# InfluxData | Documentation | Writing Data with the HTTP API

https://docs.influxdata.com/influxdb/v1.2/guides/writing\_data/

Writing Data with the HTTP API

There are many ways to write data into InfluxDB including the [command line interface](https://docs.influxdata.com/influxdb/v1.2/tools/shell/), [client libraries](https://docs.influxdata.com/influxdb/v1.2/clients/api/) and plugins for common data formats such as [Graphite](https://docs.influxdata.com/influxdb/v1.2/write_protocols/graphite/). Here we’ll show you how to create a database and write data to it using the built-in HTTP API.

[Creating a database using the HTTP API](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#creating-a-database-using-the-http-api)

To create a database send a POST request to the /query endpoint and set the URL parameter q to CREATE DATABASE <new\_database\_name>. The example below sends a request to InfluxDB running on localhost and creates the database mydb:

curl -i -XPOST http://localhost:8086/query --data-urlencode "q=CREATE DATABASE mydb"

[Writing data using the HTTP API](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#writing-data-using-the-http-api)

The HTTP API is the primary means of writing data into InfluxDB, by sending POST requests to the /writeendpoint. The example below writes a single point to the mydb database. The data consist of the [measurement](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#measurement)cpu\_load\_short, the [tag keys](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#tag-key) host and region with the [tag values](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#tag-value) server01 and us-west, the [field key](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#field-key)value with a [field value](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#field-value) of 0.64, and the [timestamp](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#timestamp) 1434055562000000000.

curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary 'cpu\_load\_short,host=server01,region=us-west value=0.64 1434055562000000000'

When writing points, you must specify an existing database in the db query parameter. Points will be written todb’s default retention policy if you do not supply a retention policy via the rp query parameter. See the [API Reference](https://docs.influxdata.com/influxdb/v1.2/tools/api/#write) documentation for a complete list of the available query parameters.

The body of the POST - we call this the [Line Protocol](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#line-protocol) - contains the time-series data that you wish to store. They consist of a measurement, tags, fields, and a timestamp. InfluxDB requires a measurement name. Strictly speaking, tags are optional but most series include tags to differentiate data sources and to make querying both easy and efficient. Both tag keys and tag values are strings. Field keys are required and are always strings, and,[by default](https://docs.influxdata.com/influxdb/v1.2/write_protocols/line_protocol_reference/#data-types), field values are floats. The timestamp - supplied at the end of the line in Unix time in nanoseconds since January 1, 1970 UTC - is optional. If you do not specify a timestamp InfluxDB uses the server’s local nanosecond timestamp in Unix epoch. Anything that has to do with time in InfluxDB is always UTC.

[Writing multiple points](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#writing-multiple-points)

Post multiple points to multiple series at the same time by separating each point with a new line. Batching points in this manner results in much higher performance.

The following example writes three points to the database mydb. The first point belongs to the series with the measurement cpu\_load\_short and tag set host=server02 and has the server’s local timestamp. The second point belongs to the series with the measurement cpu\_load\_short and tag set host=server02,region=us-westand has the specified timestamp 1422568543702900257. The third point has the same specified timestamp as the second point, but it is written to the series with the measurement cpu\_load\_short and tag setdirection=in,host=server01,region=us-west.

curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary 'cpu\_load\_short,host=server02 value=0.67

cpu\_load\_short,host=server02,region=us-west value=0.55 1422568543702900257

cpu\_load\_short,direction=in,host=server01,region=us-west value=2.0 1422568543702900257'

[Writing points from a file](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#writing-points-from-a-file)

Write points from a file by passing @filename to curl. The data in the file should follow InfluxDB’s [line protocol syntax](https://docs.influxdata.com/influxdb/v1.2/write_protocols/write_syntax/).

