influxdata®

InfluxDB Open Source Software Onboarding Guide

Chapter 1

Introduction

Welcome to InfluxDB open source (OSS)! InfluxDB is a platform built specifically for handling time series data.

Users typically interact with InfluxDB OSS in two ways.

1. **Storing and querying data**: These users tend to interact extensively with the InfluxDB API.
2. **Administration**: These users tend to be administrators tasked with managing InfluxDB deployments, maintenance, configuration, and scaling.

To learn more about InfluxDB's features and querying capabilities, check out the courses available at InfluxDB University, the complimentary *Time to Awesome* book, and InfluxDB's extensive documentation.

This guide primarily focuses on tasks and processes for **administrators**. It covers processes for installing and configuring a single InfluxDB OSS 2.x node. This includes how to estimate sizing, installation, and configuration. It also goes over how to use Docker with InfluxDB OSS and how to deploy InfluxDB OSS nodes at scale using Ansible. Finally, the guide provides instructions on how to set up several different methods for monitoring InfluxDB OSS.

We anticipate that future versions of this guide will include additional details and instructions about operating InfluxDB OSS as well as more methods for deploying InfluxDB OSS nodes at scale.

We hope you find this guide helpful for getting started with InfluxDB and improving your Time to Awesome!

Resources

- Documentation
- Complementary "how to" book, Time To Awesome
- InfluxDB University courses
- InfluxData YouTube channel
- InfluxData Blog

Chapter 2

Sizing and Optimizations

The optimal size of an InfluxDB instance is dependent on a lot of factors. This chapter covers how you should think about those factors and how to measure and estimate them to get as close to an accurate size prediction as possible.

Three workload factors, write volume, query volume, and storage, affect the sizing of an InfluxDB OSS node. You can estimate write volume and storage needs fairly accurately. However, the more complex your write workload is, the more human input you'll need in that estimation, so we will cover both cases. Storage is similar in this way but with the added complexity of compression. Queries are more challenging to estimate, but having a rough idea of your query volume is enough to get started

This chapter highlights two different approaches to hardware sizing, one that uses rough estimates, and another that factors in more detail.

InfluxData provides estimated hardware sizing guidelines that you can use to extrapolate sizing for your own workload. It is important to keep in mind that these guidelines are very rough. While InfluxData provides this rough guideline, this chapter is about how to estimate the way these factors apply to your specific circumstance.

System requirements

Memory, disk speed, and compute power

Estimated guidelines in the following table include writes per second, queries per second, and number of unique series, CPU, RAM, and IOPS (input/output operations per second).

vCPU or CPU	RAM	IOPS	Values per second	Queries* per second	Unique series
2-4 cores	2-4 GB	500	< 5,000	< 5	< 100,000
4-6 cores	8-32 GB	500-1000	< 250,000	< 25	< 1,000,000
8+ cores	32+ GB	1000+	> 250,000	> 25	> 1,000,000

* **Queries per second for moderate operational/online dashboard queries.** Queries vary widely in their impact on the

system. For simple or complex queries, we recommend testing and adjusting the suggested requirements as needed. See query guidelines for details.

Another important thing to note is these estimates don't differentiate between workloads that are read-heavy compared to writes. The table below illustrates this difference in the context of CPU cores with a real world benchmark.

Data shape/schema, method of writing it, query type, and query cadence all affect these benchmarks in vastly different ways. Nevertheless, a benchmark with information about the sample data and queries used, gives you an idea of how your workload might compare.

Details about these benchmarks:

- Telegraf system plugins (system, cpu, mem, disk, diskio, etc.) produced this data. To get an idea of what those metrics look like, visit their respective READMEs.
- The columns marked "Isolated" measure the ingest rate and read rate when the other is not happening against the node at all. In other words, when measuring writes, queries are off. When measuring queries, writes are off.
- The columns marked "Combined" are when both writes and reads are turned on. The way to read those columns is essentially that 3 different tests were conducted per row;
    - Writes on with queries off
    - Queries on with writes off
    - Writes and queries on

	Isolated		Combined	
CPU Cores	Writes (values/sec)	Reads (queries/sec)	Writes (values/sec)	Reads (queries/sec)
4	188,012	50	99,700	40
8	405,636	90	207,283	80
16	673,668	150	375,022	140
32	1,056,353	240	650,245	220

Note: The CPU favors queries.

Storage allocation

We recommend running InfluxDB on locally attached solid state drives (SSDs). This is true for any time series database as time series is naturally high velocity. Write volume, query volume, and on-going file compactions all impact disk I/O. Preventing storage from being a bottleneck requires storage that supports high velocity I/O.

