

Key/Value Store Walkthrough



- [Prerequisite: enabling JetStream](#)
 - [Creating a KV bucket](#)
 - [Storing a value](#)
 - [Getting a value](#)
 - [Deleting a value](#)
 - [Atomic operations](#)
 - [Create \(aka exclusive locking\)](#)
 - [Update with CAS \(aka optimistic locking\)](#)
 - [Watching a K/V Store](#)
 - [Cleaning up](#)
-

The Key/Value Store is a JetStream feature, so we need to verify it is enabled by

```
nats account info
```

which may return

```
JetStream Account Information:
```

```
JetStream is not supported in this account
```

In this case, you should enable JetStream.

Prerequisite: enabling JetStream

If you are running a local `nats-server` stop it and restart it with JetStream enabled using `nats-server -js` (if that's not already done)

You can then check that JetStream is enabled by using

```
nats account info
```

Connection Information:

```
Client ID: 6
Client IP: 127.0.0.1
RTT: 64.996µs
Headers Supported: true
Maximum Payload: 1.0 MiB
Connected URL: nats://127.0.0.1:4222
Connected Address: 127.0.0.1:4222
Connected Server ID: ND2XVDA4Q363J0IFKJTPZW3ZKZCANH7NJI4EJMFSSPTRXDBFG4M4C34K
```

JetStream Account Information:

```
Memory: 0 B of Unlimited
Storage: 0 B of Unlimited
Streams: 0 of Unlimited
Consumers: 0 of Unlimited
```

Creating a KV bucket

A 'KV bucket' is like a stream; you need to create it before using it, as in `nats kv add <KV Bucket Name>`:

```
nats kv add my-kv
```

my_kv Key-Value Store Status

```
Bucket Name: my-kv
History Kept: 1
Values Stored: 0
Compressed: false
Backing Store Kind: JetStream
Bucket Size: 0 B
Maximum Bucket Size: unlimited
```

Maximum Value Size: unlimited

Maximum Age: unlimited

JetStream Stream: KV_my-kv

Storage: File

Storing a value

Now that we have a bucket, we can assign, or 'put', a value to a specific key:

```
nats kv put my-kv Key1 Value1
```

which should return the key's value `Value1`

Getting a value

We can fetch, or 'get', the value for a key "Key1":

```
nats kv get my-kv Key1
```

```
my-kv > Key1 created @ 12 Oct 21 20:08 UTC
```

```
Value1
```

Deleting a value

You can always delete a key and its value by using

```
nats kv del my-kv Key1
```

It is harmless to delete a non-existent key (check this!!).

Atomic operations

K/V Stores can also be used in concurrent design patterns, such as semaphores, by using atomic 'create' and 'update' operations.

E.g. a client wanting exclusive use of a file can lock it by creating a key, whose value is the file name, with `create` and deleting this key after completing use of that file. A client can increase the resilience against failure by using a timeout for the `bucket` containing this key. The client can use `update` with a revision number to keep the `bucket` alive.

Updates can also be used for more fine-grained concurrency control, sometimes known as `optimistic locking`, where multiple clients can try a task, but only one can successfully complete it.

Create (aka exclusive locking)

Create a lock/semaphore with the `create` operation.

```
nats kv create my-sem Semaphore1 Value1
```

Only one `create` can succeed. First come, first serve. All concurrent attempts will result in an error until the key is deleted

```
nats kv create my-sem Semaphore1 Value1
nats: error: nats: wrong last sequence: 1: key exists
```

Update with CAS (aka optimistic locking)

We can also atomically `update`, sometimes known as a CAS (compare and swap) operation, a key with an additional parameter `revision`

```
nats kv update my-sem Semaphore1 Value2 13
```

A second attempt with the same revision 13, will fail

```
nats kv update my-sem Semaphore1 Value2 13
nats: error: nats: wrong last sequence: 14
```

Watching a K/V Store

An unusual functionality of a K/V Store is being able to 'watch' a bucket, or a specific key in that bucket, and receive real-time updates to changes in the store.

For the example above, run `nats kv watch my-kv`. This will start a watcher on the bucket we have just created earlier. By default, the KV bucket has a history size of one, and so it only remembers the last change. In our case, the watcher should see a delete of the value associated with the key "Key1":

```
nats kv watch my-kv
```

```
[2021-10-12 13:15:03] DEL my-kv > Key1
```

If we now concurrently change the value of 'my-kv' by

```
nats kv put my-kv Key1 Value2
```

The watcher will see that change:

```
[2021-10-12 13:25:14] PUT my-kv > Key1: Value2
```

Cleaning up

When you are finished using a bucket, you can delete the bucket, and its resources, by using the `rm` operator:

```
nats kv rm my-kv
```

```
## #1
❑ cocobook-sync ## 2026-10-28 11:10:30 UTC
❑ cocobook-sync ## 2026-10-28 11:10:30 UTC
```