

# Using the Infinispan Command Line Interface

# Table of Contents

1. Getting Started with the Infinispan CLI .....	1
1.1. Adding Infinispan Credentials .....	1
1.2. Connecting to Infinispan Servers .....	1
1.3. Navigating CLI Resources .....	2
1.3.1. CLI Resources .....	3
1.4. Shutting Down Infinispan Servers .....	4
2. Performing Cache Operations with the Infinispan CLI .....	6
2.1. Creating Caches with the Infinispan Command Line Interface (CLI) .....	6
2.1.1. XML Configuration .....	7
2.1.2. JSON Configuration .....	7
2.2. Adding Cache Entries .....	7
2.3. Clearing Caches and Deleting Entries .....	8
2.4. Deleting Caches .....	8
3. Performing Batch Operations .....	10
3.1. Performing Batch Operations with Files .....	10
3.2. Performing Batch Operations Interactively .....	11
4. Working with Counters .....	13
4.1. Creating Counters .....	13
4.2. Adding Deltas to Counters .....	14
5. Querying Caches with Protobuf Metadata .....	16
5.1. Configuring Media Types .....	16
5.2. Registering Protobuf Schemas .....	17
5.3. Querying Caches with Protobuf Schemas .....	18
6. Performing Cross-Site Replication Operations .....	22
6.1. Bringing Backup Locations Offline and Online .....	22
6.2. Pushing State to Backup Locations .....	22
7. Command Reference .....	24
7.1. ADD(1) .....	24
7.1.1. NAME .....	24
7.1.2. SYNOPSIS .....	24
7.1.3. OPTIONS .....	24
7.1.4. EXAMPLES .....	24
7.1.5. SEE ALSO .....	24
7.2. CACHE(1) .....	24
7.2.1. NAME .....	24
7.2.2. SYNOPSIS .....	25
7.2.3. EXAMPLE .....	25
7.2.4. SEE ALSO .....	25

7.3. CAS(1)	25
7.3.1. NAME	25
7.3.2. SYNOPSIS	25
7.3.3. OPTIONS	25
7.3.4. EXAMPLE	25
7.3.5. SEE ALSO	25
7.4. CD(1)	25
7.4.1. NAME	26
7.4.2. DESCRIPTION	26
7.4.3. SYNOPSIS	26
7.4.4. EXAMPLE	26
7.4.5. SEE ALSO	26
7.5. CLEARCACHE(1)	26
7.5.1. NAME	26
7.5.2. SYNOPSIS	26
7.5.3. EXAMPLES	26
7.5.4. SEE ALSO	26
7.6. CONNECT(1)	26
7.6.1. NAME	26
7.6.2. DESCRIPTION	27
7.6.3. SYNOPSIS	27
7.6.4. OPTIONS	27
7.6.5. EXAMPLE	27
7.6.6. SEE ALSO	27
7.7. CONTAINER(1)	27
7.7.1. NAME	27
7.7.2. SYNOPSIS	27
7.7.3. EXAMPLE	27
7.7.4. SEE ALSO	27
7.8. COUNTER(1)	27
7.8.1. NAME	28
7.8.2. SYNOPSIS	28
7.8.3. EXAMPLE	28
7.8.4. SEE ALSO	28
7.9. CREATE(1)	28
7.9.1. NAME	28
7.9.2. SYNOPSIS	28
7.9.3. CREATE CACHE OPTIONS	28
7.9.4. CREATE COUNTER OPTIONS	28
7.9.5. EXAMPLES	29
7.9.6. SEE ALSO	29