For best results, InfluxDB servers should have a minimum of 1000 IOPS on storage to ensure recovery and availability. We recommend at least 2000 IOPS for rapid recovery of cluster data nodes after downtime.

See your cloud provider documentation for IOPS detail on your storage volumes.

Estimating volumes

This section describes how InfluxDB users can estimate their workload volumes to better understand the sizing needs of their instance(s). We will start with writes (ingestion volume).

Writes

There are several units we can use to measure writes:

- Records/points (lines of line protocol)
- Values (number of field values, of which there can be more than one in a record)
- Bytes

We need to measure this over the unit of time, i.e., what is our measure write volume over a given time interval?

Recall that we have been sizing in terms of "values" (see the tables above). We do this for a few reasons:

- Records can contain one or more values
- They are the easiest to estimate without touching a computer – you can merely *think* about how many there are

Estimating based on bytes is another option, discussed later in this chapter.

The estimation methodology assumes you don't have real data to measure but rather a theoretical workload. This is common if you have never used InfluxDB before.

Values estimation

The first thing to do here is determine whether your write load is “regular” or “irregular” in nature. In other words, does each data source report data on a regular cadence or is it more event-driven, requiring more sporadic writes?

Regular

Most metrics cases have regular sampling/reporting intervals. There are many workloads that contain both regular and irregular reporting intervals. However, we suggest you estimate them separately and only combine them after concluding your estimate(s).

For regular data, you need to do a few things:

- Estimate the number of unique time series for which you want to record data
- Determine your sampling rate for these series (1 second, 10 seconds, 60 seconds, 5 minutes, etc.)
- Assuming each series has the same sample rate, convert the count of series to and their sample rates to a time unit of your choosing (seconds typically work well as a common denominator for comparing rates).

  Example: 100,000 series have a 10 second sample rate and another 100,000 have a minutely sample rate. 100,000/10 = 10,000 per second. 100,000/60 = 1,666 per second. 10,000 + 1,666 = 11,666 series values per second.
- *Note: If you have series with different sample rates, you’ll have to do these estimates for each sample rate group, convert each groups’ estimate to a common time unit, and add them up.*

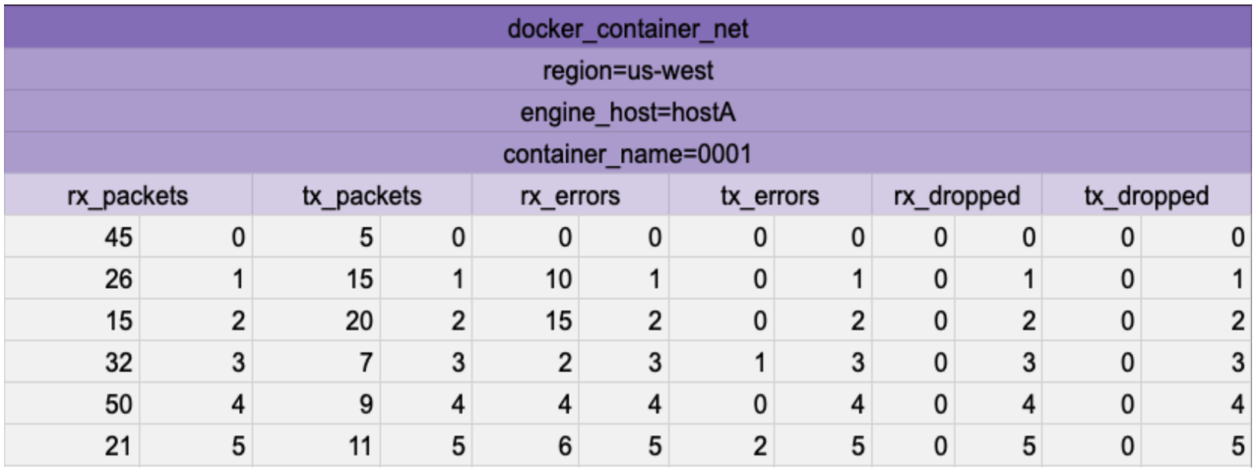
To estimate series count, consider:

- The types of data sources
- The number of each type
- For each type, how many individual metrics you are tracking.

The more precise your estimate, the more accurate your scaled number will be. Accuracy matters more as scale increases.

To illustrate series count, consider the Docker metrics below. The measurement is `` `docker_container_net` `` and this measurement has 3 tags, `` `region` ``, `` `engine_host` ``, and `` `container_name` ``. These tags identify, at the most granular level, each Docker container.

