Using the Infinispan REST Server

Table of Contents

Chapter 1. REST Server

The Infinispan Server distribution contains a module that implements RESTful HTTP access to the Infinispan data grid, built on Netty.

1.1. Running the REST server

The REST server endpoint is part of the Infinispan Server and by default listens on port 8080. To run the server locally, download the zip distribution and execute in the extracted directory:

```
bin/standalone.sh -b 0.0.0.0
```

or alternatively, run via docker:

```
docker run -it -p 8080:8080 -e "APP_USER=user" -e "APP_PASS=changeme"
jboss/infinispan-server
```

1.1.1. Security

The REST server is protected by authentication, so before usage it is necessary to create an application login. When running via docker, this is achieved by the APP_USER and APP_PASS command line arguments, but when running locally, this can be done with:

```
bin/add-user.sh -u user -p changeme -a
```

1.2. Supported protocols

The REST Server supports HTTP/1.1 as well as HTTP/2 protocols. It is possible to switch to HTTP/2 by either performing a HTTP/1.1 Upgrade procedure or by negotiating communication protocol using TLS/ALPN extension.

Note: TLS/ALPN with JDK8 requires additional steps from the client perspective. Please refer to your client documentation but it is very likely that you will need Jetty ALPN Agent or OpenSSL bindings.

1.3. CORS

The REST server supports CORS including preflight and rules based on the request origin.

Example:

```
<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>
```

The rules are evaluated sequentially based on the "Origin" header set by the browser; in the example above if the origin is either "http://host1" or "https://host1" the rule "restrict host1" will apply, otherwise the next rule will be tested. Since the rule "allow ALL" permits all origins, any script coming from a different origin will be able to perform the methods specified and use the headers supplied.

The <cors-rule> element can be configured as follows:

Config	Description	Mandatory
name	The name of the rule	yes
allow-credentials	Enable CORS requests to use credentials	no
allowed-origins	A comma separated list used to set the CORS 'Access-Control-Allow-Origin' header to indicate the response can be shared with the origins	yes
allowed-methods	A comma separated list used to set the CORS 'Access-Control-Allow-Methods' header in the preflight response to specify the methods allowed for the configured origin(s)	yes
max-age-seconds	The amount of time CORS preflight request headers can be cached	no
expose-headers	A comma separated list used to set the CORS 'Access-Control-Expose-Headers' in the preflight response to specify which headers can be exposed to the configured origin(s)	no

1.4. Data formats

1.4.1. Configuration

Each cache exposed via REST stores data in a configurable data format defined by a MediaType. More details in the configuration here.

An example of storage configuration is as follows:

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

When no MediaType is configured, Infinispan assumes "application/octet-stream" for both keys and values, with the following exceptions:

- If the cache is indexed, it assumes "application/x-protostream"

1.4.2. Supported formats

Data can be written and read in different formats than the storage format; Infinispan can convert between those formats when required.

The following "standard" formats can be converted interchangeably:

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

The following formats can be converted to/from the formats above:

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

Finally, the following conversion is also supported:

- Between *application/x-protostream* and *application/json*

All the REST API calls 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.

1.4.3. Accept header

The REST server is compliant with the RFC-2616 Accept header, and will negotiate the correct MediaType based on the conversions supported. Example, sending the following header when reading data:

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

will cause Infinispan to try first to return content in JSON format (higher priority 0.8). If it's not possible to convert the storage format to JSON, next format tried will be *text/plain* (second highest priority 0.7), and finally it falls back to */*, that will pick a format suitable for displaying automatically based on the cache configuration.

1.4.4. Key-Content-Type header

Most REST API calls have the Key included in the URL. Infinispan will assume the Key is a *java.lang.String* when handling those calls, but it's possible to use a specific header *Key-Content-Type* for keys in different formats.

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*

1.4.5. JSON/Protostream conversion

When caches are indexed, or specifically configured to store *application/x-protostream*, it's possible to send and receive JSON documents that are automatically converted to/from protostream. In order for the conversion to work, a protobuf schema must be registered.

