# Red Hat Data Grid 8.0

## Data Grid Command Line Interface

Data Grid Documentation

Last Updated: 2020-06-02

# Red Hat Data Grid 8.0 Data Grid Command Line Interface

Data Grid Documentation

## Legal Notice



## Abstract

Connect to Data Grid servers via the command line interface (CLI) to access data and perform management operations.

# Table of Contents

CHAPTER 1. RED HAT DATA GRID ..... 6
1.1. DATA GRID DOCUMENTATION 6
1.2. DATA GRID DOWNLOADS 6

CHAPTER 2. GETTING STARTED WITH THE DATA GRID CLI ..... 7
2.1. STARTING THE DATA GRID CLI 7
2.2. CONNECTING TO DATA GRID SERVERS 7
2.3. NAVIGATING CLI RESOURCES 7
2.3.1. CLI Resources 8
2.4. RESOURCE STATISTICS 9
2.5. SHUTTING DOWN DATA GRID SERVERS 10

CHAPTER 3. PERFORMING CACHE OPERATIONS WITH THE DATA GRID CLI ..... 12
3.1. CREATING CACHES FROM TEMPLATES 12
3.2. CREATING CACHES FROM XML OR JSON FILES 12
3.2.1. XML Configuration 13
3.2.2. JSON Configuration 13
3.3. ADDING CACHE ENTRIES 13
3.4. CLEARING CACHES AND DELETING ENTRIES 14
3.5. DELETING CACHES 14

CHAPTER 4. PERFORMING BATCH OPERATIONS ..... 15
4.1. PERFORMING BATCH OPERATIONS WITH FILES 15
4.2. PERFORMING BATCH OPERATIONS INTERACTIVELY 15

CHAPTER 5. WORKING WITH COUNTERS ..... 17
5.1. CREATING COUNTERS 17
5.2. ADDING DELTAS TO COUNTERS 18

CHAPTER 6. QUERYING CACHES WITH PROTOBUF METADATA ..... 19
6.1. CONFIGURING MEDIA TYPES 19
6.2. REGISTERING PROTOBUF SCHEMAS 20
6.3. QUERYING CACHES WITH PROTOBUF SCHEMAS 21

CHAPTER 7. PERFORMING CROSS-SITE REPLICATION OPERATIONS ..... 24
7.1. BRINGING BACKUP LOCATIONS OFFLINE AND ONLINE 24
7.2. PUSHING STATE TO BACKUP LOCATIONS 24

CHAPTER 8. PATCHING DATA GRID SERVER INSTALLATIONS ..... 25
8.1. DATA GRID SERVER PATCHES 25
8.2. DOWNLOADING SERVER PATCHES 25
8.3. CREATING SERVER PATCHES 26
8.4. INSTALLING SERVER PATCHES 27
8.5. ROLLING BACK SERVER PATCHES 28

CHAPTER 9. COMMAND REFERENCE ..... 30
9.1. ADD(1) 30
9.1.1. NAME 30
9.1.2. SYNOPSIS 30
9.1.3. OPTIONS 30
9.1.4. EXAMPLES 30
9.1.5. SEE ALSO 30
9.2. CACHE(1) 30

9.2.1. NAME 30
9.2.2. SYNOPSIS 30
9.2.3. EXAMPLE 31
9.2.4. SEE ALSO 31
9.3. CAS(1) 31
9.3.1. NAME 31
9.3.2. SYNOPSIS 31
9.3.3. OPTIONS 31
9.3.4. EXAMPLE 31
9.3.5. SEE ALSO 31
9.4. CD(1) 31
9.4.1. NAME 31
9.4.2. DESCRIPTION 31
9.4.3. SYNOPSIS 31
9.4.4. EXAMPLE 32
9.4.5. SEE ALSO 32
9.5. CLEARCACHE(1) 32
9.5.1. NAME 32
9.5.2. SYNOPSIS 32
9.5.3. EXAMPLES 32
9.5.4. SEE ALSO 32
9.6. CONNECT(1) 32
9.6.1. NAME 32
9.6.2. DESCRIPTION 32
9.6.3. SYNOPSIS 32
9.6.4. OPTIONS 32
9.6.5. EXAMPLE 32
9.6.6. SEE ALSO 33
9.7. CONTAINER(1) 33
9.7.1. NAME 33
9.7.2. SYNOPSIS 33
9.7.3. EXAMPLE 33
9.7.4. SEE ALSO 33
9.8. COUNTER(1) 33
9.8.1. NAME 33
9.8.2. SYNOPSIS 33
9.8.3. EXAMPLE 33
9.8.4. SEE ALSO 33
9.9. CREATE(1) 33
9.9.1. NAME 33
9.9.2. SYNOPSIS 34
9.9.3. CREATE CACHE OPTIONS 34
9.9.4. CREATE COUNTER OPTIONS 34
9.9.5. EXAMPLES 34
9.9.6. SEE ALSO 34
9.10. DESCRIBE(1) 34
9.10.1. NAME 34
9.10.2. SYNOPSIS 35
9.10.3. EXAMPLES 35
9.10.4. SEE ALSO 35
9.11. DISCONNECT(1) 35
9.11.1. NAME 35
9.11.2. SYNOPSIS 35

9.11.3. EXAMPLE 35
9.11.4. SEE ALSO 35
9.12. DROP(1) 35
9.12.1. NAME 35
9.12.2. SYNOPSIS 35
9.12.3. EXAMPLES 36
9.12.4. SEE ALSO 36
9.13. ENCODING(1) 36
9.13.1. NAME 36
9.13.2. DESCRIPTION 36
9.13.3. SYNOPSIS 36
9.13.4. EXAMPLE 36
9.13.5. SEE ALSO 36
9.14. GET(1) 36
9.14.1. NAME 36
9.14.2. SYNOPSIS 37
9.14.3. OPTIONS 37
9.14.4. EXAMPLE 37
9.14.5. SEE ALSO 37
9.15. HELP(1) 37
9.15.1. NAME 37
9.15.2. SYNOPSIS 37
9.15.3. EXAMPLE 37
9.15.4. SEE ALSO 37
9.16. LS(1) 37
9.16.1. NAME 37
9.16.2. SYNOPSIS 37
9.16.3. EXAMPLES 37
9.16.4. SEE ALSO 38
9.17. PATCH(1) 38
9.17.1. NAME 38
9.17.2. DESCRIPTION 38
9.17.3. SYNOPSIS 38
9.17.4. PATCH LIST OPTIONS 38
9.17.5. PATCH INSTALL OPTIONS 38
9.17.6. PATCH DESCRIBE OPTIONS 38
9.17.7. PATCH ROLLBACK OPTIONS 39
9.17.8. PATCH CREATE OPTIONS 39
9.17.9. EXAMPLES 39
9.18. PUT(1) 39
9.18.1. NAME 39
9.18.2. DESCRIPTION 39
9.18.3. SYNOPSIS 39
9.18.4. OPTIONS 39
9.18.5. EXAMPLES 40
9.18.6. SEE ALSO 40
9.19. QUERY(1) 40
9.19.1. NAME 40
9.19.2. SYNOPSIS 40
9.19.3. OPTIONS 40
9.19.4. EXAMPLES 41
9.19.5. SEE ALSO 41
9.20. QUIT(1) 41

