Using the Infinispan REST Server

Table of Contents

Chapter 1. Infinispan REST Server

Infinispan servers provide RESTful HTTP access to data via a REST module built on Netty.

REST endpoints listen on port 11222 by default.

1.1. REST Authentication

You can configure authentication to the REST endpoint with the `user-tool.sh` script in the Infinispan server distribution.

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

1.2. Supported Protocols

The REST Server supports `HTTP/1.1` and `HTTP/2` protocols.

You can switch to `HTTP/2` with either of the following:

- performing an HTTP/1.1 upgrade procedure.
- negotiating the communication protocol using an TLS/ALPN extension.

TLS/ALPN with JDK8 requires additional client configuration. Refer to the appropriate documentation for your REST client but you are likely to need Jetty ALPN Agent or OpenSSL bindings.

1.3. Configuring Data Formats via the REST API

Each cache exposed via REST stores data in a configurable data format defined by a MediaType.

See the Encoding section for more information about MediaTypes and encoding data with Infinispan.

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"

1.3.1. 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.

1.3.2. Accept Headers

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.

1.3.3. Names with special chars

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 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*

1.3.5. 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/v2/caches/___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
}
```

1.4. Cross-Origin Resource Sharing (CORS) Requests

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
allowed-headers	A comma separated list used to set the CORS 'Access-Control-Allow-Headers' in the preflight response to specify which headers can be used by the configured origin(s)	no

Config	Description	Mandatory
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

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.

The details in this section apply to the REST v2 API only.

The REST v1 API is deprecated and no longer supported in future Infinispan versions.

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

References

- Infinispan XML Configuration
- Infinispan JSON Configuration

XML Configuration

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

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

Example XML Configuration

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

JSON Configuration

Infinispan configuration in JSON format:

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

Example JSON Configuration

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

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}
```

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 retreive 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 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.

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 HTTP `CONFLICT` status instead of modifying values. See Adding Values.

2.1.9. Retrieving Data By Keys

Retrieve data for specific keys with `GET` requests.

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

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

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.

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

2.1.14. Clearing Caches

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

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

2.1.15. Getting Cache Size

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

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

2.1.16. Getting Cache Statistics

Obtain runtime statistics for caches with `GET` requests.

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

2.1.17. Querying Caches

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

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

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 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 `POST` requests as follows:

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

The following example shows a query in the request body:

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

2.1.18. Re-indexing Data

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

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

Values for the `mode` parameter are as follows:

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

2.1.19. Purging Indexes

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

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

2.1.20. Retrieving Index Statistics

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

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

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.

2.1.21. Retrieving Query Statistics

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

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

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.

2.1.22. Clearing Query Statistics

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

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

2.1.23. Listing Caches

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

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

2.1.24. 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 Sites

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

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

Infinispan responds with the status of each backup site in JSON format, as in the following 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 Specific Backup Sites

Retrieve the status of specific backup sites with `GET` requests.

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

Infinispan responds with the status of each node in the backup site in JSON format, as in the following 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 Backup Sites offline

Take backup sites offline for specific caches with `GET` requests and the `?action=take-offline` parameter.

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

Bringing Backup Sites Online

Bring backup sites online for specific caches with `GET` requests and the `?action=bring-online` parameter.

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

Starting State Pushes

Start pushing state of caches to backup sites with `GET` requests and the `?action=start-push-state` parameter.

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

Canceling State Pushes

Cancel state push for caches to backup sites with `GET` requests and the `?action=cancel-push-state` parameter.

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

Clearing State Transfer Status

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

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

Getting Status of State Pushes

Retrieve status of ongoing state pushes to backup sites with `GET` requests.

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

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

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

Table 11. Returned Status

Value	Description
`SENDING`	State push to the backup site is in progress.
`OK`	State is pushed to the backup site successfully.
`ERROR`	Error occurred with state push to the backup site.
`CANCELLING`	State push to the backup site is being canceled.

Tuning Take Offline Parameters of Remote Sites

Remote sites are automatically marked as offline if certain conditions are met.