For each Docker container, the measurement contains the six fields below the tags, `` `rx_packets` ``, `` `tx_packets` ``, etc. If this Docker container had five more measurements associated with it, each with six fields (like this one), that would be thirty-six individual metrics. If there are one thousand such containers, you would have a constant inflow of 36,000 series.



docker_container_net											
region=us-west											
engine_host=hostA											
container_name=0001											
rx_packets		tx_packets		rx_errors		tx_errors		rx_dropped		tx_dropped	
45	0	5	0	0	0	0	0	0	0	0	0
26	1	15	1	10	1	0	1	0	1	0	1
15	2	20	2	15	2	0	2	0	2	0	2
32	3	7	3	2	3	1	3	0	3	0	3
50	4	9	4	4	4	0	4	0	4	0	4
21	5	11	5	6	5	2	5	0	5	0	5

(The value in the second column for each field value (i.e, nos. 0-5) corresponds to the measurement number.)

Let's assume that the only data you're tracking are these six measurements in 1,000 different containers. If you used a 1-minute sample rate, you can convert this into seconds: 36,000 metrics / 60 seconds = 36 metric values per second.

Irregular

Irregular sampling adds complexity. It is similar conceptually to measuring regular volume, but requires additional work. Two things must be done here:

- Determine seasonality – a pattern of your sample rate repeating itself, if possible
- Measure actual metrics written in that window

Attempting to take a sample rate from a random batch or an arbitrary minute or hour window may lead to an inaccurate estimation of real volume over a longer period of time.

Data scientists may be able to help gauge seasonality, but many people have to simply guess. If you can determine that the irregularity of sampling follows a repeating pattern, say every 7 days, measure how many metrics you write in that 7-day period. There are techniques available for completing this process; however, they are not InfluxDB-specific and are outside the scope of this book.

Bytes estimation

Estimating bytes is very similar to estimating values except you measure those values with a computer.

For regular workloads:

- Measure the size of a batch
- Count the number of batches
- Convert batch data to a common time unit

To measure the size of a batch:

- Write it to a file and check the size of the file
- Use provided metrics from your current DB (or InfluxDB) to gauge average batch sizes
- Or use any other method you deem best for your toolset

Note: InfluxDB supports gzipped data so if you intend to write gzipped data, measure it gzipped.

For irregular workloads, combine the strategies of measuring irregular values with the measuring of bytes.

Storage

Write volumes affect storage, so start planning storage needs only after determining write volumes. Storage incorporates compression, which adds complexity. For this stage of the sizing process, it is best to also have your write bytes estimated and not gzipped. Gzipping adds a layer of compression that complicates estimation.

The major parameters that affect storage are:

- Write bytes
- Retention (how long you're storing your data)
- Compressibility

Without compression, your storage would be your write bytes accumulated over the duration of your retention, but compression significantly reduces data storage needs.

The expected compression ratio of your specific data is impossible to know without actually storing your real workload in InfluxDB and letting it get fully compacted. That said, understanding the factors that affect the efficacy of compression will help guide your estimate.

Here is a list of the factors that affect data compaction, in order of importance:

- Value types
    - Bools compress better than numbers, which compress better than strings
- Float precision
    - The float compression algorithm is most efficient with floats of similar lengths so truncating is optimal
- Timestamp precision
    - Similar to the floats, the timestamp compression likes even intervals. High precision timestamps can reduce your compression ratio
- Data precision
    - Data closer together in time have better odds of being compacted in the same storage blocks
- Batch sizes
    - Less impactful when fully compacted but impacts the first phase of compactions

Value types:

- Numeric:
    - Most metric data is numerical and numerical data typically gets 70-90% compression compared to raw. Your data precision affects where it falls in that range.
- Bool:
    - Boolean series have a total value cardinality of 2, which makes for very effective compression, typically exceeding 90%.
- String:
    - On average, users achieve around 60% compression for string data.
- Timestamps:
    - Timestamps are too variable to predict, but less precise timestamps and consistent intervals result in better compression ratios.

After estimating your compression rates, you need to consider your data retention. Are you writing all data to one bucket? If so, how long will data stay in the bucket before eviction? For simplicity, sake let’s assume you have one bucket with a one-year retention period.

To estimate your storage needs, take your estimated compressed data for a period of time (data usually compacts fully after a few days) and amortize it over a year. As long as you procure disk space that exceeds your annual data accumulation, you should be fine.

Queries

Determining sizing requirements for query workloads is difficult because both query types and the shape of the underlying data are highly variable.

Many factors impact queries, and the level of impact depends on additional factors. This makes predicting query volume and performance an ever-changing task.