9.20.1. NAME 41
9.20.2. SYNOPSIS 41
9.20.3. EXAMPLE 41
9.20.4. SEE ALSO 41
9.21. REMOVE(1) 41
9.21.1. NAME 41
9.21.2. SYNOPSIS 41
9.21.3. OPTIONS 41
9.21.4. EXAMPLE 41
9.21.5. SEE ALSO 42
9.22. RESET(1) 42
9.22.1. NAME 42
9.22.2. SYNOPSIS 42
9.22.3. EXAMPLE 42
9.22.4. SEE ALSO 42
9.23. SCHEMA(1) 42
9.23.1. NAME 42
9.23.2. SYNOPSIS 42
9.23.3. OPTIONS 42
9.23.4. EXAMPLE 42
9.23.5. SEE ALSO 42
9.24. SHUTDOWN(1) 43
9.24.1. NAME 43
9.24.2. SYNOPSIS 43
9.24.3. EXAMPLES 43
9.24.4. SEE ALSO 43
9.25. SITE(1) 43
9.25.1. NAME 43
9.25.2. SYNOPSIS 43
9.25.3. OPTIONS 43
9.25.4. EXAMPLES 44
9.26. TASK(1) 44
9.26.1. NAME 44
9.26.2. SYNOPSIS 44
9.26.3. EXAMPLES 44
9.26.4. OPTIONS 44
9.26.5. SEE ALSO 45
9.27. VERSION(1) 45
9.27.1. NAME 45
9.27.2. SYNOPSIS 45
9.27.3. EXAMPLE 45
9.27.4. SEE ALSO 45

# CHAPTER 1. RED HAT DATA GRID

Data Grid is a high-performance, distributed in-memory data store.

**Schemaless data structure**

Flexibility to store different objects as key-value pairs.

**Grid-based data storage**

Designed to distribute and replicate data across clusters.

**Elastic scaling**

Dynamically adjust the number of nodes to meet demand without service disruption.

**Data interoperability**

Store, retrieve, and query data in the grid from different endpoints.

## 1.1. DATA GRID DOCUMENTATION

Documentation for Data Grid is available on the Red Hat customer portal.

- Data Grid 8.0 Documentation
- Data Grid 8.0 Component Details
- Supported Configurations for Data Grid 8.0

## 1.2. DATA GRID DOWNLOADS

Access the Data Grid Software Downloads on the Red Hat customer portal.

**NOTE**

You must have a Red Hat account to access and download Data Grid software.

# CHAPTER 2. GETTING STARTED WITH THE DATA GRID CLI

The command line interface (CLI) lets you remotely connect to Data Grid servers to access data and perform administrative functions.

**Prerequisites**

- At least one running Data Grid server.

## 2.1. STARTING THE DATA GRID CLI

Start the Data Grid CLI as follows:

1. Open a terminal in **$ISPN_HOME**.
2. Run the CLI.

```
$ bin/cli.sh
[disconnected]>
```

## 2.2. CONNECTING TO DATA GRID SERVERS

Do one of the following:

- Run the **connect** command to connect to a Data Grid server on the default port of **11222**:

```
[disconnected]> connect
[hostname1@cluster//containers/default]>
```

- Specify the location of a Data Grid server. For example, connect to a local server that has a port offset of 100:

```
[disconnected]> connect 127.0.0.1:11322
[hostname2@cluster//containers/default]>
```

**TIP**

Press the tab key to display available commands and options. Use the **-h** option to display help text.

## 2.3. NAVIGATING CLI RESOURCES

The Data Grid CLI exposes a navigable tree that allows you to list, describe, and manipulate Data Grid cluster resources.

When you connect to a Data Grid cluster, it opens in the context of the default cache container.

```
[//containers/default]>
```

- Use **ls** to list resources.

```
[//containers/default]> ls
caches
```

```
counters
configurations
schemas
tasks
```

- Use **cd** to navigate the resource tree.

```
[//containers/default]> cd caches
```

- Use **describe** to view information about resources.

```
[//containers/default]> describe
{
  "name" : "default",
  "version" : "xx.x.x-FINAL",
  "cluster_name" : "cluster",
  "coordinator" : true,
  "cache_configuration_names" : [ "org.infinispan.REPL_ASYNC", "___protobuf_metadata",
"org.infinispan.DIST_SYNC", "org.infinispan.LOCAL", "org.infinispan.INVALIDATION_SYNC",
"org.infinispan.REPL_SYNC", "org.infinispan.SCATTERED_SYNC",
"org.infinispan.INVALIDATION_ASYNC", "org.infinispan.DIST_ASYNC" ],
  "physical_addresses" : "[192.0.2.0:7800]",
  "coordinator_address" : "<hostname>",
  "cache_manager_status" : "RUNNING",
  "created_cache_count" : "1",
  "running_cache_count" : "1",
  "node_address" : "<hostname>",
  "cluster_members" : [ "<hostname1>", "<hostname2>" ],
  "cluster_members_physical_addresses" : [ "192.0.2.0:7800", "192.0.2.0:7801" ],
  "cluster_size" : 2,
  "defined_caches" : [ {
    "name" : "mycache",
    "started" : true
  }, {
    "name" : "___protobuf_metadata",
    "started" : true
  } ]
}
```

### 2.3.1. CLI Resources

The Data Grid CLI exposes different resources to:

- create, modify, and manage local or clustered caches.
- perform administrative operations for Data Grid clusters.

#### Cache Resources

```
[//containers/default]> ls
caches
counters
configurations
schemas
```

**caches**

Data Grid cache instances. The default cache container is empty. Use the CLI to create caches from templates or **infinispan.xml** files.

**counters**

**Strong** or **Weak** counters that record the count of objects.

**configurations**

Data Grid configurations.

**schemas**

Protocol Buffers (Protobuf) schemas that structure data in the cache.

**tasks**

Remote tasks creating and managing Data Grid cache definitions.

### Cluster Resources

```
[hostname@cluster/]> ls
containers
cluster
server
```

**containers**

Cache containers on the Data Grid cluster.

**cluster**

Lists Data Grid servers joined to the cluster.

**server**

Resources for managing and monitoring Data Grid servers.