Example of a properly-formatted file (cpu\_data.txt):

cpu\_load\_short,host=server02 value=0.67

cpu\_load\_short,host=server02,region=us-west value=0.55 1422568543702900257

cpu\_load\_short,direction=in,host=server01,region=us-west value=2.0 1422568543702900257

Write the data in cpu\_data.txt to the mydb database with:  
  
curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary @cpu\_data.txt

**Note:** If your data file has more than 5,000 points, it may be necessary to split that file into several files in order to write your data in batches to InfluxDB. By default, the HTTP request times out after five seconds. InfluxDB will still attempt to write the points after that time out but there will be no confirmation that they were successfully written.

[Schemaless Design](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#schemaless-design)

InfluxDB is a schemaless database. You can add new measurements, tags, and fields at any time. Note that if you attempt to write data with a different type than previously used (for example, writing a string to a field that previously accepted integers), InfluxDB will reject those data.

[A note on REST…](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#a-note-on-rest)

InfluxDB uses HTTP solely as a convenient and widely supported data transfer protocol.

Modern web APIs have settled on REST because it addresses a common need. As the number of endpoints grows the need for an organizing system becomes pressing. REST is the industry agreed style for organizing large numbers of endpoints. This consistency is good for those developing and consuming the API: everyone involved knows what to expect.

REST, however, is a convention. InfluxDB makes do with three API endpoints. This simple, easy to understand system uses HTTP as a transfer method for [InfluxQL](https://docs.influxdata.com/influxdb/v1.2/query_language/spec/). The InfluxDB API makes no attempt to be RESTful.

[HTTP response summary](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#http-response-summary)

2xx: If your write request received HTTP 204 No Content, it was a success!

4xx: InfluxDB could not understand the request.

5xx: The system is overloaded or significantly impaired.

**Examples of error responses:**

Writing a float to a field that previously accepted booleans:

curl -i -XPOST 'http://localhost:8086/write?db=hamlet' --data-binary 'tobeornottobe booleanonly=true'

curl -i -XPOST 'http://localhost:8086/write?db=hamlet' --data-binary 'tobeornottobe booleanonly=5'

returns:

HTTP/1.1 400 Bad Request

Content-Type: application/json

Request-Id: [...]

X-Influxdb-Version: 1.2.x

Date: Wed, 01 Mar 2017 19:38:01 GMT

Content-Length: 150

{"error":"field type conflict: input field \"booleanonly\" on measurement \"tobeornottobe\" is type float, already exists as type boolean dropped=1"}

Writing a point to a database that doesn’t exist:

curl -i -XPOST 'http://localhost:8086/write?db=atlantis' --data-binary 'liters value=10'

returns:

HTTP/1.1 404 Not Found

Content-Type: application/json

Request-Id: [...]

X-Influxdb-Version: 1.2.x

Date: Wed, 01 Mar 2017 19:38:35 GMT

Content-Length: 45

{"error":"database not found: \"atlantis\""}

[Next steps](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/#next-steps)

Now that you know how to write data with the built-in HTTP API discover how to query them with the [Querying Data](https://docs.influxdata.com/influxdb/v1.2/guides/querying_data/) guide! For more information about writing data with the HTTP API, please see the [API reference documentation](https://docs.influxdata.com/influxdb/v1.2/tools/api/#write).

# InfluxData | Documentation | API Client Libraries

https://docs.influxdata.com/influxdb/v1.2/tools/api\_client\_libraries/

This is a list of the client libraries which have some support for InfluxDB version 0.9. Most should be fully compatible with InfluxDB version 1.2. Functionality will vary, and there are, as yet, no standard features that all libraries must implement in order to be listed here.

