST /rest/v2/cluster?action=stop&server={server1_host}&server={server2_host}
```

Infinispan responds with `204 (No Content)`.

2.6.3. Backing Up Infinispan 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 22. 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
```

Infinispan 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 Infinispan Cluster Resources

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

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.

Restoring from Backup Archives on Infinispan Server

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

Table 23. JSON Parameters

Key	Required or Optional	Value
`location`	REQUIRED	Specifies the path of the backup archive to restore.

Key	Required or Optional	Value
`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 Infinispan restores only those resources. See the *Resource Parameters* table for more information.

Table 24. 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.
`tasks`	OPTIONAL	Specifies either an array of server tasks to back up or * for all tasks.

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

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

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 25. 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
```

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

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

2.6.10. Checking Restore Progress

Verify that a restore operation is complete.

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

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

2.6.11. Deleting Restore Metadata

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

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

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

2.7. Infinispan Server logging configuration

View and modify the logging configuration on Infinispan 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
```

Infinispan 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
```

Infinispan 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={appe
nder}...
```

Infinispan 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}
```

Infinispan 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 Infinispan server tasks.

2.8.1. Retrieving Server Tasks Information

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

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

Table 26. Request Parameters

Parameter	Required or Optional	Value
`type`	OPTIONAL	`user`: will exclude internal (admin) tasks from the results

Infinispan responds with a list of available tasks. The list includes the names of tasks, the engines that handle tasks, the named parameters for tasks, the execution modes of tasks, either `ONE_NODE` or `ALL_NODES`, and the allowed security role in `JSON` format, as in the following example:

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

2.8.2. Executing Tasks

Execute tasks with `POST` requests that include the task name and required parameters prefixed with

param.

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

Infinispan 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 Infinispan uploads the script, you can execute it with GET requests.

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

2.9. Working with Infinispan 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
```

Infinispan 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
```

Infinispan 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
```

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

```
["observer"]
```

2.9.5. Granting roles to a principal

Grant one or more new roles to a principal.

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

Table 27. 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 /v2/security/roles/some_principal?action=deny&role=role1&role=role2
```

Table 28. Request Parameters

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

Chapter 3. REST Client Examples

Part of the point of a RESTful service is that you don't need to have tightly coupled client libraries/bindings. All you need is a HTTP client library. For Java, Apache HTTP Commons Client works just fine (and is used in the integration tests), or you can use java.net API.

3.1. Ruby REST Example

```
# Shows how to interact with the REST api from ruby.
# No special libraries, just standard net/http
#
# Author: Michael Neale
#
require 'net/http'

uri = URI.parse('http://localhost:11222/rest/v2/caches/default/MyKey')
http = Net::HTTP.new(uri.host, uri.port)

#Create new entry

post = Net::HTTP::Post.new(uri.path, {"Content-Type" => "text/plain"})
post.basic_auth('user','pass')
post.body = "DATA HERE"

resp = http.request(post)

puts "POST response code : " + resp.code

#get it back

get = Net::HTTP::Get.new(uri.path)
get.basic_auth('user','pass')
resp = http.request(get)

puts "GET response code: " + resp.code
puts "GET Body: " + resp.body

#use PUT to overwrite

put = Net::HTTP::Put.new(uri.path, {"Content-Type" => "text/plain"})
put.basic_auth('user','pass')
put.body = "ANOTHER DATA HERE"

resp = http.request(put)

puts "PUT response code : " + resp.code

#and remove...
delete = Net::HTTP::Delete.new(uri.path)
```

```
delete.basic_auth('user','pass')

resp = http.request(delete)

puts "DELETE response code : " + resp.code

#Create binary data like this... just the same...

uri = URI.parse('http://localhost:11222/rest/v2/caches/default/MyLogo')
put = Net::HTTP::Put.new(uri.path, {"Content-Type" => "application/octet-stream"})
put.basic_auth('user','pass')
put.body = File.read('./logo.png')

resp = http.request(put)

puts "PUT response code : " + resp.code

#and if you want to do json...
require 'rubygems'
require 'json'