## 2.4. RESOURCE STATISTICS

You can inspect server-collected statistics for some of the resources within a Data Grid server using the **stats** command. Use the **stats** command either from the context of a resource which collects statistics (containers, caches) or with a path to such a resource:

```
[//containers/default]> stats
{
  "statistics_enabled" : true,
  "number_of_entries" : 0,
  "hit_ratio" : 0.0,
  "read_write_ratio" : 0.0,
  "time_since_start" : 0,
  "time_since_reset" : 49,
  "current_number_of_entries" : 0,
  "current_number_of_entries_in_memory" : 0,
  "total_number_of_entries" : 0,
  "off_heap_memory_used" : 0,
  "data_memory_used" : 0,
  "stores" : 0,
  "retrievals" : 0,
  "hits" : 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
}
```

```
[//containers/default]> stats /containers/default/caches/mycache
{
  "time_since_start" : -1,
  "time_since_reset" : -1,
  "current_number_of_entries" : -1,
  "current_number_of_entries_in_memory" : -1,
  "total_number_of_entries" : -1,
  "off_heap_memory_used" : -1,
  "data_memory_used" : -1,
  "stores" : -1,
  "retrievals" : -1,
  "hits" : -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
}
```

## 2.5. SHUTTING DOWN DATA GRID SERVERS

Use the CLI to gracefully shutdown running servers. This ensures that Data Grid passivates all entries to disk and persists state.

- Use the **shutdown server** command to stop individual servers.

  ```
  [//containers/default]> shutdown server $hostname
  ```

- Use the **shutdown cluster** command to stop all servers joined to the cluster.

  ```
  [//containers/default]> shutdown cluster
  ```

### Verification

Check the server logs for the following messages:

```
ISPN080002: Data Grid Server stopping
```

```
ISPN000080: Disconnecting JGroups channel cluster
ISPN000390: Persisted state, version=<$version> timestamp=YYYY-MM-DDTHH:MM:SS
ISPN080003: Data Grid Server stopped
```

# CHAPTER 3. PERFORMING CACHE OPERATIONS WITH THE DATA GRID CLI

The command line interface (CLI) lets you remotely connect to Data Grid servers to access data and perform administrative functions.

**Prerequisites**

- Start the Data Grid CLI.
- Connect to a running Data Grid cluster.

## 3.1. CREATING CACHES FROM TEMPLATES

Use Data Grid cache templates to add caches with recommended default settings.

**Procedure**

1. Create a distributed, synchronous cache from a template and name it "mycache".

```
[//containers/default]> create cache --template=org.infinispan.DIST_SYNC mycache
```

**TIP**

Press the tab key after the **--template=** argument to list available cache templates.

2. Retrieve the cache configuration.

```
[//containers/default]> describe caches/mycache
{
  "distributed-cache" : {
    "mode" : "SYNC",
    "remote-timeout" : 17500,
    "state-transfer" : {
      "timeout" : 60000
    },
    "transaction" : {
      "mode" : "NONE"
    },
    "locking" : {
      "concurrency-level" : 1000,
      "acquire-timeout" : 15000,
      "striping" : false
    },
    "statistics" : true
  }
}
```

## 3.2. CREATING CACHES FROM XML OR JSON FILES

Add caches with custom Data Grid configuration in XML or JSON format.

**Procedure**

- Add the path to your configuration file with the **--file=** option as follows:

```
[//containers/default]> create cache --file=prod_dist_cache.xml dist_cache_01
```

### 3.2.1. XML Configuration

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

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

**Example XML Configuration**

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

### 3.2.2. JSON Configuration

Data Grid configuration in JSON format:

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

**Example JSON Configuration**

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

## 3.3. ADDING CACHE ENTRIES

Add data to caches with the Data Grid CLI.

**Prerequisites**

- Create a cache named "mycache" and **cd** into it.

```
[//containers/default]> cd caches/mycache
```

**Procedure**

1. Put an entry into "mycache".

```
[//containers/default/caches/mycache]> put hello world
```

**TIP**

If not in the context of a cache, use the **--cache=** parameter. For example:

```
[//containers/default]> put --cache=mycache hello world
```

2. Get the entry to verify it.

```
[//containers/default/caches/mycache]> get hello
world
```

## 3.4. CLEARING CACHES AND DELETING ENTRIES

Remove data from caches with the Data Grid CLI.

**Procedure**

- Clear caches. This command deletes all entries from a cache.

```
[//containers/default]> clearcache mycache
```

- Remove specific entries from a cache.

```
[//containers/default]> remove --cache=mycache hello
```

## 3.5. DELETING CACHES

Drop caches to remove them and delete all data they contain.

**Procedure**

- Remove caches with the **drop** command.

```
[//containers/default]> drop cache mycache
```

# CHAPTER 4. PERFORMING BATCH OPERATIONS

Process operations in groups, either interactively or using batch files.

**Prerequisites**

- A running Data Grid cluster.

## 4.1. PERFORMING BATCH OPERATIONS WITH FILES

Create files that contain a set of operations and then pass them to the Data Grid CLI.

**Procedure**

1. Create a file that contains a set of operations.
   For example, create a file named **batch** that creates a cache named **mybatch**, adds two entries to the cache, and disconnects from the CLI.

```
$ cat > batch<<EOF
create cache --template=org.infinispan.DIST_SYNC mybatch
put --cache=mybatch hello world
put --cache=mybatch hola mundo
disconnect
EOF
```

2. Run the CLI and specify the file as input.

```
$ bin/cli.sh -c localhost:11222 -f batch
```

3. Open a new CLI connection to Data Grid and verify **mybatch**.

```
[//containers/default]> ls caches
___protobuf_metadata
mybatch
[//containers/default]> ls caches/mybatch
hola
hello
[//containers/default]> disconnect
[disconnected]>
```

## 4.2. PERFORMING BATCH OPERATIONS INTERACTIVELY

Use the standard input stream, **stdin**, to perform batch operations interactively.

**Procedure**

1. Start the Data Grid CLI in interactive mode.

```
$ bin/cli.sh -c localhost:11222 -f -
```

NOTE

If you do not use the **-c** flag, you must run the **connect** command.

```
$ bin/cli.sh -f -
connect
```

2. Run batch operations, for example:

```
create cache --template=org.infinispan.DIST_SYNC mybatch
put --cache=mybatch hello world
put --cache=mybatch hola mundo
disconnect
quit
```

TIP

Use **echo** to add commands in interactive mode.

The following example shows how to use **echo describe** to get cluster information:

```
$ echo describe|bin/cli.sh -c localhost:11222 -f -
{
  "name" : "default",
  "version" : "10.0.0-SNAPSHOT",
  "coordinator" : false,
  "cache_configuration_names" : [ "org.infinispan.REPL_ASYNC", "___protobuf_metadata",
"org.infinispan.DIST_SYNC", "qcache", "org.infinispan.LOCAL", "dist_cache_01",
"org.infinispan.INVALIDATION_SYNC", "org.infinispan.REPL_SYNC",
"org.infinispan.SCATTERED_SYNC", "mycache", "org.infinispan.INVALIDATION_ASYNC",
"mybatch", "org.infinispan.DIST_ASYNC" ],
  "cluster_name" : "cluster",
  "physical_addresses" : "[192.168.1.7:7800]",
  "coordinator_address" : "thundercat-34689",
  "cache_manager_status" : "RUNNING",
  "created_cache_count" : "4",
  "running_cache_count" : "4",
  "node_address" : "thundercat-47082",
  "cluster_members" : [ "thundercat-34689", "thundercat-47082" ],
  "cluster_members_physical_addresses" : [ "10.36.118.25:7801", "192.168.1.7:7800" ],
  "cluster_size" : 2,
  "defined_caches" : [ {
    "name" : "___protobuf_metadata",
    "started" : true
  }, {
    "name" : "mybatch",
    "started" : true
  } ]
}
```

# CHAPTER 5. WORKING WITH COUNTERS

Counters provide atomic increment and decrement operations that record the count of objects.