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
```

Canceling Receiving States

Return Infinispan caches to normal after state is pushed from backup sites with `GET` requests and the `?action=cancel-receive-state` parameter.

This operation is useful if the link between two sites is broken and the receiver site maintains an ongoing state transfer state.

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

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. Adding Values to Counters

Add values to specific counters with `POST` requests.

This method processes `plain/text` content only.

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

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

`WEAK` counters never respond after operations.

`STRONG` counters return the current value after each operation.

2.2.5. Getting Counter Values

Retrieve counter values with `GET` requests.

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

Table 12. Headers

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

2.2.6. Resetting Counters

Restore the intial value of counters without `GET` requests and the `?action=reset` parameter.

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

2.2.7. Incrementing Counters

Increment counter values with `GET` request` and the `?action=increment` parameter.

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

`WEAK` counters never respond after operations.

`STRONG` counters return the current value after each operation.

2.2.8. Adding Deltas to Counters

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

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

`WEAK` counters never respond after operations.

`STRONG` counters return the current value after each operation.

2.2.9. Decrementing Counter Values

Decrement counter values with `GET` requests and the `?action=decrement` parameter.

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

`WEAK` counters never respond after operations.

`STRONG` counters return the current value after each operation.

2.2.10. Performing compareAndSet Operations on Strong Counters

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

```
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`.

2.2.11. Performing compareAndSwap Operations on Strong Counters

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

```
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.

2.2.12. Listing Counters

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

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

2.3. Working with Cache Managers

Interact with Infinispan Cache Managers to get cluster and usage statistics.

2.3.1. 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
        }
    ]

}
```


- `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

2.3.2. 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.
  - `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.

2.3.3. 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`

2.3.4. 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.

2.3.5. Obtaining Global Configuration for Cache Managers

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

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

Table 13. 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

2.3.6. 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"
              }
          }
      }
  }
]
```

2.3.7. Listing Cache Templates

List all templates available for creating caches with GET requests.

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

2.3.8. (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"
}]
```

2.3.9. 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.

2.3.10. Cross-Site Operations with Cache Managers

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

Getting Status of Backup Sites

Retrieve the status for all backup sites 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"
      ]
   }
}
```

The `status` field has 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 Backup Sites Offline

Take backup sites offline from Cache Managers with `GET` requests and the `?action=take-offline` parameter.

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

Bringing Backup Sites Online

Bring backup sites online from Cache Managers with `GET` requests and the `?action=bring-online` parameter.

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

Pushing State

Start pushing state of all caches for a Cache Manager to a backup site with `GET` requests and the `?action=start-push-state` parameter.

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

Canceling State Push Operations

Cancel ongoing state push operations for Cache Managers with `GET` requests and the `?action=cancel-push-state` parameter.

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

2.4. Working with Infinispan Servers

Monitor and manage Infinispan server instances.

2.4.1. 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"
}
```

2.4.2. 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.

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

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

2.4.4. Removing Caches from Ignore Lists

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

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

2.4.5. Confirming Ignored Caches

Confirm that caches are ignored with `GET` requests.

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

2.4.6. 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"
            }
        }
    }
}
```

2.4.7. Getting Environment Variables

Retrieve all environment variables for Infinispan servers with `GET` requests.

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

2.4.8. 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.

2.4.9. 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.

2.4.10. Stopping Infinispan Servers

Stop Infinispan servers with `GET` requests.

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

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

2.5. Working with Infinispan Clusters

Monitor and perform administrative tasks on Infinispan clusters.

2.5.1. Stopping Infinispan Clusters

Shut down entire Infinispan clusters with `GET` requests.

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

Infinispan responds with `200(OK)` and then performs an orderly shutdown of the entire cluster.

2.5.2. 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.

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

Infinispan responds with `200(OK)`.

2.6. Using Server Tasks

Retrieve, execute, and upload Infinispan server tasks.

2.6.1. Retrieving Server Tasks Information

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

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

Table 14. 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"
  }
]
```

2.6.2. Executing Tasks

Execute tasks with `GET` requests that include the task name and required parameters prefixed with `param`.

```
GET /rest/v2/tasks/myTask?action=exec&param.p1=v1&param.p2=v2
```

Infinispan responds with the task result.

2.6.3. 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
```

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:8080/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:8080/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:8080/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:8080/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:8080/rest/v2/caches/default/1",
"Infinispan REST Test", user, pass);
        restExample.getMethod("http://localhost:8080/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"));
   }
}
```