Nevertheless, we can try to determine a general query volume, e.g., small, medium, large, by considering the parameters that affect query performance. This requires a basic understanding of your query load's complexity. You can cross-reference that information with query quantity.

The table below provides a rough approximation of the query complexity scale (*Note: the benchmark tables above used "moderate" queries)*:

Query complexity	Criteria
Simple	● 0-2 functions, 0 regex ● Small time bound (< 30 min for moderate workloads) ● No tag grouping ● Execute in a few milliseconds
Moderate	● 2+ functions and 1+ regex ● Contains grouping ● Execute in a few hundred milliseconds
Complex	● Many functions, multiple regex ● Multiple complex functions ● Large time range ● Execute in multiple to many seconds

After estimating query complexity and volume, you can take advantage of query profiling to gain a deeper understanding of how to improve your queries. See the Flux Profiler package.

Things we know impact performance:

- Use of push-down functions
    - InfluxDB pushes functions like `range()`, `filter()`, `aggregateWindow()` down to storage, making them very fast. Here is a complete list
        - Caveat: push-down functions are pushed down until the query encounters a non-push-down function. At that point, the remainder of the functions in that query occur in memory, not the datastore. Be mindful of function order where appropriate to take advantage of push-downs.
    - Profiling can tell you which functions are push down functions.
- Query cardinality: Determined by how many index entries the query touches.
- Points scanned
- Points returned
- Points grouped
- Window periods
- Groups
- Window-groups

This concludes the primer for estimating the size of InfluxDB instances. If, at any point, you believe you are not getting effective usage of the resources you allotted to your instance(s), it might mean you have room to optimize.

For optimizing, see this blog post and the available resources on optimization in the InfluxDB docs:

- Optimize writes
- Optimize Flux queries

Chapter 3

Install InfluxDB 2.x OSS (open source)

Install InfluxDB where it's convenient for you – on your IoT devices, at the edge, or testing on a device for a home project. Really, anywhere you want!

This chapter covers step-by-step instructions for how to install and set up InfluxDB on various operating systems. To quickly deploy InfluxDB at scale, we recommend using Docker.

Installing InfluxDB OSS

First, choose your operating system, and then click the link below to find instructions for how to install InfluxDB in less than 5 minutes:

- macOS
- Linux
- Windows
- Raspberry Pi

Note: InfluxDB (`influxd`) and the influx CLI are separate packages and are versioned separately. For information about installing the influx CLI, see how to install and use the influx CLI.

Install on macOS

1. Do one of the following:
   - Use Homebrew
   - Manually download and install

2. After installing InfluxDB with Homebrew or manual download, do the following:
   - Access networking port
   - Start and configure InfluxDB

Use Homebrew

We recommend using Homebrew to install InfluxDB on macOS:

```
brew update
brew install influxdb
```

Note: Homebrew also installs `influxdb-cli` as a dependency. For information about using the `influx CLI`, see the influx CLI reference documentation.

Manually download and install

To download the InfluxDB binaries for macOS directly, do the following:

1. Download the InfluxDB (macOS) package.

   Note: This downloads InfluxDB 2.3.0. To download another version, replace the version in the following download url as needed (for example, from 2.3.0 to 2.3.1): `https://dl.influxdata.com/influxdb/releases/influxdb2-2.3.1-darwin-amd64.tar.gz`

2. Unpackage the InfluxDB binary. Do one of the following:

   - Double-click the downloaded package file in **Finder**.
   - Run the following command in a macOS command prompt application, such as Terminal or iTerm2:

     ```
     # Unpackage contents to the current working directory
     ```

     ```
     tar zxvf ~/Downloads/influxdb2-{{< latest-patch
     >}}-darwin-amd64.tar.gz
     ```

3. (Optional) Place the binary in your `$PATH`

   ```
   sudo cp influxdb2-{{< latest-patch >}}-darwin-amd64/influxd
   /usr/local/bin/
   ```

Note: If you do not move the influxd binary into your `$PATH`, prefix the executable ./ to run it in place.

Recommended - Verify the authenticity of the downloaded binary.

For added security, use gpg to verify the signature of your download. (Most operating systems include the gpg command by default. If gpg is not available, see the GnuPG homepage for installation instructions.)