7.10. DESCRIBE(1) .....	29
7.10.1. NAME .....	29
7.10.2. SYNOPSIS .....	29
7.10.3. EXAMPLES .....	29
7.10.4. SEE ALSO .....	30
7.11. DISCONNECT(1) .....	30
7.11.1. NAME .....	30
7.11.2. SYNOPSIS .....	30
7.11.3. EXAMPLE .....	30
7.11.4. SEE ALSO .....	30
7.12. DROP(1) .....	30
7.12.1. NAME .....	30
7.12.2. SYNOPSIS .....	30
7.12.3. EXAMPLES .....	30
7.12.4. SEE ALSO .....	30
7.13. ENCODING(1) .....	31
7.13.1. NAME .....	31
7.13.2. DESCRIPTION .....	31
7.13.3. SYNOPSIS .....	31
7.13.4. EXAMPLE .....	31
7.13.5. SEE ALSO .....	31
7.14. GET(1) .....	31
7.14.1. NAME .....	31
7.14.2. SYNOPSIS .....	31
7.14.3. OPTIONS .....	31
7.14.4. EXAMPLE .....	32
7.14.5. SEE ALSO .....	32
7.15. HELP(1) .....	32
7.15.1. NAME .....	32
7.15.2. SYNOPSIS .....	32
7.15.3. EXAMPLE .....	32
7.15.4. SEE ALSO .....	32
7.16. LS(1) .....	32
7.16.1. NAME .....	32
7.16.2. SYNOPSIS .....	32
7.16.3. EXAMPLES .....	32
7.16.4. SEE ALSO .....	33
7.17. PATCH(1) .....	33
7.17.1. NAME .....	33
7.17.2. DESCRIPTION .....	33
7.17.3. SYNOPSIS .....	33

7.17.4. PATCH LIST OPTIONS .....	33
7.17.5. PATCH INSTALL OPTIONS .....	33
7.17.6. PATCH DESCRIBE OPTIONS .....	33
7.17.7. PATCH ROLLBACK OPTIONS .....	34
7.17.8. PATCH CREATE OPTIONS .....	34
7.17.9. EXAMPLES .....	34
7.18. PUT(1) .....	34
7.18.1. NAME .....	34
7.18.2. DESCRIPTION .....	34
7.18.3. SYNOPSIS .....	34
7.18.4. OPTIONS .....	35
7.18.5. EXAMPLES .....	35
7.18.6. SEE ALSO .....	35
7.19. QUERY(1) .....	35
7.19.1. NAME .....	35
7.19.2. SYNOPSIS .....	35
7.19.3. OPTIONS .....	36
7.19.4. EXAMPLES .....	36
7.19.5. SEE ALSO .....	36
7.20. QUIT(1) .....	36
7.20.1. NAME .....	36
7.20.2. SYNOPSIS .....	36
7.20.3. EXAMPLE .....	36
7.20.4. SEE ALSO .....	36
7.21. REMOVE(1) .....	36
7.21.1. NAME .....	37
7.21.2. SYNOPSIS .....	37
7.21.3. OPTIONS .....	37
7.21.4. EXAMPLE .....	37
7.21.5. SEE ALSO .....	37
7.22. RESET(1) .....	37
7.22.1. NAME .....	37
7.22.2. SYNOPSIS .....	37
7.22.3. EXAMPLE .....	37
7.22.4. SEE ALSO .....	37
7.23. SCHEMA(1) .....	37
7.23.1. NAME .....	37
7.23.2. SYNOPSIS .....	38
7.23.3. OPTIONS .....	38
7.23.4. EXAMPLE .....	38
7.23.5. SEE ALSO .....	38

7.24. SHUTDOWN(1) .....	38
7.24.1. NAME .....	38
7.24.2. SYNOPSIS .....	38
7.24.3. EXAMPLES .....	38
7.24.4. SEE ALSO .....	38
7.25. SITE(1) .....	38
7.25.1. NAME .....	38
7.25.2. SYNOPSIS .....	39
7.25.3. OPTIONS .....	39
7.25.4. EXAMPLES .....	39
7.26. TASK(1) .....	40
7.26.1. NAME .....	40
7.26.2. SYNOPSIS .....	40
7.26.3. EXAMPLES .....	40
7.26.4. OPTIONS .....	40
7.26.5. SEE ALSO .....	40
7.27. VERSION(1) .....	40
7.27.1. NAME .....	40
7.27.2. SYNOPSIS .....	40
7.27.3. EXAMPLE .....	41
7.27.4. SEE ALSO .....	41

# Chapter 1. Getting Started with the Infinispan CLI

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

## 1.1. Adding Infinispan Credentials

Infinispan Server provides a default property realm that restricts access to authenticated users only. Use the Infinispan CLI to add credentials.

