Datomic Cloud Documentation

Search

- Home ›
- ›
- Client API
- Support
- Forum

What is Datomic?

- Data Model
- Architecture
- Supported Operations
- Programming with Data and EDN

Local Dev and CI

Cloud Setup

- Start a System
- Configure Access
- Get Connected

Tutorial

- Client API
- Assertion
- Read
- Accumulate
- Read Revisited
- Retract
- History

Client API

Videos

- AWS Setup
- Edn
- Datalog
- Datoms
- HTTP Direct
- CLI tools

Hide All Examples

Client API

The Client API is the way your code accesses Datomic. The API is available in both synchronous and asynchronous forms.

You create a Datomic client object and use it to list, create and delete databases, and to obtain a connection to a specific database.

Given a connection, you can transact new data or obtain the db as a value, from which you can query with q or pull specific entities.

This page covers everything you need to use the Datomic Client APIs:

- How to install the client library.

- An overview of the two API flavors: synchronous and asynchronous.
- Key API objects: clients and connections.
- Key API *concepts*: error handling, timeouts, chunking, and offset/limit.

Day of Datomic Cloud gives an overview of the Client API.

Installing the Client Library

The Datomic Client library includes both the synchronous and asynchronous APIs, and is provided via Maven Central.

Clojure CLI

To use the Client library from a Clojure CLI REPL, add the following to your *deps.edn* dependencies map:

```
com.datomic/client-cloud {:mvn/version "0.8.102"}
```

Maven

To retrieve the Client library for a Maven project, add the following snippet inside the <dependencies> block of your pom.xml file:

```
<dependency>
 <groupId>com.datomic</groupId>
 <artifactId>client-cloud</artifactId>
 <version>0.8.102</version>
</dependency>
```

Leiningen

To include the Client library in a Leiningen project, add the following snippet to your *project.clj* file in the collection under :dependencies key.

```
com.datomic/client-cloud {:mvn/version "0.8.102"}
```

Make sure that Clojure dependency is set to at least `[org.clojure/clojure "1.9.0"]`.

Synchronous API

Synchronous API functions have the following common semantics:

- they block the calling thread if they access a remote resource
- they return a value
- they indicate anomalies by throwing an exception

The following example shows a simple transaction using the Synchronous API.

```
;; load the Synchronous API with prefix 'd'
(require '[datomic.client.api :as d])
```

```
;; transact a movie
(def result (d/transact conn {:tx-data [{:db/id "goonies"
                                         :movie/title "The Goonies"
                                         :movie/genre "action/adventure"
                                         :movie/release-year 1985}]}))

;; what id was assigned to The Goonies?
(-> result :tempids (get "goonies"))
```

⇧

Asynchronous API

The Asynchronous API functions differ from those in the Synchronous API in that

- they return a core.async channel if they access a remote resource
- they never block the calling thread
- they place result(s) on the channel
- they place anomalies on the channel

You must use a channel-taking operator such as <!! to retrieve the result of an Asynchronous operation, and you must explicitly check for anomalies:

```
;; load the Asynchronous API with prefix 'd', plus core.async and anomaly APIs
(require '[clojure.core.async :refer (<!!)]
         '[cognitect.anomalies :as anom]
         '[datomic.client.api.async :as d])

;; transact a movie
(def result (<!! (d/transact conn {:tx-data [{:db/id "goonies"
                                              :movie/title "The Goonies"
                                              :movie/genre "action/adventure"
                                              :movie/release-year 1985}]})))

;; what id was assigned to The Goonies?
(when-not (::anom/anomaly result)
  (-> result :tempids (get "goonies")))
```

⇧

Client Object

All use of the Client API begins by creating a client. The args map for creating a client expects the following keys:

Key	Value
:server-type	:ion OR :cloud
:region	AWS region
:system	your system name

Key	Value
:endpoint	created by CloudFormation, either the system or the query group endpoint
:timeout	msec timeout, optional, default 60000

See the ion server-type documentation for more details on the server-type.

Database Operations

The client object can be used to create, list, and delete databases.

Create and delete operations take effect immediately. After a database is deleted, a Primary Compute Node will asynchronously delete all resources associated with a database.

Connection

All use of a database is through a connection object. The connect API returns a connection, which is then passed as an argument to the transaction and query APIs.

Database values remember their association with a connection, so APIs that work against a single database (e.g. datoms) do not need to redundantly specify a connection argument.

Connections do not require any special handling on the client side: Connections are thread safe, do not need to be pooled, and do not need to be closed when not in use. Connections are cached automatically, so creating the same connection multiple times is inexpensive.

Handling Errors

Errors are represented as an anomalies map. In the sync API, you can retrieve the specific anomaly as the ex-data of an exception. In the async API, the anomaly will be placed on the channel.

::cognitect.anomalies/busy

If you get a *busy* response from the Client API, your request rate has temporarily exceeded the capacity of a node, and has already been through an exponential backoff and retry implemented by the client. At this point you have three options:

- For transactions: continue to retry the request at the application level with your own exponential backoff. The Mbrainz Importer example project demonstrates a batch import with retry.
- For queries: retry the request as long as the returned anomaly is retryable. If you consistently receive *busy* responses, you may wish to expand the capacity of your system, by upgrading from Solo to Production or by adding/scaling a query group.
- Give up on completing the request.