1. Download and import InfluxData's public key:

   ```
   curl -s https://repos.influxdata.com/influxdb2.key | gpg --import -
   ```

2. `Download the signature file for the release by adding .asc to the download URL.`
   `For example:`

   ```
   wget https://dl.influxdata.com/influxdb/releases/influxdb2-{{<
   latest-patch >}}-darwin-amd64.tar.gz.asc
   ```

3. Verify the signature with `gpg --verify`:
   ```
   gpg --verify influxdb2-{{< latest-patch >}}-darwin-amd64.tar.gz.asc
   influxdb2-{{< latest-patch >}}-darwin-amd64.tar.gz
   ```

4. The output from this command should include the following:
   ```
   gpg: Good signature from "InfluxData <support@influxdata.com>" [unknown]
   ```

Note: Both InfluxDB 1.x and 2.x have associated influxd and influx binaries. If InfluxDB 1.x binaries are already in your `$PATH`, run the binaries in place or rename them before putting them in your `$PATH`. If you rename the binaries, all references to influxd and influx in this documentation refer to your renamed binaries.

Access networking port

By default, InfluxDB uses `TCP port 8086` for client-server communication over the [InfluxDB HTTP API]().

Start and configure InfluxDB

To start InfluxDB, open your terminal and run the `influxd` daemon.

Note: Running InfluxDB on macOS Catalina:

macOS Catalina requires downloaded binaries to be signed by registered Apple developers. Currently, when you first attempt to run `influxd`, macOS will prevent it from running. To manually authorize the `influxd` binary:

1. Attempt to run `influxd`.
2. Open **System Preferences** and click **Security & Privacy**.
3. Under the **General** tab, there is a message about `influxd` being blocked. Click **Open Anyway**.

Warning: "too many open files" errors.
After running `influxd`, you might see an error in the log output like the following: `too many open files`

To resolve this error, follow the recommended steps to increase file and process limits for your operating system version, and then restart `influxd`.

To configure InfluxDB, see InfluxDB configuration options, and the `influxd` documentation for information about available flags and options.

Note: **InfluxDB "phone home"**
By default, InfluxDB sends telemetry data back to InfluxData. The InfluxData telemetry page provides information about what data is collected and how it is used.

To opt-out of sending telemetry data back to InfluxData, include the `--reporting-disabled` flag when starting `influxd`:

```
influxd --reporting-disabled
```

Set up InfluxDB OSS

After installing InfluxDB, you'll be ready to set up InfluxDB!

Install on Linux

Do one of the following:

- Install InfluxDB as a service with systemd and then pass arguments to systemd
- Manually download and install the influxd binary

Note: InfluxDB (`influxd`) and the `influx` CLI are separate packages and are versioned separately. By default, the recommended Linux installation will install both. If you want to install InfluxDB (`influxd`) only, run the install command with the `` `--no-install-recommends` `` flag. For information about using the `influx` CLI, see: Install and use the influx CLI.

Install InfluxDB as a service with systemd

1. Download and install the appropriate `.deb` or `.rpm` file using a URL from the InfluxData downloads page with the following commands:

```
# Ubuntu/Debian
wget https://dl.influxdata.com/influxdb/releases/influxdb2-{{<
latest-patch >}}-xxx.deb
sudo dpkg -i influxdb2-{{< latest-patch >}}-xxx.deb


# Red Hat/CentOS/Fedora
wget https://dl.influxdata.com/influxdb/releases/influxdb2-{{<
latest-patch >}}-xxx.rpm
sudo yum localinstall influxdb2-{{< latest-patch >}}-xxx.rpm
```

Use the exact filename of the download of `.rpm` package (for example, `influxdb2-{{< latest-patch >}}-amd64.rpm`).

2. Start the InfluxDB service:

   ```
   sudo service influxdb start
   ```

   Installing the InfluxDB package creates a service file at `/lib/systemd/services/influxdb.service` to start InfluxDB as a background service on startup.

3. Restart your system and verify that the service is running correctly:

   ```
   $  sudo service influxdb status
      influxdb.service - InfluxDB is an open-source, distributed,
   time series database
     Loaded: loaded (/lib/systemd/system/influxdb.service; enabled;
   vendor preset: enable>
     Active: active (running)
   ```

For information about where InfluxDB stores data on disk when running as a service, see File system layout.

To customize your InfluxDB configuration, use either command line flags (arguments), environment variables, or an InfluxDB configuration file. See InfluxDB configuration options for more information.

Pass arguments to systemd

1. Add one or more lines like the following containing arguments for `influxd` to `/etc/default/influxdb2:`

   ```
   ARG1="--http-bind-address :8087"
   ARG2="<another argument here>"
   ```

2. Edit the `/lib/systemd/system/influxdb.service` file as follows:

   ```
   ExecStart=/usr/bin/influxd $ARG1 $ARG2
   ```

Manually download and install the influxd binary

1. Do one of the following:
   - **Download from your browser**

     InfluxDB (amd64)

     InfluxDB (arm)
   - **Download from the command line**

     ```
     # amd64
     wget https://dl.influxdata.com/influxdb/releases/influxdb2-{{<
     latest-patch >}}-linux-amd64.tar.gz

     # arm
     wget https://dl.influxdata.com/influxdb/releases/influxdb2-{{<
     latest-patch >}}-linux-arm64.tar.gz
     ```

2. Extract the downloaded binary.

   Note: The following commands are examples. Adjust the filenames, paths, and utilities if necessary.