### *Procedure*

1. Open a terminal in `$ISP_HOME`.
2. Define credentials with the `user` command as in the following examples:
  - Create a new user named "myuser" and specify a password:

#### **Linux**

```
$ bin/cli.sh user create myuser -p "qwer1234!"
```

#### **Microsoft Windows**

```
$ bin\cli.bat user create myuser -p "qwer1234!"
```

- Create a new user that belongs to the "supervisor", "reader", and "writer" groups if you use security authorization:

#### **Linux**

```
$ bin/cli.sh user create myuser -p "qwer1234!" -g supervisor,reader,writer
```

#### **Microsoft Windows**

```
$ bin\cli.bat user create myuser -p "qwer1234!" -g supervisor,reader,writer
```

## 1.2. Connecting to Infinispan Servers

Establish CLI connections to Infinispan.

### *Prerequisites*

Add user credentials and have at least one running Infinispan server instance.

### *Procedure*

1. Open a terminal in `$ISP_HOME`.

2. Start the CLI.

- **Linux:**

```
$ bin/cli.sh
```

- **Microsoft Windows:**

```
$ bin\cli.bat
```

3. Run the **connect** command and enter your username and password when prompted.

- Infinispan Server on the default port of **11222**:

```
[disconnected]> connect
```

- Infinispan Server with a port offset of **100**:

```
[disconnected]> connect 127.0.0.1:11322
```

## 1.3. Navigating CLI Resources

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



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

When you connect to a Infinispan 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
  } ]
}
```

### 1.3.1. CLI Resources

The Infinispan CLI exposes different resources to:

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

#### *Cache Resources*

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

## caches

Infinispan 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

Infinispan configurations.

## schemas

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

## tasks

Remote tasks creating and managing Infinispan cache definitions.

### Cluster Resources

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

## containers

Cache containers on the Infinispan cluster.

## cluster

Lists Infinispan servers joined to the cluster.

## server

Resources for managing and monitoring Infinispan servers.

# 1.4. Shutting Down Infinispan Servers

Gracefully shut down running Infinispan servers to passivate all entries to disk and persist state.

### Procedure

1. Create a CLI connection to Infinispan.
2. Do one of the following:
  - Stop individual servers with the `shutdown server` command:

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

- Stop all nodes in the cluster with the `shutdown cluster` command:

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

### *Verification*

Check the server logs for the following messages:

```
ISPN080002: Infinispan Server stopping
ISPN000080: Disconnecting JGroups channel cluster
ISPN000390: Persisted state, version=<$version> timestamp=YYYY-MM-DDTHH:MM:SS
ISPN080003: Infinispan Server stopped
```

# Chapter 2. Performing Cache Operations with the Infinispan CLI

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

## 2.1. Creating Caches with the Infinispan Command Line Interface (CLI)

Use the Infinispan CLI to add caches from templates or configuration files in XML or JSON format.

### Prerequisites

Add Infinispan credentials and start at least one Infinispan server instance.

### Procedure

1. Create a CLI connection to Infinispan.
2. Add cache definitions with the `create cache` command.
  - Add a cache definition from an XML or JSON file with the `--file` option.

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

- Add a cache definition from a template with the `--template` option.

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



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

3. Verify the cache exists with the `ls` command.

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

4. Retrieve the cache configuration with the `describe` command.

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

### Reference

- [Creating Infinispan CLI Connections](#)
- [Performing Cache Operations with the Infinispan CLI](#)

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

### 2.1.2. 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": {
    "name": "myCache",
    "mode": "SYNC",
    "encoding": {
      "media-type": "application/x-protostream"
    },
    "memory": {
      "max-count": 1000000,
      "when-full": "REMOVE"
    }
  }
}
```

## 2.2. Adding Cache Entries

Create **key:value** pair entries in the data container.

### Prerequisites

Create a Infinispan cache that can store your data.

### Procedure

1. Create a CLI connection to Infinispan.
2. Add entries into your cache as follows:
  - Use the **put** command from the context of a cache:

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

- Use the **--cache=** with the **put** command:

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

3. Use the **get** command to verify entries.

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

## 2.3. Clearing Caches and Deleting Entries

Remove data from caches with the Infinispan CLI.