The registration can be done via REST, by doing a POST/PUT in the *___protobuf_metadata* cache. Example using cURL:

```
curl -u user:password -X POST --data-binary @./schema.proto
http://127.0.0.1:8080/rest/___protobuf_metadata/schema.proto
```

When writing a JSON document, a special field ***_type*** must be present in the document to identity the protobuf *Message* corresponding to the document.

For example, consider the following schema:

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

A conformant JSON document would be:

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

1.5. REST V1 API

The REST V1 API supports basic cache capabilities including operations on keys and query, and is now deprecated. For a more powerful and comprehensive API, check the REST V2 API.

HTTP PUT and POST methods are used to place data in the cache, with URLs to address the cache name and key(s) - the data being the body of the request (the data can be anything you like). Other headers are used to control the cache settings and behaviour.

1.5.1. Putting data in

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

A PUT request of the above URL form will place the payload (body) in the given cache, with the given key (the named cache must exist on the server). For example `http://someserver/hr/payRoll-3` (in which case `hr` is the cache name, and `payRoll-3` is the key). Any existing data will be replaced, and Time-To-Live and Last-Modified values etc will updated (if applicable).

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

Exactly the same as PUT, only if a value in a cache/key already exists, it will return a Http CONFLICT status (and the content will not be updated).

Headers

- Key-Content-Type: OPTIONAL The content type for the Key present in the URL.
- Content-Type : OPTIONAL The MediaType of the Value being sent.
- timeToLiveSeconds : OPTIONAL number (the number of seconds before this entry will automatically be deleted). If no parameter is sent, Infinispan assumes configuration default value. Passing any negative value will create an entry which will live forever.
- maxIdleTimeSeconds : OPTIONAL number (the number of seconds after last usage of this entry when it will automatically be deleted). If no parameter is sent, Infinispan configuration default value. Passing any negative value will create an entry which will live forever.

Passing 0 as parameter for timeToLiveSeconds and/or maxIdleTimeSeconds

- If both `timeToLiveSeconds` and `maxIdleTimeSeconds` are 0, the cache will use the default `lifespan` and `maxIdle` values configured in XML/programmatically
- If *only* `maxIdleTimeSeconds` is 0, it uses the `timeToLiveSeconds` value passed as parameter (or -1 if not present), and default `maxIdle` configured in XML/programmatically
- If *only* `timeToLiveSeconds` is 0, it uses default `lifespan` configured in XML/programmatically, and `maxIdle` is set to whatever came as parameter (or -1 if not present)

1.5.2. Getting data back out

HTTP GET and HEAD are used to retrieve data from entries.

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

This will return the data found in the given cacheName, under the given key - as the body of the response. A Content-Type header will be present in the response according to the Media Type negotiation. Browsers can use the cache directly of course (eg as a CDN). An ETag will be returned unique for each entry, as will the Last-Modified and Expires headers field indicating the state of the data at the given URL. ETags allow browsers (and other clients) to ask for data only in the case where it has changed (to save on bandwidth) - this is standard HTTP and is honoured by Infinispan.

Headers

- Key-Content-Type: OPTIONAL The content type for the Key present in the URL. When omitted, *application/x-java-object; type=java.lang.String* is assumed
- Accept: OPTIONAL The required format to return the content

It is possible to obtain additional information by appending the "extended" parameter on the query string, as follows:

`GET /rest/cacheName/cacheKey?extended`

This will return the following custom headers:

- Cluster-Primary-Owner: the node name of the primary owner for this key
- Cluster-Backup-Owners: the node names of the backup owners for this key
- Cluster-Node-Name: the JGroups node name of the server that has handled the request
- Cluster-Physical-Address: the physical JGroups address of the server that has handled the request.

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

The same as GET, only no content is returned (only the header fields). You will receive the same content that you stored. E.g., if you stored a String, this is what you get back. If you stored some XML or JSON, this is what you will receive. If you stored a binary (base 64 encoded) blob, perhaps a serialized; Java; object - you will need to; deserialize this yourself.

Similarly to the GET method, the HEAD method also supports returning extended information via headers. See above.