**Prerequisites**

- Start the Data Grid CLI.
- Connect to a running Data Grid cluster.

## 5.1. CREATING COUNTERS

Create strong and weak counters with the Data Grid CLI.

**Procedure**

1. Run **create counter** with the appropriate arguments.
   a. Create **my-weak-counter**.

```
[//containers/default]> create counter --concurrency-level=1 --initial-value=5 --
storage=PERSISTENT --type=weak my-weak-counter
```

   b. Create **my-strong-counter**.

```
[//containers/default]> create counter --initial-value=3 --storage=PERSISTENT --
type=strong my-strong-counter
```

2. List available counters.

```
[//containers/default]> ls counters
my-strong-counter
my-weak-counter
```

3. Verify counter configurations.
   a. Describe **my-weak-counter**.

```
[//containers/default]> describe counters/my-weak-counter

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

   b. Describe **my-strong-counter**.

```
[//containers/default]> describe counters/my-strong-counter

{
```

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

## 5.2. ADDING DELTAS TO COUNTERS

Increment or decrement counters with arbitrary values.

**Procedure**

1. Select a counter.

```
[//containers/default]> counter my-weak-counter
```

2. List the current count.

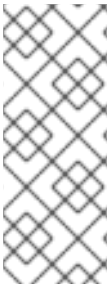
```
[//containers/default/counters/my-weak-counter]> ls
5
```

3. Increment the counter value by **2**.

```
[//containers/default/counters/my-weak-counter]> add --delta=2
```

4. Decrement the counter value by **-4**.

```
[//containers/default/counters/my-weak-counter]> add --delta=-4
```



NOTE

Strong counters return values after the operation is applied. Use **--quiet=true** to hide the return value.

For example, **my-strong-counter]> add --delta=3 --quiet=true**.

Weak counters return empty responses.

# CHAPTER 6. QUERYING CACHES WITH PROTOBUF METADATA

Data Grid supports using Protocol Buffers (Protobuf) to structure data in the cache so that you can query it.

**Prerequisites**

- Start the Data Grid CLI.
- Connect to a running Data Grid cluster.

## 6.1. CONFIGURING MEDIA TYPES

Encode cache entries with different media types to store data in a format that best suits your requirements.

For example, the following procedure shows you how to configure the **application/x-protostream** media type.

**Procedure**

1. Create a Data Grid configuration file that adds a distributed cache named **qcache** and configures the media type, for example:

```
<infinispan>
  <cache-container>
    <distributed-cache name="qcache">
      <encoding>
        <key media-type="application/x-protostream"/>
        <value media-type="application/x-protostream"/>
      </encoding>
    </distributed-cache>
  </cache-container>
</infinispan>
```

2. Create **qcache** from **pcache.xml** with the **--file=** option.

```
[//containers/default]> create cache --file=pcache.xml pcache
```

3. Verify **pcache**.

```
[//containers/default]> ls caches
pcache
___protobuf_metadata
[//containers/default]> describe caches/pcache
{
  "distributed-cache" : {
    "mode" : "SYNC",
    "encoding" : {
      "key" : {
        "media-type" : "application/x-protostream"
      },
```

```
      "value" : {
        "media-type" : "application/x-protostream"
      }
    },
    "transaction" : {
      "mode" : "NONE"
    }
  }
}
```

4. Add an entry to **pcache** and check the encoding.

```
[//containers/default]> put --cache=pcache good morning
[//containers/default]> cd caches/pcache
[//containers/default/caches/pcache]> get good
{
  "_type" : "string",
  "_value" : "morning"
}
```

## 6.2. REGISTERING PROTOBUF SCHEMAS

Protobuf schemas contain data structures known as messages in **.proto** definition files.

### Procedure

1. Create a schema file named **person.proto** with the following messages:

```
package org.infinispan.rest.search.entity;

message Address {
    required string street = 1;
    required string postCode = 2;
}

message PhoneNumber {
    required string number = 1;
}

message Person {
    optional int32 id = 1;
    required string name = 2;
    required string surname = 3;
    optional Address address = 4;
    repeated PhoneNumber phoneNumbers = 5;
    optional uint32 age = 6;
    enum Gender {
        MALE = 0;
        FEMALE = 1;
    }

    optional Gender gender = 7;
}
```

2. Register **person.proto**.

```
[//containers/default]> schema --upload=person.proto person.proto
```

3. Verify **person.proto**.

```
[//containers/default]> cd caches/___protobuf_metadata
[//containers/default/caches/___protobuf_metadata]> ls
person.proto
[//containers/default/caches/___protobuf_metadata]> get person.proto
```

## 6.3. QUERYING CACHES WITH PROTOBUF SCHEMAS

Data Grid automatically converts JSON to Protobuf so that you can read and write cache entries in JSON format and use Protobuf schemas to query them.

For example, consider the following JSON documents:

**lukecage.json**

```
{
  "_type":"org.infinispan.rest.search.entity.Person",
  "id":2,
  "name":"Luke",
  "surname":"Cage",
  "gender":"MALE",
  "address":{"street":"38th St","postCode":"NY 11221"},
  "phoneNumbers":[{"number":4444},{"number":5555}]
}
```

**jessicajones.json**

```
{
  "_type":"org.infinispan.rest.search.entity.Person",
  "id":1,
  "name":"Jessica",
  "surname":"Jones",
  "gender":"FEMALE",
  "address":{"street":"46th St","postCode":"NY 10036"},
  "phoneNumbers":[{"number":1111},{"number":2222},{"number":3333}]
}
```

**matthewmurdock.json**

```
{
  "_type":"org.infinispan.rest.search.entity.Person",
  "id":3,
  "name":"Matthew",
  "surname":"Murdock",
  "gender":"MALE",
  "address":{"street":"57th St","postCode":"NY 10019"},
  "phoneNumbers":[]
}
```

Each of the preceding JSON documents contains:

- a **_type** field that identifies the Protobuf message to which the JSON document corresponds.
- several fields that correspond to datatypes in the **person.proto** schema.

**Procedure**

1. Navigate to the **pcache** cache.

```
[//containers/default/caches]> cd pcache
```

2. Add each JSON document as an entry to the cache, for example:

```
[//containers/default/caches/pcache]> put --encoding=application/json --file=jessicajones.json
jessicajones
[//containers/default/caches/pcache]> put --encoding=application/json --
file=matthewmurdock.json matthewmurdock
[//containers/default/caches/pcache]> put --encoding=application/json --file=lukecage.json
lukecage
```

3. Verify that the entries exist.