### Procedure

1. Create a CLI connection to Infinispan.
2. Do one of the following:
  - Delete all entries with the **clearcache** command.

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

- Remove specific entries with the **remove** command.

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

## 2.4. Deleting Caches

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

### Procedure

1. Create a CLI connection to Infinispan.

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

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

# Chapter 3. Performing Batch Operations

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

## Prerequisites

- A running Infinispan cluster.

## 3.1. Performing Batch Operations with Files

Create files that contain a set of operations and then pass them to the Infinispan 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. Create a new Infinispan CLI connection and verify `mybatch`.

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



CLI batch files support system property expansion. Strings that use the `${property}` format are replaced with the value of the `property` system property.



## 3.2. Performing Batch Operations Interactively

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

### Procedure

1. Start the Infinispan CLI in interactive mode.

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



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
```



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 4. Working with Counters

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

## Prerequisites

- Start the Infinispan CLI.
- Connect to a running Infinispan cluster.

## 4.1. Creating Counters

Create strong and weak counters with the Infinispan CLI.

### Procedure

1. Create a CLI connection to Infinispan.
2. Run the `create counter` command 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
```

3. List available counters.

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

4. 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
  }
}
```

## 4.2. 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
```



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 5. Querying Caches with Protobuf Metadata

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

## Prerequisites

- Start the Infinispan CLI.
- Connect to a running Infinispan cluster.

## 5.1. 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 Infinispan 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"
}

```

## 5.2. 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

```

## 5.3. Querying Caches with Protobuf Schemas

Infinispan 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 6. Performing Cross-Site Replication Operations

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

## Prerequisites

- Start the Infinispan CLI.
- Connect to a running Infinispan cluster.

## 6.1. Bringing Backup Locations Offline and Online

Take backup locations offline manually and bring them back online.

### Procedure

1. Create a CLI connection to Infinispan.
2. Check if backup locations are online or offline with the `site status` command:

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



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

3. Manage backup locations as follows:
  - 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.

## 6.2. Pushing State to Backup Locations

Transfer cache state to remote backup locations.

### Procedure

1. Create a CLI connection to Infinispan.
2. 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 7. Command Reference

Review manual pages for Infinispan CLI commands.



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
```

## 7.1. ADD(1)

### 7.1.1. NAME

`add` - increments and decrements counters with arbitrary values.

### 7.1.2. SYNOPSIS

`add` ['OPTIONS'] ['COUNTER\_NAME']

### 7.1.3. 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`.

### 7.1.4. 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`.

### 7.1.5. SEE ALSO

`cas(1)`, `reset(1)`

## 7.2. CACHE(1)

### 7.2.1. NAME

`cache` - selects the default cache for subsequent commands.

### 7.2.2. SYNOPSIS

**cache** ['CACHE\_NAME']

### 7.2.3. EXAMPLE

**cache mycache**

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

### 7.2.4. SEE ALSO

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

## 7.3. CAS(1)

### 7.3.1. NAME

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

### 7.3.2. SYNOPSIS

**cas** ['OPTIONS'] ['COUNTER\_NAME']

### 7.3.3. 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.

### 7.3.4. EXAMPLE

**cas --expect=10 --value=20 cnt\_a**

Sets the value of **cnt\_a** to **20** only if the current value is **10**

### 7.3.5. SEE ALSO

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

## 7.4. CD(1)

### 7.4.1. NAME

`cd` - navigates the server resource tree.

### 7.4.2. DESCRIPTION

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

### 7.4.3. SYNOPSIS

`cd` [`PATH`]

### 7.4.4. EXAMPLE

`cd caches`

Changes to the `caches` path in the resource tree.

### 7.4.5. SEE ALSO

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

## 7.5. CLEARCACHE(1)

### 7.5.1. NAME

`clearcache` - removes all entries from a cache.

### 7.5.2. SYNOPSIS

`clearcache` [`CACHE_NAME`]

### 7.5.3. EXAMPLES

`clearcache mycache`

Removes all entries from `mycache`.

### 7.5.4. SEE ALSO

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

## 7.6. CONNECT(1)

### 7.6.1. NAME

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



## 7.6.2. DESCRIPTION

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

## 7.6.3. SYNOPSIS

**connect** ['OPTIONS'] ['SERVER\_LOCATION']

## 7.6.4. OPTIONS

**-u, --username='USERNAME'**

Specifies a username to authenticate with \${infinispan.brand.name} servers.

**-p, --password='PASSWORD'**

Specifies passwords.

## 7.6.5. 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.

## 7.6.6. SEE ALSO

disconnect(1)

# 7.7. CONTAINER(1)

## 7.7.1. NAME

container - selects the container for running subsequent commands.

## 7.7.2. SYNOPSIS

**container** ['CONTAINER\_NAME']

## 7.7.3. EXAMPLE