#now for fun, lets do some JSON !
uri = URI.parse('http://localhost:11222/rest/v2/caches/jsonCache/user')
put = Net::HTTP::Put.new(uri.path, {"Content-Type" => "application/json"})
put.basic_auth('user','pass')

data = {:name => "michael", :age => 42 }
put.body = data.to_json

resp = http.request(put)

puts "PUT response code : " + resp.code

get = Net::HTTP::Get.new(uri.path)
get.basic_auth('user','pass')
resp = http.request(get)

puts "GET Body: " + resp.body
```

3.2. Python 3 REST Example

```
import urllib.request

# Setup basic auth
base_uri = 'http://localhost:11222/rest/v2/caches/default'
auth_handler = urllib.request.HTTPBasicAuthHandler()
auth_handler.add_password(user='user', passwd='pass', realm='ApplicationRealm', uri
=base_uri)
opener = urllib.request.build_opener(auth_handler)
urllib.request.install_opener(opener)

# putting data in
data = "SOME DATA HERE \!"

req = urllib.request.Request(url=base_uri + '/Key', data=data.encode("UTF-8"), method
='PUT',
                             headers={"Content-Type": "text/plain"})
with urllib.request.urlopen(req) as f:
    pass

print(f.status)
print(f.reason)

# getting data out
resp = urllib.request.urlopen(base_uri + '/Key')
print(resp.read().decode('utf-8'))
```

3.3. Java REST Example

```
package org.infinispan;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Base64;

/**
 * Rest example accessing a cache.
 *
 * @author Samuel Tauil (samuel@redhat.com)
 */
public class RestExample {

    /**
     * Method that puts a String value in cache.
     *
```

```
     * @param urlServerAddress URL containing the cache and the key to insert
     * @param value            Text to insert
     * @param user             Used for basic auth
     * @param password         Used for basic auth
     */
    public void putMethod(String urlServerAddress, String value, String user, String
password) throws IOException {
        System.out.println("----------------------------------------");
        System.out.println("Executing PUT");
        System.out.println("----------------------------------------");
        URL address = new URL(urlServerAddress);
        System.out.println("executing request " + urlServerAddress);
        HttpURLConnection connection = (HttpURLConnection) address.openConnection();
        System.out.println("Executing put method of value: " + value);
        connection.setRequestMethod("PUT");
        connection.setRequestProperty("Content-Type", "text/plain");
        addAuthorization(connection, user, password);
        connection.setDoOutput(true);

        OutputStreamWriter outputStreamWriter = new OutputStreamWriter(connection
.getOutputStream());
        outputStreamWriter.write(value);

        connection.connect();
        outputStreamWriter.flush();
        System.out.println("----------------------------------------");
        System.out.println(connection.getResponseCode() + " " + connection
.getResponseMessage());
        System.out.println("----------------------------------------");
        connection.disconnect();
    }

    /**
     * Method that gets a value by a key in url as param value.
     *
     * @param urlServerAddress URL containing the cache and the key to read
     * @param user             Used for basic auth
     * @param password         Used for basic auth
     * @return String value
     */
    public String getMethod(String urlServerAddress, String user, String password)
throws IOException {
        String line;
        StringBuilder stringBuilder = new StringBuilder();

        System.out.println("----------------------------------------");
        System.out.println("Executing GET");
        System.out.println("----------------------------------------");

        URL address = new URL(urlServerAddress);
        System.out.println("executing request " + urlServerAddress);
```

```
        HttpURLConnection connection = (HttpURLConnection) address.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Content-Type", "text/plain");
        addAuthorization(connection, user, password);
        connection.setDoOutput(true);

        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader
(connection.getInputStream()));

        connection.connect();

        while ((line = bufferedReader.readLine()) != null) {
            stringBuilder.append(line).append('\n');
        }

        System.out.println("Executing get method of value: " + stringBuilder.toString
());

        System.out.println("----------------------------------------");
        System.out.println(connection.getResponseCode() + " " + connection
.getResponseMessage());
        System.out.println("----------------------------------------");

        connection.disconnect();

        return stringBuilder.toString();
    }

    private void addAuthorization(HttpURLConnection connection, String user, String
pass) {
        String credentials = user + ":" + pass;
        String basic = Base64.getEncoder().encodeToString(credentials.getBytes());
        connection.setRequestProperty("Authorization", "Basic " + basic);
    }