```
[//containers/default/caches/pcache]> ls
lukecage
matthewmurdock
jessicajones
```

4. Query the cache to return entries from the Protobuf **Person** entity where the gender datatype is **MALE**.

```
[//containers/default/caches/pcache]> query "from org.infinispan.rest.search.entity.Person p
where p.gender = 'MALE'"
{
  "total_results" : 2,
  "hits" : [ {
    "hit" : {
      "_type" : "org.infinispan.rest.search.entity.Person",
      "id" : 2,
      "name" : "Luke",
      "surname" : "Cage",
      "gender" : "MALE",
      "address" : {
        "street" : "38th St",
        "postCode" : "NY 11221"
      },
      "phoneNumbers" : [ {
        "number" : "4444"
      }, {
        "number" : "5555"
      } ]
    }
  }, {
    "hit" : {
      "_type" : "org.infinispan.rest.search.entity.Person",
```

```
      "id" : 3,
      "name" : "Matthew",
      "surname" : "Murdock",
      "gender" : "MALE",
      "address" : {
        "street" : "57th St",
        "postCode" : "NY 10019"
      }
    }
  } ]
}
```

# CHAPTER 7. PERFORMING CROSS-SITE REPLICATION OPERATIONS

Data Grid clusters running in different locations can discover and communicate with each other to backup data.

**Prerequisites**

- Start the Data Grid CLI.
- Connect to a running Data Grid cluster.

## 7.1. BRINGING BACKUP LOCATIONS OFFLINE AND ONLINE

Take backup locations offline manually and bring them back online.

**Procedure**

- Check if backup locations are online or offline with the **site status** command:

```
//containers/default]> site status --cache=cacheName --site=NYC
```

NOTE

**--site** is an optional argument. If not set, the CLI returns all backup locations.

- Bring backup locations online with the **bring-online** command:

```
//containers/default]> site bring-online --cache=customers --site=NYC
```

- Take backup locations offline with the **take-offline** command:

```
//containers/default]> site take-offline --cache=customers --site=NYC
```

For more information and examples, run the **help site** command.

## 7.2. PUSHING STATE TO BACKUP LOCATIONS

Transfer cache state to remote backup locations.

**Procedure**

- Use the **site** command to push state transfer, as in the following example:

```
//containers/default]> site push-site-state --cache=cacheName --site=NYC
```

For more information and examples, run the **help site** command.

# CHAPTER 8. PATCHING DATA GRID SERVER INSTALLATIONS

Install and manage patches for Data Grid server installations.

You can apply patches to multiple Data Grid servers with different versions to upgrade to a desired target version. However, patches do not take effect if Data Grid servers are running. For this reason you install patches while servers are offline. If you want to upgrade Data Grid clusters without downtime, create a new cluster with the target version and perform a rolling upgrade to that version instead of patching.

## 8.1. DATA GRID SERVER PATCHES

Data Grid server patches are **.zip** archives that contain artifacts that you can apply to your **$RHDG_HOME** directory to fix issues and add new features.

Patches also provide a set of rules for Data Grid to modify your server installation. When you apply patches, Data Grid overwrites some files and removes others, depending on if they are required for the target version.

However, Data Grid does not make any changes to configuration files that you have created or modified when applying a patch. Server patches do not modify or replace any custom configuration or data.

## 8.2. DOWNLOADING SERVER PATCHES

Download patches that you can apply to Data Grid servers.

### Procedure

1. Access the Red Hat customer portal.
2. Download the appropriate Data Grid server patch from the software downloads section.
3. Open a terminal window and navigate to **$RHDG_HOME**.
4. Start the CLI.

```
$ bin/cli.sh
[disconnected]>
```

5. Describe the patch file you downloaded.

```
[disconnected]> patch describe /path/to/redhat-datagrid-$version-server-patch.zip
Red Hat Data Grid patch target=$target_version source=$source_version
created=$timestamp
```

- **$target_version** is the Data Grid version that applies when you install the patch on a server.
- **$source_version** is one or more Data Grid server versions where you can install the patch.

### Verification

Use the checksum to verify the integrity of your download.

1. Run the **md5sum** or **sha256sum** command with the downloaded patch as the argument, for example:

```
$ sha256sum redhat-datagrid-$version-server-patch.zip
```

2. Compare with the **MD5** or **SHA-256** checksum value on the Data Grid **Software Details** page.

## 8.3. CREATING SERVER PATCHES

You can create patches for Data Grid servers from an existing server installation.

You can create patches for Data Grid servers starting from 8.0.1. You can patch 8.0 GA servers with 8.0.1. However you cannot patch 7.3.x or earlier servers with 8.0.1 or later.

You can also create patches that either upgrade or downgrade the Data Grid server version. For example, you can create a patch from version 8.0.1 and use it to upgrade version 8.0 GA or downgrade a later version.

IMPORTANT

Red Hat supports patched server deployments only with patches that you download from the Red Hat customer portal. Red Hat does not support server patches that you create yourself.

**Procedure**

1. Navigate to **$RHDG_HOME** for a Data Grid server installation that has the target version for the patch you want to create.
2. Start the CLI.

```
$ bin/cli.sh
[disconnected]>
```

3. Use the **patch create** command to generate a patch archive and include the **-q** option with a meaningful qualifier to describe the patch.

```
[disconnected]> patch create -q "this is my test patch" path/to/mypatch.zip \
path/to/target/server/home path/to/source/server/home
```

The preceding command generates a **.zip** archive in the specified directory. Paths are relative to **$RHDG_HOME** for the target server.

TIP

Create single patches for multiple different Data Grid versions, for example:

```
[disconnected]> patch create -q "this is my test patch" path/to/mypatch.zip \
path/to/target/server/home \
path/to/source/server1/home path/to/source/server2/home
```

Where **server1** and **server2** are different Data Grid versions where you can install "mypatch.zip".

4. Describe the generated patch archive.

```
[disconnected]> patch describe path/to/mypatch.zip
```

```
Red Hat Data Grid patch target=$target_version(my test patch)  source=$source_version
created=$timestamp
```

- **$target_version** is the Data Grid server version from which the patch was created.
- **$source_version** is one or more Data Grid server versions to which you can apply the patch. You can apply patches to Data Grid servers that match the **$source_version** only. Attempting to apply patches to other versions results in the following exception:

```
java.lang.IllegalStateException: The supplied patch cannot be applied to
`$source_version`
```

## 8.4. INSTALLING SERVER PATCHES

Apply patches to Data Grid servers to upgrade or downgrade an existing version.

### Prerequisites

- Download a server patch for the target version.

### Procedure

1. Navigate to **$RHDG_HOME** for the Data Grid server you want to patch.
2. Stop the server if it is running.

NOTE

If you patch a server while it is running, the version changes take effect after restart. If you do not want to stop the server, create a new cluster with the target version and perform a rolling upgrade to that version instead of patching.

3. Start the CLI.

```
$ bin/cli.sh
[disconnected]>
```

4. Install the patch.