Headers

- Key-Content-Type: OPTIONAL The content type for the Key present in the URL. When omitted, *application/x-java-object; type=java.lang.String* is assumed

1.5.3. Listing keys

`GET /rest/{cacheName}`

This will return a list of keys present in the given cacheName as the body of the response. The format of the response can be controlled via the Accept header as follows:

- *application/xml* - the list of keys will be returned in XML format.

- *application/json* - the list of keys will be return in JSON format.
- *text/plain* - the list of keys will be returned in plain text format, one key per line

If the cache identified by cacheName is distributed, only the keys owned by the node handling the request will be returned. To return all keys, append the "global" parameter to the query, as follows:

```
GET /rest/cacheName?global
```

1.5.4. Removing data

Data can be removed at the cache key/element level, or via a whole cache name using the HTTP delete method.

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

Removes the given key name from the cache.

Headers

- Key-Content-Type: OPTIONAL The content type for the Key present in the URL. When omitted, *application/x-java-object; type=java.lang.String* is assumed

```
DELETE /rest/{cacheName}
```

Removes ALL the entries in the given cache name (i.e., everything from that path down). If the operation is successful, it returns 200 code.

1.5.5. Querying

The REST server supports Ickle Queries in JSON format. It's important that the cache is configured with *application/x-protostream* for both Keys and Values. If the cache is indexed, no configuration is needed.

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

Will execute an Ickle query in the given cache name.

Request parameters

- *query*: REQUIRED the query string
- *max_results*: OPTIONAL the number of results to return, default is *10*
- *offset*: OPTIONAL the index of the first result to return, default is *0*
- *query_mode*: OPTIONAL the execution mode of the query once it's received by server. Valid values are *FETCH* and *BROADCAST*. Default is *FETCH*.

Query Result

Results are JSON documents containing one or more hits. Example:

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

- *total_results*: NUMBER, the total number of results from the query.
- *hits*: ARRAY, list of matches from the query
- *hit*: OBJECT, each result from the query. Can contain all fields or just a subset of fields in case a *Select* clause is used.

`POST /{cacheName}?action=search`

Similar to que query using GET, but the body of the request is used instead to specify the query parameters.

Example:

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

1.6. REST v2 (version 2) API

The Infinispan REST v2 API improves on the REST v1 API, offering the same features and capabilities in addition to supporting resources beyond caching.

The REST v1 API is deprecated and will not be supported in the next version of Infinispan

1.6.1. Working with Caches

Use the REST API to create and manage caches on your Infinispan cluster and interact with cached entries.

Creating Caches

To create a named cache across the Infinispan cluster, invoke a `POST` request:

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

To configure the cache, you supply the configuration in `XML` or `JSON` format as part of the request payload.

XML Configuration

A configuration in `XML` format must conform to the schema and include:

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

The following example shows a valid `XML` configuration:

```
<infinispan>
    <cache-container>
        <distributed-cache name="cacheName" mode="SYNC">
            <memory>
                <object size="20"/>
            </memory>
        </distributed-cache>
    </cache-container>
</infinispan>
```

JSON Configuration

A configuration in `JSON` format payload:

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

The following example shows the previous `XML` configuration in `JSON` format:

```
{
  "distributed-cache": {
    "mode": "SYNC",
    "memory": {
      "object": {
        "size": 20
      }
    }
  }
}
```

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

Creating Caches with Templates

To create caches across a Infinispan cluster with pre-defined templates, invoke a `POST` request with no payload and an extra request parameter:

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

Retrieving Cache Configuration

To retrieve the configuration of a Infinispan cache, invoke a `GET` request:

```
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.

Converting Cache Configurations

To convert a certain existing cache configuration that is in XML format to JSON, invoke:

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

The POST body must contain a valid cache XML configuration and the response will contain the equivalent JSON representation.

Adding Entries

To add entries to a named cache, invoke a `POST` request:

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

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

If a value already exists for the specified key, the `POST` request returns an HTTP `CONFLICT` status and does not modify the value. To update values, you should use `PUT` requests. See Replacing Entries.

Table 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.
`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.

Replacing Entries

To replace entries in a named cache, invoke a `PUT` request:

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

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

Retrieving Data By Keys

To retrieve data for a specific key in a cache, invoke a `GET` request:

```
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 /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.