```
container default
```

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

## 7.7.4. SEE ALSO

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

# 7.8. COUNTER(1)

### 7.8.1. NAME

counter - selects the default counter for subsequent commands.

### 7.8.2. SYNOPSIS

**counter** ['COUNTER\_NAME']

### 7.8.3. EXAMPLE

`counter cnt_a`

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

### 7.8.4. SEE ALSO

`add(1)`, `cas(1)`

## 7.9. CREATE(1)

### 7.9.1. NAME

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

### 7.9.2. SYNOPSIS

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

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

### 7.9.3. 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.

### 7.9.4. 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.

## 7.9.5. 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`.

## 7.9.6. SEE ALSO

`drop(1)`

# 7.10. DESCRIBE(1)

## 7.10.1. NAME

`describe` - displays information about resources.

## 7.10.2. SYNOPSIS

**describe** [`PATH`]

## 7.10.3. 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.

#### 7.10.4. SEE ALSO

cd(1), ls(1)

## 7.11. DISCONNECT(1)

### 7.11.1. NAME

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

### 7.11.2. SYNOPSIS

**disconnect**

### 7.11.3. EXAMPLE

**disconnect**

Ends the current CLI session.

### 7.11.4. SEE ALSO

connect(1)

## 7.12. DROP(1)

### 7.12.1. NAME

drop - deletes caches and counters.

### 7.12.2. SYNOPSIS

**drop cache** **CACHE\_NAME**

**drop counter** **COUNTER\_NAME**

### 7.12.3. EXAMPLES

**drop cache mycache**

Deletes the **mycache** cache.

**drop counter cnt\_a**

Deletes the **cnt\_a** counter.

### 7.12.4. SEE ALSO

create(1), clearcache(1)

## 7.13. ENCODING(1)

### 7.13.1. NAME

encoding - displays and sets the encoding for cache entries.

### 7.13.2. 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`

### 7.13.3. SYNOPSIS

**encoding** ['ENCODING']

### 7.13.4. EXAMPLE

`encoding application/json`

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

### 7.13.5. SEE ALSO

`get(1)`, `put(1)`

## 7.14. GET(1)

### 7.14.1. NAME

get - retrieves entries from a cache.

### 7.14.2. SYNOPSIS

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

### 7.14.3. OPTIONS

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

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

#### 7.14.4. EXAMPLE

`get hello -c mycache`

Retrieves the value of the key named `hello` from `mycache`.

#### 7.14.5. SEE ALSO

`query(1)`, `put(1)`

### 7.15. HELP(1)

#### 7.15.1. NAME

`help` - prints manual pages for commands.

#### 7.15.2. SYNOPSIS

`help` [`COMMAND`]

#### 7.15.3. EXAMPLE

`help get`

Prints the manual page for the `get` command.

#### 7.15.4. SEE ALSO

`version(1)`

### 7.16. LS(1)

#### 7.16.1. NAME

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

#### 7.16.2. SYNOPSIS

`ls` [`PATH`]

#### 7.16.3. EXAMPLES

`ls caches`

Lists the available caches.

`ls ../`

Lists parent resources.

## 7.16.4. SEE ALSO

cd(1)

## 7.17. PATCH(1)

### 7.17.1. NAME

patch - manages server patches.

### 7.17.2. 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.

### 7.17.3. 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'...]

### 7.17.4. 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.

### 7.17.5. PATCH INSTALL OPTIONS

**--dry-run**

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

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

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

### 7.17.6. PATCH DESCRIBE OPTIONS

**-v, --verbose**

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

### 7.17.7. PATCH ROLLBACK OPTIONS

#### **--dry-run**

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

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

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

### 7.17.8. PATCH CREATE OPTIONS

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

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

### 7.17.9. 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.

## 7.18. PUT(1)

### 7.18.1. NAME

put - adds or updates cache entries.

### 7.18.2. DESCRIPTION

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

### 7.18.3. SYNOPSIS

`put ['OPTIONS'] KEY [VALUE]`



## 7.18.4. 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.

## 7.18.5. 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.

## 7.18.6. SEE ALSO

`get(1)`, `remove(1)`

# 7.19. QUERY(1)

## 7.19.1. NAME

`query` - retrieves entries that match Ickle query strings.

## 7.19.2. SYNOPSIS