```
[disconnected]> patch install path/to/patch.zip

Red Hat Data Grid patch target=$target_version source=$source_version \
created=$timestamp installed=$timestamp
```

- **$target_version** displays the Data Grid version that the patch installed.
- **$source_version** displays the Data Grid version before you installed the patch.

5. Start the server to verify the patch is installed.

```
$ bin/server.sh
...
ISPN080001: Red Hat Data Grid Server $version
```

If the patch is installed successfully **$version** matches **$target_version**.

TIP

Use the **--server** option to install patches in a different **$RHDG_HOME** directory, for example:

```
[disconnected]> patch install path/to/patch.zip --server=path/to/server/home
```

## 8.5. ROLLING BACK SERVER PATCHES

Remove patches from Data Grid servers by rolling them back and restoring the previous Data Grid version.

IMPORTANT

If a server has multiple patches installed, you can roll back the last installed patch only.

Rolling back patches does not revert configuration changes you make to Data Grid server. Before you roll back patches, you should ensure that your configuration is compatible with the version to which you are rolling back.

**Procedure**

1. Navigate to **$RHDG_HOME** for the Data Grid server installation you want to roll back.
2. Stop the server if it is running.
3. Start the CLI.

```
$ bin/cli.sh
[disconnected]>
```

4. List the installed patches.

```
[disconnected]> patch ls

Red Hat Data Grid patch target=$target_version source=$source_version
created=$timestamp installed=$timestamp
```

   - **$target_version** is the Data Grid server version after the patch was applied.
   - **$source_version** is the version for Data Grid server before the patch was applied. Rolling back the patch restores the server to this version.

5. Roll back the last installed patch.

```
[disconnected]> patch rollback
```

6. Quit the CLI.

```
[disconnected]> quit
```

7. Start the server to verify the patch is rolled back to the previous version.

```
$ bin/server.sh
...
ISPN080001: Data Grid Server $version
```

If the patch is rolled back successfully **$version** matches **$source_version**.

**TIP**

Use the **--server** option to rollback patches in a different **$RHDG_HOME** directory, for example:

```
[disconnected]> patch rollback --server=path/to/server/home
```

# CHAPTER 9. COMMAND REFERENCE

Review manual pages for Data Grid CLI commands.

**TIP**

Use **help** command to access manual pages directly from your CLI session.

For example, to view the manual page for the **get** command do the following:

```
$ help get
```

## 9.1. ADD(1)

### 9.1.1. NAME

add - increments and decrements counters with arbitrary values.

### 9.1.2. SYNOPSIS

**add** ['OPTIONS'] ['COUNTER_NAME']

### 9.1.3. OPTIONS

**--delta='nnn'**

Sets a delta to increment or decrement the counter value. Defaults to **1**.

**-q, --quiet='[true|false]'**

Hides return values for strong counters. The default is **false**.

### 9.1.4. EXAMPLES

**add --delta=10 cnt_a**
Increments the value of **cnt_a** by **10**.

**add --delta=-5 cnt_a**
Decrements the value of **cnt_a** by **5**.

### 9.1.5. SEE ALSO

cas(1), reset(1)

## 9.2. CACHE(1)

### 9.2.1. NAME

cache - selects the default cache for subsequent commands.

### 9.2.2. SYNOPSIS

**cache** ['CACHE_NAME']

### 9.2.3. EXAMPLE

**cache mycache**
Selects **mycache** and is the same as navigating the resource tree using **cd caches/mycache**.

### 9.2.4. SEE ALSO

cd(1), clear(1), container(1), get(1), put(1), remove(1)

## 9.3. CAS(1)

### 9.3.1. NAME

cas - performs 'compare-and-swap' operations on strong counters.

### 9.3.2. SYNOPSIS

**cas** ['OPTIONS'] ['COUNTER_NAME']

### 9.3.3. OPTIONS

**--expect='nnn'**

Specifies the expected value of the counter.

**--value='nnn'**

Sets a new value for the counter.

**-q, --quiet='[true|false]'**

Hides return values. The default is false.

### 9.3.4. EXAMPLE

**cas --expect=10 --value=20 cnt_a**
Sets the value of **cnt_a** to **20** only if the current value is **10**

### 9.3.5. SEE ALSO

add(1), cas(1), reset(1)

## 9.4. CD(1)

### 9.4.1. NAME

cd - navigates the server resource tree.

### 9.4.2. DESCRIPTION

**PATH** can be absolute or relative to the current resource. **../** specifies parent resources.

### 9.4.3. SYNOPSIS

**cd** ['PATH']

### 9.4.4. EXAMPLE

**cd caches**
Changes to the **caches** path in the resource tree.

### 9.4.5. SEE ALSO

cache(1), ls(1), container(1)

## 9.5. CLEARCACHE(1)

### 9.5.1. NAME

clearcache - removes all entries from a cache.

### 9.5.2. SYNOPSIS

**clearcache** ['CACHE_NAME']

### 9.5.3. EXAMPLES

**clearcache mycache**
Removes all entries from **mycache**.

### 9.5.4. SEE ALSO

cache(1), drop(1), remove(1)

## 9.6. CONNECT(1)

### 9.6.1. NAME

connect - connects to running ${infinispan.brand.name} servers.

### 9.6.2. DESCRIPTION

Defaults to **http://localhost:11222** and prompts for credentials if authentication is required.

### 9.6.3. SYNOPSIS

**connect** ['OPTIONS'] ['SERVER_LOCATION']

### 9.6.4. OPTIONS

**-u, --username='USERNAME'**
Specifies a username to authenticate with ${infinispan.brand.name} servers.

**-p, --password='PASSWORD'**
Specifies passwords.

### 9.6.5. EXAMPLE

**connect 127.0.0.1:11322 -u test -p changeme**
Connects to a locally running server using a port offset of **100** and example credentials.

### 9.6.6. SEE ALSO

disconnect(1)

## 9.7. CONTAINER(1)

### 9.7.1. NAME

container - selects the container for running subsequent commands.

### 9.7.2. SYNOPSIS

**container** ['CONTAINER_NAME']

### 9.7.3. EXAMPLE

**container default**
Selects the default container and is the same as navigating the resource tree using **cd containers/default**.

### 9.7.4. SEE ALSO

cd(1), clear(1), container(1), get(1), put(1), remove(1)

## 9.8. COUNTER(1)

### 9.8.1. NAME

counter - selects the default counter for subsequent commands.

### 9.8.2. SYNOPSIS

**counter** ['COUNTER_NAME']

### 9.8.3. EXAMPLE

**counter cnt_a**
Selects **cnt_a** and is the same as navigating the resource tree using **cd counters/cnt_a**.

### 9.8.4. SEE ALSO

add(1), cas(1)

## 9.9. CREATE(1)

### 9.9.1. NAME

create - creates caches and counters on ${infinispan.brand.name} servers.

### 9.9.2. SYNOPSIS

**create cache** ['OPTIONS'] **CACHE_NAME**

**create counter** ['OPTIONS'] **COUNTER_NAME**

### 9.9.3. CREATE CACHE OPTIONS

**-f, --file='FILE'**

Specifies a configuration file in JSON or XML format.

**-t, --template='TEMPLATE'**

Specifies a configuration template. Use tab autocompletion to see available templates.

**-v, --volatile='[true|false]'**

Specifies whether the cache is persistent or volatile. The default is false.

### 9.9.4. CREATE COUNTER OPTIONS

**-t, --type='[weak|strong]'**

Specifies if the counter is weak or strong.

**-s, --storage='[PERSISTENT|VOLATILE]'**

Specifies whether the counter is persistent or volatile.

**-c, --concurrency-level='nnn'**

Sets the concurrency level of the counter.

**-i, --initial-value='nnn'**

Sets the initial value of the counter.

**-l, --lower-bound='nnn'**

Sets the lower bound of a **strong** counter.

**-u, --upper-bound='nnn'**

Sets the upper bound of a **strong** counter.

### 9.9.5. EXAMPLES

**create cache --template=org.infinispan.DIST_SYNC mycache**
Creates a cache named **mycache** from the **DIST_SYNC** template.

**create counter --initial-value=3 --storage=PERSISTENT --type=strong cnt_a**
Creates a strong counter named **cnt_a**.

### 9.9.6. SEE ALSO

drop(1)

## 9.10. DESCRIBE(1)

### 9.10.1. NAME

describe - displays information about resources.

### 9.10.2. SYNOPSIS

**describe** ['PATH']

### 9.10.3. EXAMPLES

**describe //containers/default**
Displays information about the default container.

**describe //containers/default/caches/mycache**
Displays information about the **mycache** cache.

**describe //containers/default/caches/mycache/k1**
Displays information about the **k1** key.

**describe //containers/default/counters/cnt1**
Displays information about the **cnt1** counter.

### 9.10.4. SEE ALSO

cd(1), ls(1)

## 9.11. DISCONNECT(1)

### 9.11.1. NAME

disconnect - ends CLI sessions with ${infinispan.brand.name} servers.

### 9.11.2. SYNOPSIS

**disconnect**

### 9.11.3. EXAMPLE

**disconnect**
Ends the current CLI session.

### 9.11.4. SEE ALSO

connect(1)

## 9.12. DROP(1)

### 9.12.1. NAME

drop - deletes caches and counters.

### 9.12.2. SYNOPSIS

**drop cache CACHE_NAME**

**drop counter COUNTER_NAME**

### 9.12.3. EXAMPLES

**drop cache mycache**
Deletes the **mycache** cache.

**drop counter cnt_a**
Deletes the **cnt_a** counter.

### 9.12.4. SEE ALSO

create(1), clearcache(1)

## 9.13. ENCODING(1)

### 9.13.1. NAME

encoding - displays and sets the encoding for cache entries.

### 9.13.2. DESCRIPTION

Sets a default encoding for **put** and **get** operations on a cache. If no argument is specified, the **encoding** command displays the current encoding.

Valid encodings use standard MIME type (IANA media types) naming conventions, such as the following:

- **text/plain**
- **application/json**
- **application/xml**
- **application/octet-stream**

### 9.13.3. SYNOPSIS

**encoding** ['ENCODING']

### 9.13.4. EXAMPLE

**encoding application/json**
Configures the currently selected cache to encode entries as **application/json**.

### 9.13.5. SEE ALSO

get(1), put(1)

## 9.14. GET(1)

### 9.14.1. NAME

get - retrieves entries from a cache.

### 9.14.2. SYNOPSIS

**get** ['OPTIONS'] **KEY**

### 9.14.3. OPTIONS

**-c, --cache='NAME'**

Specifies the cache from which to retrieve entries. Defaults to the currently selected cache.

### 9.14.4. EXAMPLE

**get hello -c mycache**
Retrieves the value of the key named **hello** from **mycache**.

### 9.14.5. SEE ALSO

query(1), put(1)

## 9.15. HELP(1)

### 9.15.1. NAME

help - prints manual pages for commands.

### 9.15.2. SYNOPSIS

**help** ['COMMAND']

### 9.15.3. EXAMPLE

**help get**
Prints the manual page for the **get** command.

### 9.15.4. SEE ALSO

version(1)

## 9.16. LS(1)

### 9.16.1. NAME

ls - lists resources for the current path or a given path.

### 9.16.2. SYNOPSIS

**ls** ['PATH']

### 9.16.3. EXAMPLES

**ls caches**
Lists the available caches.

**ls ../**
Lists parent resources.

### 9.16.4. SEE ALSO

cd(1)

## 9.17. PATCH(1)

### 9.17.1. NAME

patch - manages server patches.

### 9.17.2. DESCRIPTION

List, describe, install, rollback, and create server patches.

Patches are zip archive files that contain artifacts to upgrade servers and resolve issues or add new features. Patches can apply target versions to multiple server installations with different versions.

### 9.17.3. SYNOPSIS

**patch ls**

**patch install** 'patch-file'

**patch describe** 'patch-file'

**patch rollback**

**patch create** 'patch-file' 'target-server' 'source-server-1' ['source-server-2'...]

### 9.17.4. PATCH LIST OPTIONS

**--server='path/to/server'**
Sets the path to a target server outside the current server home directory.