Sync API Error Example

The query below is incorrect because the `:find` clause uses a variable name `?nomen` that is not bound by the query. The sync API will throw an exception:

```
(require '[datomic.client.api :as d])
(d/q '[:find ?nomen
       :where [_ :artist/name ?name]]
     db)
=> ExceptionInfo Query is referencing unbound variables: #{?nomen}
```

⇧

You can discover the category of anomaly by inspecting the `ex-data` of the exception:

```
(ex-data *e)
=> {:cognitect.anomalies/category
    :cognitect.anomalies/incorrect,
    :cognitect.anomalies/message
    "Query is referencing unbound variables: #{?nomen}",
    ...}
```

⇧

Async API Error Example

With the Async API, the same anomaly appears as a map on channel:

```
(require '[datomic.client.api.async :as d])
(<!! (d/q {:query '[:find ?nomen
                    :where [_ :artist/name ?name]]
           :args [db]}))
=> {:cognitect.anomalies/category
    :cognitect.anomalies/incorrect,
    :cognitect.anomalies/message
    "Query is referencing unbound variables: #{?nomen}",
    ...}
```

⇧

Timeouts

All APIs that communicate with a remote process enforce a timeout that you can specify via the args map:

Key	Meaning	Default
:timeout	max msec wait before giving up	60000

The call to create a client takes a `:timeout` which establishes the default for all API calls through that client.

API timeouts cause an anomaly with category `::anom/unavailable`, as shown below:

```
(d/q {:query '[:find (count ?name)
               :where [_ :artist/name ?name]]
      :args [db]
      :timeout 1})
(ex-data *e)
```

```
=> #:cognitect.anomalies{:category :cognitect.anomalies/unavailable,
                         :message "Total timeout elapsed"}
```

⇧

Chunked Results

Client APIs that can return an arbitrary number of results return those results in *chunks*. You can control chunking by adding the `:chunk` keyword to argument maps:

Key	Meaning	Default	Max
:chunk	max results in a single chunk	1000	(unlimited)

The default chunk size delivers good performance, and should only be changed to address a measurable performance need.

- Smaller chunks require more roundtrips and potentially higher latency overall, but require less memory on the client (assuming the client program drops each chunk after use).
- Larger chunks reverse this tradeoff, delivering results in fewer roundtrips but requiring more memory on the client side.

Note that the chunk size is orthogonal to the actual work done by the server:

- Datalog queries always realize the entire result in memory first, and then chunk the result back to the client.
- APIs that pull datoms from indexes (datoms, index-range and tx-range) are lazy and realize chunks one at a time on both the server and client.

When you use an ion client, the "client" and "server" are the same process, and the `:chunk` argument is ignored.

Synchronous API functions are designed for convenience. They return a single collection or iterable and do not expose chunks directly. The chunk size argument is nevertheless available and relevant for performance tuning.

The Asynchronous API provides more flexibility, exposing the chunks directly, one at a time, on a core.async channel. Processing results by async transduction is the most efficient way to deal with large results, and is demonstrated in the example below.

Chunked Results Example

If you wanted to know the average length of an artist name in the mbrainz data set, you could use the following async transduction to process all the names in chunks of 10,000.

```
(defn averager
  "Reducing fn that calculates average of its inputs"
  ([] [0 0])                  ;; init
  ([[n total]]                ;; complete
     (/ total (double n)))
  ([[n total] count]          ;; step
     [(inc n) (+ total count)]))

(defn nth-counter
  "Transform for chunked query results that gets the count of
the nth item from each result tuple."
```

```
  [n]
  (comp (halt-when :anom/category)  ;; fail fast
        cat                         ;; flatten the chunks
        (map #(nth % n))            ;; nth of chunk
        (map count)))

(def datoms-query {:limit -1
                   :chunk 10000
                   :index :aevt
                   :components [:artist/name]})
(->> (async/datoms db datoms-query)
     (a/transduce (nth-counter 2) averager (averager))
     <!!)
=> 13.893066724625081
```

⇧

The decomposition of async programs into named reducing fns and transforms such as `averager` and `nth-counter` makes it easy to test the parts of an async program entirely synchronously, and without the need for mocking and stubbing:

```
;; check that averager actually averages
(transduce identity averager [5 4])
=> 4.5

;; check that nth-counter returns count of nth element in chunk of tuples
(def chunks [[[:person-1 :name "Flintstone"]
              [:person-2 :name "Rubble"]]])
(sequence (nth-counter 2) chunks)
=> (10 6)

;; check that the reducing fn and xform compose
(transduce (nth-counter 2) averager chunks)
=> 8.0
```

⇧

You should add async transductions to your program when you have e.g. a measured performance need. For this simple example, all the code above could be replaced by a simple API query:

```
(d/q '[:find (avg ?ct)
       :with ?artist
       :where [?artist :artist/name ?name]
              [(count ?name) ?ct]]
     db)
=> [[13.893066724625081]]
```

⇧

Offset and Limit

Client APIs that can return an arbitrary number of results allow you to request only part of these results by specifying the following optional keys in the args map:

Key	Meaning	Default

Key	Meaning	Default
:offset	number of results to omit from the beginning	0
:limit	maximum number of results to return	1000

You can specify a `:limit` of `-1` to request all results.

Offset and Limit Example

The full mbrainz example has over 600,000 names. The following query uses `:offset` and `:limit` to return 2 names starting with the 10th item in the query result.

```
;; query
{:query '[:find ?name
          :where [_ :artist/name ?name]]
          :args [db]
          :offset 10
          :limit 2}

;; result
[["Kenneth Ishak & The Freedom Machines"]
 ["The Plastik"]]
```

⇧