    /**
     * Main method example.
     */
    public static void main(String[] args) throws IOException {
        RestExample restExample = new RestExample();
        String user = "user";
        String pass = "pass";
        restExample.putMethod("http://localhost:11222/rest/v2/caches/default/1",
"Infinispan REST Test", user, pass);
        restExample.getMethod("http://localhost:11222/rest/v2/caches/default/1", user,
pass);
    }
}
```

3.4. HttpClient API REST Example

```
package org.infinispan;

import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.util.Base64;

/**
 * RestExample class shows you how to access your cache via HttpClient API with Java
11 or later.
 *
 * @author Gustavo Lira (glira@redhat.com)
 */
public class RestExample {
   private static final String SERVER_ADDRESS = "http://localhost:11222";
   private static final String CACHE_URI = "/rest/v2/caches/default";

   /**
    * postMethod create a named cache.
    * @param httpClient HTTP client that sends requests and receives responses
    * @param builder    Encapsulates HTTP requests
    * @throws IOException
    * @throws InterruptedException
    */
   public void postMethod(HttpClient httpClient, HttpRequest.Builder builder) throws
IOException, InterruptedException {
      System.out.println("----------------------------------------");
      System.out.println("Executing POST");
      System.out.println("----------------------------------------");

      HttpRequest request = builder.POST(HttpRequest.BodyPublishers.noBody()).build();
      HttpResponse<Void> response = httpClient.send(request, HttpResponse.
BodyHandlers.discarding());

      System.out.println("----------------------------------------");
      System.out.println(response.statusCode());
      System.out.println("----------------------------------------");
   }

   /**
    * putMethod stores a String value in your cache.
    * @param httpClient HTTP client that sends requests and receives responses
    * @param builder    Encapsulates HTTP requests
    * @throws IOException
    * @throws InterruptedException
    */
```

```
   public void putMethod(HttpClient httpClient, HttpRequest.Builder builder) throws
IOException, InterruptedException {
      System.out.println("----------------------------------------");
      System.out.println("Executing PUT");
      System.out.println("----------------------------------------");

      String cacheValue = "Infinispan REST Test";
      HttpRequest request = builder.PUT(HttpRequest.BodyPublishers.ofString(
cacheValue)).build();
      HttpResponse<Void> response = httpClient.send(request, HttpResponse.
BodyHandlers.discarding());

      System.out.println("----------------------------------------");
      System.out.println(response.statusCode());
      System.out.println("----------------------------------------");
   }

   /**
    * getMethod get a String value from your cache.
    * @param httpClient HTTP client that sends requests and receives responses
    * @param builder    Encapsulates HTTP requests
    * @return           String value
    * @throws IOException
    */
   public String getMethod(HttpClient httpClient, HttpRequest.Builder builder) throws
IOException, InterruptedException {
      System.out.println("----------------------------------------");
      System.out.println("Executing GET");
      System.out.println("----------------------------------------");

      HttpRequest request = builder.GET().build();
      HttpResponse<String> response = httpClient.send(request, HttpResponse
.BodyHandlers.ofString());

      System.out.println("Executing get method of value: " + response.body());

      System.out.println("----------------------------------------");
      System.out.println(response.statusCode());
      System.out.println("----------------------------------------");

      return response.body();
   }

   public static void main(String[] args) throws IOException, InterruptedException {
      RestExample restExample = new RestExample();
      HttpClient httpClient = HttpClient.newBuilder().version(HttpClient.Version
.HTTP_1_1).build();

      restExample.postMethod(httpClient, getHttpReqestBuilder(String.format("%s%s",
SERVER_ADDRESS, CACHE_URI)));
      restExample.putMethod(httpClient, getHttpReqestBuilder(String.format("%s%s/1",
```

```
SERVER_ADDRESS, CACHE_URI)));
      restExample.getMethod(httpClient, getHttpReqestBuilder(String.format("%s%s/1",
SERVER_ADDRESS, CACHE_URI)));
   }

   private static String basicAuth(String username, String password) {
      return "Basic " + Base64.getEncoder().encodeToString((username + ":" + password
).getBytes());
   }

   private static final HttpRequest.Builder getHttpReqestBuilder(String url) {
      return HttpRequest.newBuilder()
            .uri(URI.create(url))
            .header("Content-Type", "text/plain")
            .header("Authorization", basicAuth("user", "pass"));
   }
}
```