**-v, --verbose**
Shows the content of each installed patch, including information about individual files.

### 9.17.5. PATCH INSTALL OPTIONS

**--dry-run**
Shows the operations that the patch peforms without applying any changes.

**--server='path/to/server'**
Sets the path to a target server outside the current server home directory.

### 9.17.6. PATCH DESCRIBE OPTIONS

**-v, --verbose**

Shows the content of the patch, including information about individual files

### 9.17.7. PATCH ROLLBACK OPTIONS

**--dry-run**

Shows the operations that the patch peforms without applying any changes.

**--server='path/to/server'**

Sets the path to a target server outside the current server home directory.

### 9.17.8. PATCH CREATE OPTIONS

**-q, --qualifier='name'**

Specifies a descriptive qualifier string for the patch; for example, 'one-off for issue nnnn'.

### 9.17.9. EXAMPLES

**patch ls**
Lists the patches currently installed on a server in order of installation.

**patch install mypatch.zip**
Installs "mypatch.zip" on a server in the current directory.

**patch install mypatch.zip --server=/path/to/server/home**
Installs "mypatch.zip" on a server in a different directory.

**patch describe mypatch.zip**
Displays the target version and list of source versions for "mypatch.zip".

**patch create mypatch.zip 'target-server' 'source-server-1' ['source-server-2'...]**
Creates a patch file named "mypatch.zip" that uses the version of the target server and applies to the source server versions.

**patch rollback**
Rolls back the last patch that was applied to a server and restores the previous version.

## 9.18. PUT(1)

### 9.18.1. NAME

put - adds or updates cache entries.

### 9.18.2. DESCRIPTION

Creates entries for new keys. Replaces values for existing keys.

### 9.18.3. SYNOPSIS

**put** ['OPTIONS'] **KEY** [**VALUE**]

### 9.18.4. OPTIONS

**-c, --cache='NAME'**

Specifies the name of the cache. Defaults to the currently selected cache.

**-e, --encoding='ENCODING'**

Sets the media type for the value.

**-f, --file='FILE'**

Specifies a file that contains the value for the entry.

**-l, --ttl='TTL'**

Sets the number of seconds before the entry is automatically deleted (time-to-live). Defaults to the value for **lifespan** in the cache configuration if **0** or not specified. If you set a negative value, the entry is never deleted.

**-i, --max-idle='MAXIDLE'**

Sets the number of seconds that the entry 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. Defaults to the value for **maxIdle** in the cache configuration if **0** or not specified. If you set a negative value, the entry is never deleted.

**-a, --if-absent=[true|false]**

Puts an entry only if it does not exist.

### 9.18.5. EXAMPLES

**put -c mycache hello world**
Adds the **hello** key with a value of **world** to the **mycache** cache.

**put -c mycache -f myfile -i 500 hola**
Adds the **hola** key with the value from the contents of **myfile**. Also sets a maximum idle of **500** seconds.

### 9.18.6. SEE ALSO

get(1), remove(1)

## 9.19. QUERY(1)

### 9.19.1. NAME

query - retrieves entries that match Ickle query strings.

### 9.19.2. SYNOPSIS

**query** ['OPTIONS'] **QUERY_STRING**

### 9.19.3. OPTIONS

**-c, --cache='NAME'**

Specifies the cache to query. Defaults to the currently selected cache.

**--max-results='MAX_RESULTS'**

Sets the number of results to return. The default is **10**.

**-o, --offset='OFFSET'**

Specifies the index of the first result to return. The default is **0**.

**--query-mode='QUERY_MODE'**

Specifies how the server executes the query. Values are **FETCH** and BROADCAST. The default is **FETCH**.

### 9.19.4. EXAMPLES

**query "from org.infinispan.rest.search.entity.Person p where p.gender = 'MALE'"**
Queries the currently selected cache to return entries from a Protobuf **Person** entity where the gender datatype is **MALE**.

### 9.19.5. SEE ALSO

schema(1)

## 9.20. QUIT(1)

### 9.20.1. NAME

quit - exits the command line interface.

### 9.20.2. SYNOPSIS

**quit**

### 9.20.3. EXAMPLE

**quit**
Exits the CLI.

### 9.20.4. SEE ALSO

disconnect(1), shutdown(1)

## 9.21. REMOVE(1)

### 9.21.1. NAME

remove - deletes entries from a cache.

### 9.21.2. SYNOPSIS

**remove KEY** ['OPTIONS']

### 9.21.3. OPTIONS

**--cache='NAME'**

Specifies the cache from which to remove entries. Defaults to the currently selected cache.

### 9.21.4. EXAMPLE

**remove --cache=mycache hola**
Deletes the **hola** entry from the **mycache** cache.

### 9.21.5. SEE ALSO

cache(1), drop(1), clearcache(1)

## 9.22. RESET(1)

### 9.22.1. NAME

reset - restores the initial values of counters.

### 9.22.2. SYNOPSIS

**reset** ['COUNTER_NAME']

### 9.22.3. EXAMPLE

**reset cnt_a**
Resets the **cnt_a** counter.

### 9.22.4. SEE ALSO

add(1), cas(1), drop(1)

## 9.23. SCHEMA(1)

### 9.23.1. NAME

schema - uploads and registers protobuf schemas.

### 9.23.2. SYNOPSIS

**schema** ['OPTIONS'] **SCHEMA_NAME**

### 9.23.3. OPTIONS

**-u, --upload='FILE'**

Uploads a file as a protobuf schema with the given name.

### 9.23.4. EXAMPLE

**schema --upload=person.proto person.proto**
Registers a **person.proto** Protobuf schema.

### 9.23.5. SEE ALSO

query(1)

## 9.24. SHUTDOWN(1)

### 9.24.1. NAME

shutdown - stops individual servers or performs orderly shutdowns for entire clusters.

### 9.24.2. SYNOPSIS

**shutdown server** ['SERVERS']

**shutdown cluster**

### 9.24.3. EXAMPLES

**shutdown server my_server01**
Stops the server with hostname **my_server01**.

**shutdown cluster**
Performs an orderly shutdown of all servers joined to the cluster.

### 9.24.4. SEE ALSO

connect(1), disconnect(1), quit(1)

## 9.25. SITE(1)

### 9.25.1. NAME

site - manages backup locations and performs cross-site replication operations.

### 9.25.2. SYNOPSIS

**site status** ['OPTIONS']

**site bring-online** ['OPTIONS']

**site take-offline** ['OPTIONS']

**site push-site-state** ['OPTIONS']

**site cancel-push-state** ['OPTIONS']

**site cancel-receive-state** ['OPTIONS']

**site push-site-status** ['OPTIONS']

### 9.25.3. OPTIONS

**--cache='CACHE_NAME'**
Specifies a cache.

**--site='SITE_NAME'**
Specifies a backup location.

### 9.25.4. EXAMPLES

**site status --cache=mycache**
Returns the status of all backup locations for **mycache**.

**site status --cache=mycache --site=NYC**
Returns the status of **NYC** for **mycache**.

**site bring-online --cache=mycache --site=NYC**
Brings the site **NYC** online for **mycache**.

**site take-offline --cache=mycache --site=NYC**
Takes the site **NYC** offline for **mycache**.

**site push-site-state --cache=mycache --site=NYC**
Backs up caches to remote backup locations.

**site push-site-status --cache=mycache**
Displays the status of the operation to backup **mycache**.

**site cancel-push-state --cache=mycache --site=NYC**
Cancels the operation to backup **mycache** to **NYC**.

**site cancel-receive-state --cache=mycache --site=NYC**
Cancels the operation to receive state from **NYC**.

**site clear-push-state-status --cache=myCache**
Clears the status of the push state operation for **mycache**.

## 9.26. TASK(1)

### 9.26.1. NAME

task - executes and uploads server-side tasks and scripts

### 9.26.2. SYNOPSIS

**task upload** --file='script' 'TASK_NAME'

**task exec** ['TASK_NAME']

### 9.26.3. EXAMPLES

**task upload --file=hello.js hello**
Uploads a script from a **hello.js** file and names it **hello**.

**task exec @@cache@names**
Runs a task that returns available cache names.

**task exec hello -Pgreetee=world**
Runs a script named **hello** and specifies the **greetee** parameter with a value of **world**.

### 9.26.4. OPTIONS

**-P, --parameters='PARAMETERS'**

Passes parameter values to tasks and scripts.

**-f, --file='FILE'**

Uploads script files with the given names.

### 9.26.5. SEE ALSO

ls(1)

## 9.27. VERSION(1)

### 9.27.1. NAME

version - displays the server version and CLI version.

### 9.27.2. SYNOPSIS

**version**

### 9.27.3. EXAMPLE

**version**
Returns the version for the server and the CLI.

### 9.27.4. SEE ALSO

help(1)