[Go](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#go)

[InfluxDB Go](https://github.com/influxdb/influxdb/blob/master/client/README.md) by [InfluxData](https://github.com/influxdata)

[Haskell](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#haskell)

[InfluxDB Haskell](https://github.com/maoe/influxdb-haskell) by [maoe](https://github.com/maoe)

[Java](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#java)

[InfluxDB Java](https://github.com/influxdb/influxdb-java) by [majst01](https://github.com/majst01)

[JavaScript/Node.js](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#javascript-node-js)

[influx](https://github.com/node-influx/node-influx) by [node-influx](https://github.com/node-influx)

[Lisp](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#lisp)

[cl-influxdb](https://github.com/mmaul/cl-influxdb) by [mmaul](https://github.com/mmaul)

[.Net](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#net)

[InfluxDB.Client.Net](https://github.com/AdysTech/InfluxDB.Client.Net) by [mvadu](https://github.com/mvadu)

[InfluxData.Net](https://github.com/pootzko/InfluxData.Net) by [pootzko](https://github.com/pootzko)

[InfluxDB Client for .NET](https://github.com/MikaelGRA/InfluxDB.Client) by [MikaelGRA](https://github.com/MikaelGRA)

[Perl](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#perl)

[AnyEvent::InfluxDB](https://github.com/ajgb/anyevent-influxdb) by [ajgb](https://github.com/ajgb)

[InfluxDB::LineProtocol](http://search.cpan.org/~domm/InfluxDB-LineProtocol/) by [domm](http://search.cpan.org/~domm/)

[InfluxDB::HTTP](https://github.com/raphaelthomas/InfluxDB-HTTP) by [Raphael Seebacher](https://github.com/raphaelthomas)

[PHP](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#php)

[InfluxDB PHP](https://github.com/influxdb/influxdb-php) by [thecodeassassin](https://github.com/thecodeassassin)

[InfluxDB PHP SDK](https://github.com/corley/influxdb-php-sdk) by [corley](https://github.com/corley)

[Python](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#python)

[InfluxDB Python](https://github.com/influxdb/influxdb-python)

[Ruby](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#ruby)

[InfluxDB Ruby](https://github.com/influxdb/influxdb-ruby)

[Influxer](https://github.com/palkan/influxer) by [palkan](https://github.com/palkan)

[Rust](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#rust)

[Flux](https://crates.io/crates/flux)

[Influent](https://crates.io/crates/influent)

[Scala](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#scala)

[scala-influxdb-client](https://github.com/paulgoldbaum/scala-influxdb-client) by [paulgoldbaum](https://github.com/paulgoldbaum)

[Sensu](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#sensu)

[sensu-influxdb-extension](https://github.com/jhrv/sensu-influxdb-extension) by [jhrv](https://github.com/jhrv)

[SNMP agent](https://docs.influxdata.com/influxdb/v1.2/tools/api_client_libraries/#snmp-agent)

[snmpcollector](https://github.com/toni-moreno/snmpcollector) by [toni-moreno](https://github.com/toni-moreno) A full featured Generic SNMP data collector with Web Administration Interface for InfluxDB.

 This documentation is [open source](https://github.com/influxdata/docs.influxdata.com/blob/master/content/influxdb/v1.2/tools/api_client_libraries.md). See a typo? Please, open an [issue](https://github.com/influxdata/docs.influxdata.com/issues/new).

**Need help getting up and running?** [Get Support](https://influxdata.com/pricing/#product-subscriptions)

# influxdb/README.md

at master · influxdata/influxdb

https://github.com/influxdata/influxdb/blob/master/client/README.md

InfluxDB Client

## Description

**NOTE:** The Go client library now has a "v2" version, with the old version being deprecated. The new version can be imported at simport "github.com/influxdata/influxdb/client/v2". It is not backwards-compatible.

A Go client library written and maintained by the **InfluxDB** team. This package provides convenience functions to read and write time series data. It uses the HTTP protocol to communicate with your **InfluxDB** cluster.

Getting Started

## Connecting To Your Database

Connecting to an **InfluxDB** database is straightforward. You will need a host name, a port and the cluster user credentials if applicable. The default port is 8086. You can customize these settings to your specific installation via the **InfluxDB** configuration file.

Though not necessary for experimentation, you may want to create a new user and authenticate the connection to your database.

For more information please check out the [Admin Docs](https://docs.influxdata.com/influxdb/latest/administration/).