   ```
   # amd64
   tar xvzf path/to/influxdb2-{{< latest-patch >}}-linux-amd64.tar.gz

   # arm
   ```

```
tar xvzf path/to/influxdb2-{{< latest-patch >}}-linux-arm64.tar.gz
```

3. (Optional) Place the extracted `influxd` executable binary in your system `$PATH`.

   ```
   # amd64
   sudo cp influxdb2-{{< latest-patch >}}-linux-amd64/influxd
   /usr/local/bin/
   ```

   ```
   # arm
   sudo cp influxdb2-{{< latest-patch >}}-linux-arm64/influxd
   /usr/local/bin/
   ```

   If you do not move the `influxd` binary into your `$PATH`, prefix the executable `./` to run it in place.

4. **(Recommended)** Verify the authenticity of downloaded binary.

   For added security, use `gpg` to verify the signature of your download. (Most operating systems include the `gpg` command by default. If `gpg` is not available, see the GnuPG homepage for installation instructions.)

   a. Download and import InfluxData's public key:

      ```
      curl -s https://repos.influxdata.com/influxdb2.key | gpg --import
      -
      ```

   b. Download the signature file for the release by adding `.asc` to the download URL. For example:

      ```
      wget https://dl.influxdata.com/influxdb/releases/influxdb2-{{<
      latest-patch >}}-linux-amd64.tar.gz.asc
      ```

   c. Verify the signature with `gpg --verify`:

      ```
      gpg --verify influxdb2-{{< latest-patch >}}-linux-amd64.tar.gz.asc
      influxdb2-{{< latest-patch >}}-linux-amd64.tar.gz
      ```

      The output from this command should include the following:

      ```
      gpg: Good signature from "InfluxData
      <support@influxdata.com>" [unknown]
      ```

Start InfluxDB

If InfluxDB was installed as a systemd service, systemd manages the `influxd` daemon and no further action is required. If the binary was manually downloaded and added to the system `$PATH`, start the `influxd` daemon with the following command: `influxd`

See the `influxd` documentation for information about available flags and options.

Networking ports

By default, InfluxDB uses TCP port `8086` for client-server communication over the InfluxDB HTTP API.

Note: InfluxDB "phone home"
By default, InfluxDB sends telemetry data back to InfluxData. The InfluxData telemetry page provides information about what data is collected and how it is used.

To opt-out of sending telemetry data back to InfluxData, include the `--reporting-disabled` flag when starting `influxd`.

```
influxd --reporting-disabled
```

Set up InfluxDB OSS

After installing InfluxDB, now, you're ready to set up InfluxDB!

Install on Windows

System requirements

- Windows 10
- 64-bit AMD architecture
- PowerShell or Windows Subsystem for Linux (WSL)

Command line examples

Use **PowerShell** or **WSL** to execute `influx` and `influxd` commands. The command line examples in this documentation use `influx` and `influxd` as if installed on the system `PATH`. If these binaries are not installed on your `PATH`, replace `influx` and `influxd` in the provided examples with `./influx` and `./influxd` respectively.

Note: InfluxDB (`influxd`) and the `influx` CLI are separate packages and are versioned separately. For information about installing the `influx` CLI, see: Install and use the influx CLI.

Download and install InfluxDB

InfluxDB (Windows)

Expand the downloaded archive into `C:\Program Files\InfluxData\` and rename the files if desired.

```
> Expand-Archive .\influxdb2-{{< latest-patch >}}-windows-amd64.zip
-DestinationPath 'C:\Program Files\InfluxData\'

> mv 'C:\Program Files\InfluxData\influxdb2-{{< latest-patch
>}}-windows-amd64' 'C:\Program Files\InfluxData\influxdb'
```

Access networking port

By default, InfluxDB uses TCP port `8086` for client-server communication over the InfluxDB HTTP API.

Start InfluxDB

- In PowerShell, navigate into `C:\Program Files\InfluxData\influxdb` and start InfluxDB by running the `influxd` daemon:

```
> cd -Path 'C:\Program Files\InfluxData\influxdb'
> ./influxd
```

See the `influxd` documentation for information about available flags and options.