Checking if Entries Exist

To check if a specific entry exists in a cache, invoke a `HEAD` request:

```
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.

Deleting Entries

To delete entries from a cache, invoke a `DELETE` request:

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

Infinispan removes the entry under `cacheKey` from the cache.

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.

Removing Caches

To remove caches, invoke a `DELETE` request:

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

Infinispan deletes all data and removes the cache named `cacheName` from the cluster.

Retrieving cache keys

To obtain all the keys from the cache in JSON format, invoke a `GET` request:

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

Table 7. Request Parameters

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

Clearing Caches

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

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

Getting Cache Size

To obtain the size of a cache across the entire cluster, invoke a `GET` request with the `?action=size` parameter:

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

Getting Cache Statistics

To obtain runtime statistics of a cache invoke a `GET` request:

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

Querying Caches

Invoke a `GET` request to perform and Ickle query on a given cache:

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

Infinispan returns one or more query hits in `JSON` format, for example:

```
{
  "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 8. Request Parameters

Parameter	Required or Optional	Value
`query`	REQUIRED	Specifies the query string.
`max_results`	OPTIONAL	Sets the number of results to return. The default is `10`.
`offset`	OPTIONAL	Specifies the index of the first result to return. The default is `0`.
`query_mode`	OPTIONAL	Specifies how the 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 a `POST` request:

```
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
}
```

Listing Caches

To obtain a list of caches available in a Infinispan cluster, invoke a `GET` request:

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

Cross site replication

Use the REST API to monitor and control Cross Site (x-site) replication on your Infinispan cluster. See Cross Site replication for more details about this feature.

Getting status of all backup sites

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

The response contains each site followed by a description of the status. Example:

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

Table 9. Returned Status

Value	Description
`online`	All nodes in the backup site are online
`offline`	All node in the backup site are offline
`mixed`	Some nodes in the backup site are online and others offline. It will include in the status the nodes that are offline. E.g.: `mixed, offline on nodes: Node1, Node2`

Getting status of a backup site

To obtain the status of a single backup site:

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

The response contains each node in the backup site with the status. Example:

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

Table 10. Returned Status

Value	Description
`online`	The node is online
`offline`	The node is offline
`failed`	Failed to obtain status, the remote cache could be shutting down or a network error occurred during the request

Taking a backup site offline

To take a backup site `siteName` offline, for the cache `cacheName`, execute a `GET` request:

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

Bringing a backup site online

To take a backup site `siteName` online, for the cache `cacheName`, execute a `GET` request:

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

Starting a state push to a backup site

To start pushing state of a cache `cacheName` to a remote backup site `siteName`, execute a `GET` request:

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

Cancelling an ongoing state push to a backup site

To cancel a state push of the cache `cacheName` to a remote backup site `siteName`, execute a `GET` request:

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

Getting the status of a state push

To obtain the status of an ongoing state push of cache `cacheName` to backup `siteName` execute a `GET` request:

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

The response will be a JSON document with each destination site name and the state transfer status. Example:

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

The possible statuses are `SENDING`, `OK`, `ERROR` and `CANCELLING`

Tuning the take offline parameters of a remote site

A remote site can be automatically marked as offline in case some conditions are met. To check the configured parameters, execute a `GET` request:

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

The response include two fields, `after_failures` and `min_wait`:

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

To change those parameters, execute a PUT request:

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

with a body containing the new values, e.g.:

```
{
  "after_failures": 4,
  "min_wait": 5000
}
```

Cancelling the receiving state on a site

The main use for this method is when the link between the sites is broken and the receiver site keeps it state transfer state forever.

To set the cluster to normal state in the scope of `cacheName` for state pushed from site `siteName` execute a `GET` request:

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

Clearing the state transfer status of a pushing site

To clear the state transfer status of a (sending) site, execute a `GET` request:

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

1.6.2. Interacting with Cache Managers

The REST API lets you interact with Infinispan Cache Managers to cluster and usage statistics.