For the impatient, you can create a new admin user bubba by firing off the [InfluxDB CLI](https://github.com/influxdata/influxdb/blob/master/cmd/influx/main.go).

influx

> create user bubba with password 'bumblebeetuna'

> grant all privileges to bubba

And now for good measure set the credentials in you shell environment. In the example below we will use $INFLUX\_USER and $INFLUX\_PWD

Now with the administrivia out of the way, let's connect to our database.

NOTE: If you've opted out of creating a user, you can omit Username and Password in the configuration below.

package main

import (

"log"

"time"

"github.com/influxdata/influxdb/client/v2"

)

const (

MyDB = "square\_holes"

username = "bubba"

password = "bumblebeetuna"

)

func main() {

// Create a new HTTPClient

c, err := client.NewHTTPClient(client.HTTPConfig{

Addr: "http://localhost:8086",

Username: username,

Password: password,

})

if err != nil {

log.Fatal(err)

}

// Create a new point batch

bp, err := client.NewBatchPoints(client.BatchPointsConfig{

Database: MyDB,

Precision: "s",

})

if err != nil {

log.Fatal(err)

}

// Create a point and add to batch

tags := map[string]string{"cpu": "cpu-total"}

fields := map[string]interface{}{

"idle": 10.1,

"system": 53.3,

"user": 46.6,

}

pt, err := client.NewPoint("cpu\_usage", tags, fields, time.Now())

if err != nil {

log.Fatal(err)

}

bp.AddPoint(pt)

// Write the batch

if err := c.Write(bp); err != nil {

log.Fatal(err)

}

}

新建BatchPoints

bp, err := client.NewBatchPoints(client.BatchPointsConfig{  
 Database: *MyDB*,  
 Precision: "s",  
})

新增tags和fields

tags := **map**[string]string{"cpu": "cpu-total"}  
fields := **map**[string]**interface**{}{  
 "idle": 10.1,  
 "system": 53.3,  
 "user": 46.6,  
}

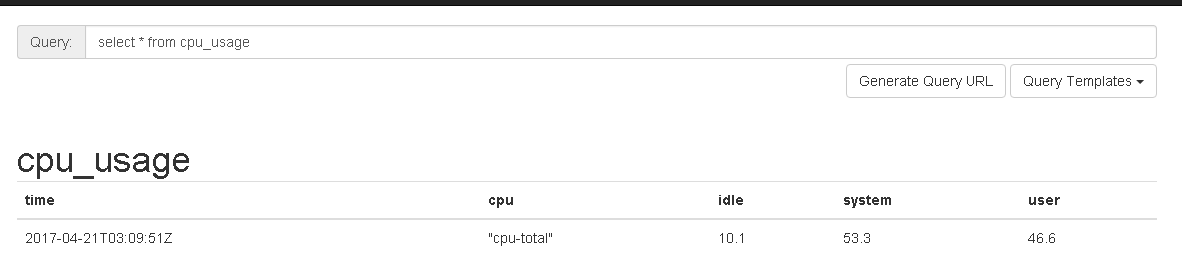
将tags和fields关联起来，新增point

pt, err := client.NewPoint("cpu\_usage", tags, fields, time.Now())

将point保存在batchPoints

bp.AddPoint(pt)

select \* from cpu\_usage



## Inserting Data

Time series data aka points are written to the database using batch inserts. The mechanism is to create one or more points and then create a batch aka batch points and write these to a given database and series. A series is a combination of a measurement (time/values) and a set of tags.

In this sample we will create a batch of a 1,000 points. Each point has a time and a single value as well as 2 tags indicating a shape and color. We write these points to a database called square\_holes using a measurement named shapes.