Note: Grant network access
When starting InfluxDB for the first time, Windows Defender will appear with the following message: *Windows Defender Firewall has blocked some features of this app.* Do the following to resolve:

1. Select **Private networks, such as my home or work network.**
2. Click **Allow access.**

Note: InfluxDB "phone home"
By default, InfluxDB sends telemetry data back to InfluxData. The InfluxData telemetry page provides information about what data is collected and how it is used.

To opt-out of sending telemetry data back to InfluxData, include the `--reporting-disabled` flag when starting `influxd`.

```
influxd --reporting-disabled
```

Set up InfluxDB OSS

After installing InfluxDB, now, you're ready to set up InfluxDB!

Install on Raspberry Pi

Requirements

To run InfluxDB on Raspberry Pi, you need:

- a Raspberry Pi 4+ or 400
- a 64-bit operating system. We recommend installing a 64-bit version of Ubuntu of Ubuntu Desktop or Ubuntu Server compatible with 64-bit Raspberry Pi.

Install Linux binaries

Follow the Linux installation instructions to install InfluxDB on a Raspberry Pi.

Monitor your Raspberry Pi

Use the InfluxDB Raspberry Pi template to easily configure collecting and visualizing system metrics for the Raspberry Pi.

Monitor 32-bit Raspberry Pi systems

If you have a 32-bit Raspberry Pi, use Telegraf to collect and send data to:

- InfluxDB OSS, running on a 64-bit system
- InfluxDB Cloud with a **Free Tier** account
- InfluxDB Cloud with a paid **Usage-Based** account with relaxed resource restrictions.

After installing InfluxDB, now, you're ready to set up InfluxDB!

Set up InfluxDB

The initial setup process for InfluxDB walks through creating a default organization, user, bucket, and Operator API token. Alternatively, you can use the InfluxDB user interface (UI) or the `influx` command line interface (CLI) for initial setup.

Note: Operator token permissions
The Operator token created in the InfluxDB setup process has full read and write access to all organizations in the database. To prevent accidental interactions across organizations, we recommend creating an All-Access token for each organization and using those to manage InfluxDB.

Set up InfluxDB through the UI

1. With InfluxDB running, visit localhost:8086.
2. Click **Get Started**

Note: InfluxDB is accessed at localhost:8086 by default, but you can also customize your InfluxDB host and port. For more information, see InfluxDB OSS URLs.

Set up your initial user

1. Enter a **Username** for your initial user.
2. Enter a **Password** and **Confirm Password** for your user.
3. Enter your initial **Organization Name**.
4. Enter your initial **Bucket Name**.
5. Click **Continue**.

InfluxDB is now initialized with a primary user, organization, and bucket. You are ready to write or collect data.

Set up InfluxDB through the influx CLI

If you haven't already, see how to install and use the influx CLI. Begin the InfluxDB setup process via the `influx CLI` by running: `influx setup`, and then do the following:

1. Enter a primary username.
2. Enter a password for your user.
3. Confirm your password by entering it again.
4. Enter a name for your primary organization.
5. Enter a name for your primary bucket.
6. Enter a retention period for your primary bucket — valid units are nanoseconds (ns), microseconds (us or µs), milliseconds (ms), seconds (s), minutes (m), hours (h), days (d), and weeks (w). Enter nothing for an infinite retention period.
7. Confirm the details for your primary user, organization, and bucket.

InfluxDB is now initialized with a primary user, organization, bucket, and API token. InfluxDB also creates a configuration profile for you so that you don't have to add your InfluxDB host, organization, and token to every command. To view that config profile, use the `influx config list` command.

To continue to use InfluxDB via the CLI, you need the API token created during setup. To view the token, log into the UI with the credentials created above. (For instructions, see View tokens in the InfluxDB UI).

You are ready to write or collect data.

Note: To automate the setup process, use flags to provide the required information.

After you've set up InfluxDB, see more information about configuring your environment.

Chapter 4

InfluxDB Configuration

InfluxDB OSS provides a huge amount of flexibility in terms of configuration. In this chapter, you will learn some general best practices and some of the more important configuration options available to you.

InfluxDB configuration basics

There are three ways to configure InfluxDB:

- Use the InfluxDB CLI to set configuration flags
- Set environment variables
- Set configuration options using a configuration file

InfluxDB will also take precedence for configuration in that order, with influxd flags taking priority and overruling anything set below, with the same applying for environment variables over configuration files.