Getting Basic Cache Manager Information

To obtain information about a cache manager, invoke a `GET` request:

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

Infinispan responds with a `JSON` document such as the following:

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

}
```


- `version` contains the 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

Getting Cluster Health

To review health information for a Infinispan cluster, invoke a `GET` request:

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

Infinispan responds with a `JSON` document such as the following:

```
{
    "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.
  - `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 or HEALTHY_REBALANCING
  - `cache_name` the name of the cache as defined in the configuration.

Getting Cache Manager Health Status

To retrieve the health status of the cache managers, without the need for authentication, invoke a `GET` request:

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

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

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

Checking REST Endpoint Availability

To check that a Infinispan server REST endpoint is available, invoke a `HEAD` request in the health resource:

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

If the preceding request returns a successful response code then the Infinispan REST server is running and serving requests.

Obtaining Global Configuration for Cache Managers

To obtain the GlobalConfiguration associated with the Cache Manager, invoke a `GET` request:

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

Table 11. 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.

Obtaining Configuration for All Caches

To get the configuration for all caches, invoke a `GET` request:

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

Infinispan responds with a `JSON` array that contains each cache and cache configuration:

```
[
  {
      "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"
              }
          }
      }
  }
]
```

(Experimental) Obtaining caches with cache information

To get the list of all the caches of a cache manager with cache informations, invoke a `GET` request:

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

Infinispan responds with a `JSON` array that contains each cache and the cache information:

```
[ {
  "status" : "RUNNING",
  "name" : "cache1",
  "type" : "local-cache",
  "size" : 123,
  "simple_cache" : false,
  "transactional" : false,
  "persistent" : false,
  "bounded": false,
  "secured": false,
  "indexed": true,
  "has_remote_backup": true
}, {
  "status" : "RUNNING",
  "name" : "cache2",
  "type" : "distributed-cache",
  "size" : 23,
  "simple_cache" : false,
  "transactional" : true,
  "persistent" : false,
  "bounded": false,
  "secured": false,
  "indexed": true,
  "has_remote_backup": true
}]
```

Getting Cache Manager Statistics

To obtain the statistics of a Cache Manager, invoke a `GET` request.

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

Infinispan responds with a `JSON` document that contains the following information:

```
{
    "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.

Managing Cross site replication

The REST API expose several operations to manage cross site replication for all the caches in a cache manager.

Retrieving backup statuses

To retrieve the statuses of all backup sites from the caches of `cacheManagerName`, do a `GET` request:

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

Example response:

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

The `status` field can assume the following values:

- `online`: all caches are online in the backup site.
- `offline`: all caches are offline in the backup site.
- `mixed`: some caches are online and others offline, and their names will be listed in the `online` and `offline` arrays respectively.

Taking a backup site offline

To take all caches from `cacheManagerName` offline, for the backup site `siteName`, execute a `GET` request:

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

Bringing a backup site online

To bring all caches from `cacheManagerName` online, for the backup site `siteName`, execute a `GET` request:

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

Starting a state push

To start pushing state of all caches of `cacheManagerName` to a backup site `siteName`, execute a `GET` request:

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

Cancelling an ongoing state push

To cancel an ongoing state push of all caches of `cacheManagerName` to a remote backup site `siteName`, execute a `GET` request:

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

1.6.3. Working with Counters

Use the REST API to create, delete, and modify counters.

Creating Counters

To create a counter, invoke a `POST` request with the configuration as payload:

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

The payload must contain a configuration for the counter in JSON format, as in the following examples:

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

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

Deleting Counters

To delete a counter, invoke a DELETE request with the counter name:

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

Retrieving Counter Configuration

The get the counter configuration, invoke a GET request with the counter name:

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

Infinispan responds with a JSON representation of the counter configuration.

Adding Values to Counters

To add a value to a named counter, invoke a POST request:

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

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

Request responses depend on the type of counter, as follows:

- `WEAK` counters return empty responses.
- `STRONG` counters return their values after the operation is applied.

This method processes `plain/text` content only.

Getting Counter Values

To retrieve the value of a counter, invoke a `GET` request:

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

Table 12. Headers

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

Resetting Counters

To reset counters, invoke a `GET` request with the `?action=reset` parameter:

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

Incrementing Counters

To increment a counter, invoke a `GET` request with the `?action=increment` parameter:

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

Responses depend on the type of counter, as follows:

- `WEAK` counters return empty responses.
- `STRONG` counters return their values after the operation is applied.

Adding Deltas to Counters

To add an arbitrary amount to a counter, invoke a `GET` request with the `?action=add` and `delta` parameters:

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

Request responses depend on the type of counter, as follows:

- `WEAK` counters return empty responses.
- `STRONG` counters return their values after the operation is applied.

Decrementing Counters

To decrement a counter, invoke a `GET` request with the `?action=decrement` parameter:

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

Request responses depend on the type of counter, as follows:

- `WEAK` counters return empty responses.
- `STRONG` counters return their values after the operation is applied.

compareAndSet Strong Counters

To atomically set the value of a strong counter with the `compareAndSet` method, invoke a `GET` request:

```
GET
/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`.

compareAndSwap Strong Counters

To atomically set the value of a strong counter with the `compareAndSwap` method, invoke a `GET` request:

```
GET
/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.

Listing Counters

To obtain a list of counters available in a Infinispan cluster, invoke a `GET` request:

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

1.6.4. Interacting with Infinispan Servers

The REST API lets you interact with Infinispan servers to retrieve server configuration and information, Java Virtual Machine (JVM) memory usage and thread dumps. You can also perform operations to manage servers.

Retrieving Basic Server Information

To view basic information about a Infinispan server, invoke a `GET` request:

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

The response contains the server name, codename, and version in `JSON` format, as in the following example:

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

Cache Managers

To obtain the list of the server's cache managers:

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

The response will contain an array with the names of the cache managers configured in the server.

Ignoring caches

A cache can be excluded temporarily from receiving requests from clients, returning a code 503 (service unavailable) for REST clients or a Server Error (code 0x85) for Hot Rod clients.

To ignore a cache, use an empty `POST` request with the cache manager name and the cache name:

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

To remove the cache from the ignore list, use a `DELETE` request:

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

Finally, to check the caches that are ignored, do a `GET` request:

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

Currently Infinispan only supports a single cache manager per server, but for future compatibility the name must be provided in the requests above.

Obtaining Server Configuration

To get the configuration for a Infinispan server, invoke a `GET` request:

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

The server responds with the configuration in `JSON` format. The structure follows the server schema, as in the following example:

```
{
    "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"
            }
        }
    }
}
```


Getting Environment Variables

To get environment variables that the server uses, invoke a `GET` request:

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

Getting JVM Memory Details

To get information about JVM memory usage, invoke a `GET` request:

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

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

Getting JVM Thread Dumps

To get the current thread dump for the JVM, invoke a `GET` request:

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

The response is the current thread dump in `text/plain` format.

Stopping Infinispan Servers

To stop the Infinispan server, invoke a `GET` request:

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

The server responds with `200(OK)` and then stops running.

1.6.5. Interacting with Infinispan Clusters

The REST API lets you interact with Infinispan clusters to retrieve cluster-wide configuration and information. You can also perform operations to manage clusters.

Stopping Infinispan Clusters

To gracefully stop Infinispan clusters, invoke a `GET` request:

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

The server responds with `200(OK)` and then the cluster performs a graceful shutdown.

You can also stop one or more specific servers by passing their names:

```
GET /rest/v2/cluster?action=stop&server=server-38760&server=bespin-1223
```

1.7. Client-Side Code

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.

1.7.1. Ruby 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:8080/rest/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:8080/rest/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:8080/rest/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
```

1.7.2. Python 3 example

```
import urllib.request

# Setup basic auth
base_uri = 'http://localhost:8080/rest/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'))
```

1.7.3. Java 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:8080/rest/default/1", "Infinispan REST
Test", user, pass);
        restExample.getMethod("http://localhost:8080/rest/default/1", user, pass);
    }
}
```

1.7.4. REST Example with the HttpClient API

```
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"));
   }
}
```