NOTE: You can specify a RetentionPolicy as part of the batch points. If not provided InfluxDB will use the database defaultretention policy.

func writePoints(clnt client.Client) {

sampleSize := 1000

bp, err := client.NewBatchPoints(client.BatchPointsConfig{

Database: "systemstats",

Precision: "us",

})

if err != nil {

log.Fatal(err)

}

rand.Seed(time.Now().UnixNano())

for i := 0; i < sampleSize; i++ {

regions := []string{"us-west1", "us-west2", "us-west3", "us-east1"}

tags := map[string]string{

"cpu": "cpu-total",

"host": fmt.Sprintf("host%d", rand.Intn(1000)),

"region": regions[rand.Intn(len(regions))],

}

idle := rand.Float64() \* 100.0

fields := map[string]interface{}{

"idle": idle,

"busy": 100.0 - idle,

}

pt, err := client.NewPoint(

"cpu\_usage",

tags,

fields,

time.Now(),

)

if err != nil {

log.Fatal(err)

}

bp.AddPoint(pt)

}

if err := clnt.Write(bp); err != nil {

log.Fatal(err)

}

}

## Querying Data

One nice advantage of using **InfluxDB** the ability to query your data using familiar SQL constructs. In this example we can create a convenience function to query the database as follows:

// queryDB convenience function to query the database

func queryDB(clnt client.Client, cmd string) (res []client.Result, err error) {

q := client.Query{

Command: cmd,

Database: MyDB,

}

if response, err := clnt.Query(q); err == nil {

if response.Error() != nil {

return res, response.Error()

}

res = response.Results

} else {

return res, err

}

return res, nil

}

Creating a Database

\_, err := queryDB(clnt, fmt.Sprintf("CREATE DATABASE %s", MyDB))

if err != nil {

log.Fatal(err)

}

Count Records

q := fmt.Sprintf("SELECT count(%s) FROM %s", "value", MyMeasurement)

res, err := queryDB(clnt, q)

if err != nil {

log.Fatal(err)

}

count := res[0].Series[0].Values[0][1]

log.Printf("Found a total of %v records\n", count)

Find the last 10 shapes records

q := fmt.Sprintf("SELECT \* FROM %s LIMIT %d", MyMeasurement, 20)

res, err = queryDB(clnt, q)

if err != nil {

log.Fatal(err)

}

for i, row := range res[0].Series[0].Values {

t, err := time.Parse(time.RFC3339, row[0].(string))

if err != nil {

log.Fatal(err)

}

val := row[1].(string)

log.Printf("[%2d] %s: %s\n", i, t.Format(time.Stamp), val)

}

## Using the UDP Client

The **InfluxDB** client also supports writing over UDP.

func WriteUDP() {

// Make client

c, err := client.NewUDPClient("localhost:8089")

if err != nil {

panic(err.Error())

}

// Create a new point batch

bp, \_ := client.NewBatchPoints(client.BatchPointsConfig{

Precision: "s",

})

// Create a point and add to batch

tags := map[string]string{"cpu": "cpu-total"}

fields := map[string]interface{}{

"idle": 10.1,

"system": 53.3,

"user": 46.6,

}

pt, err := client.NewPoint("cpu\_usage", tags, fields, time.Now())

if err != nil {

panic(err.Error())

}

bp.AddPoint(pt)

// Write the batch

c.Write(bp)

}

### Point Splitting

The UDP client now supports splitting single points that exceed the configured payload size. The logic for processing each point is listed here, starting with an empty payload.

1. If adding the point to the current (non-empty) payload would exceed the configured size, send the current payload. Otherwise, add it to the current payload.
2. If the point is smaller than the configured size, add it to the payload.
3. If the point has no timestamp, just try to send the entire point as a single UDP payload, and process the next point.
4. Since the point has a timestamp, re-use the existing measurement name, tagset, and timestamp and create multiple new points by splitting up the fields. The per-point length will be kept close to the configured size, staying under it if possible. This does mean that one large field, maybe a long string, could be sent as a larger-than-configured payload.

The above logic attempts to respect configured payload sizes, but not sacrifice any data integrity. Points without a timestamp can't be split, as that may cause fields to have differing timestamps when processed by the server.

## Go Docs

Please refer to <http://godoc.org/github.com/influxdata/influxdb/client/v2> for documentation.

See Also

You can also examine how the client library is used by the [InfluxDB CLI](https://github.com/influxdata/influxdb/blob/master/cmd/influx/main.go).