```
query ['OPTIONS'] QUERY_STRING
```

### 7.19.3. 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**.

### 7.19.4. 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**.

### 7.19.5. SEE ALSO

schema(1)

## 7.20. QUIT(1)

### 7.20.1. NAME

quit - exits the command line interface.

### 7.20.2. SYNOPSIS

**quit**

### 7.20.3. EXAMPLE

```
quit
```

Exits the CLI.

### 7.20.4. SEE ALSO

disconnect(1), shutdown(1)

## 7.21. REMOVE(1)

### 7.21.1. NAME

remove - deletes entries from a cache.

### 7.21.2. SYNOPSIS

**remove** **KEY** ['OPTIONS']

### 7.21.3. OPTIONS

**--cache='NAME'**

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

### 7.21.4. EXAMPLE

**remove --cache=mycache hola**

Deletes the **hola** entry from the **mycache** cache.

### 7.21.5. SEE ALSO

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

## 7.22. RESET(1)

### 7.22.1. NAME

reset - restores the initial values of counters.

### 7.22.2. SYNOPSIS

**reset** ['COUNTER\_NAME']

### 7.22.3. EXAMPLE

**reset cnt\_a**

Resets the **cnt\_a** counter.

### 7.22.4. SEE ALSO

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

## 7.23. SCHEMA(1)

### 7.23.1. NAME

schema - uploads and registers protobuf schemas.

## 7.23.2. SYNOPSIS

**schema** ['OPTIONS'] **SCHEMA\_NAME**

## 7.23.3. OPTIONS

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

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

## 7.23.4. EXAMPLE

**schema --upload=person.proto person.proto**

Registers a **person.proto** Protobuf schema.

## 7.23.5. SEE ALSO

query(1)

# 7.24. SHUTDOWN(1)

## 7.24.1. NAME

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

## 7.24.2. SYNOPSIS

**shutdown server** ['SERVERS']

**shutdown cluster**

## 7.24.3. 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.

## 7.24.4. SEE ALSO

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

# 7.25. SITE(1)

## 7.25.1. NAME

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

## 7.25.2. 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']

## 7.25.3. OPTIONS

**--cache='CACHE\_NAME'**

Specifies a cache.

**--site='SITE\_NAME'**

Specifies a backup location.

## 7.25.4. 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`.

## 7.26. TASK(1)

### 7.26.1. NAME

`task` - executes and uploads server-side tasks and scripts

### 7.26.2. SYNOPSIS

**task upload** --file='script' 'TASK\_NAME'

**task exec** ['TASK\_NAME']

### 7.26.3. 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`.

### 7.26.4. OPTIONS

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

Passes parameter values to tasks and scripts.

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

Uploads script files with the given names.

### 7.26.5. SEE ALSO

`ls(1)`

## 7.27. VERSION(1)

### 7.27.1. NAME

`version` - displays the server version and CLI version.

### 7.27.2. SYNOPSIS

**version**

### 7.27.3. EXAMPLE

`version`

Returns the version for the server and the CLI.

### 7.27.4. SEE ALSO

`help(1)`