To see the current configuration settings of your InfluxDB instance, you can run the following CLI command:

```
influx server-config
```

InfluxDB config management and command options

Beyond configuring the settings of the InfluxDB instance itself, users can use the CLI to manage higher level features of InfluxDB as well. Some common use cases:

- **Bucket management** - The influx bucket command can be used to create, update, delete, and list existing buckets from the command line.
- **API token management** - The influx auth command can be used for creating, activating, deleting, and listing API tokens for your InfluxDB server.
- **User management** - The influx user commands can be used for creating, deleting, updating, and listing the users of an InfluxDB instance.

The InfluxDB CLI is also capable of reading, writing, and deleting data from InfluxDB, in addition to a number of other features beyond the scope of this chapter covering configuration options for InfluxDB.

InfluxDB security configuration

InfluxDB provides a number of options for security that can be configured depending on your use case and environment. Let's look at some things you can do to make your InfluxDB installation more secure.

Enable TLS

While InfluxDB can be communicated with over plain HTTP, InfluxData strongly recommends enabling HTTPS so your data is encrypted and secure during network requests. To enable TLS you will need to generate your certs and then set the certificate file permissions so that the user running InfluxDB can read the file.

To start InfluxDB with TLS, you will need to run the following command with the path set to your cert and keys:

```
influxd \
--tls-cert="<path-to-crt>" \
--tls-key="<path-to-key>"
```

Next you can verify TLS is working by pinging InfluxDB:

```
curl -v https://localhost:8086/api/v2/ping
```

TLS settings can be further configured using the tls-min-version and tls-strict-cipher flags.

Managing secrets

Secrets are key-value pairs that contain sensitive information like API keys, passwords, or certificates. To safely store and access secrets with InfluxDB, you have 2 options:

- Use the built-in embedded key-value database.
- Set up a Vault server to store your secrets and then connect InfluxDB to access your secrets.

If you are using the built-in InfluxDB secrets storage option, you can then use the InfluxDB CLI to add, view, update, and delete secrets with the following commands:

- **Add** - influx secret update `-k <secret-key>`
- **View** - influx secret list
- **Update** - influx secret update `-k <secret-key>`
- **Delete** - influx secret delete `-k <secret-key>`

When using Vault to store your secrets, you will need to use the InfluxDB API to make changes to your secrets using PATCH requests. Here is a basic example of adding a secret via API:

```
curl --request PATCH http://localhost:8086/api/v2/orgs/<org-id>/secrets \
  --header 'Authorization: Token YOURAUTHTOKEN' \
  --header 'Content-type: application/json' \
  --data '{
      "<secret-key>": "<secret-value>"
}'
```

Your secrets can then be used from inside Flux scripts by using the Flux secrets package. The following example shows how you can store Postgres database credentials and then query that database from within Flux:

```
import "influxdata/influxdb/secrets"
import "sql"

username = secrets.get(key: "POSTGRES_USERNAME")
password = secrets.get(key: "POSTGRES_PASSWORD")

sql.from(
    driverName: "postgres",
    dataSourceName: "postgresql://${username}:${password}@localhost",
    query:"SELECT * FROM example-table",
)
```

Shards and shard groups configuration

InfluxDB organizes time series data into what are called **shards** when writing data to disk. These shards are grouped together into **shard groups**. Shards and shard groups are an important concept to understand if you are using retention policies for your InfluxDB buckets.

An InfluxDB shard is an encoded and compressed collection of time series data for a given time range, which is defined by the shard group duration. Shard groups belong to InfluxDB buckets and contain the time series data for a specific time range, which consists of all the shards within that range.

InfluxDB creates default shard group duration values based on bucket retention policies, but you also have the option to set a custom value when creating your bucket using the InfluxDB CLI. By default InfluxDB uses the following shard group durations:

- Bucket retention period less than 2 days: 1 hour shard group duration
- Bucket retention period between 2 days and 6 months: 1 day shard group duration
- bucket retention period greater than 6 months: 7 day shard group duration

To set a custom shard group duration rather than using the default values, simply add the `` `--shard-group-duration` `` flag while creating your bucket with `influx bucket create` or update an existing bucket with `influx bucket update`. For more information, you can check the full documentation on shards and shard groups

Why care about shard groups?

Depending on your use case, shard duration may become relevant. InfluxDB writes data to shards that are un-compacted. Once a shard's duration is passed, it is compacted and any writes of historical data will require uncompacting shards. If you will frequently have situations where you are writing historical data, it might make sense to define a custom shard group duration value that fits best for your workload.

Additional resources

Find the full list of InfluxDB configuration options in our documentation.

Chapter 5

Getting Started with InfluxDB and Docker

Basic Docker setup

To get started with InfluxDB using Docker, you can simply use the official Docker image provided by InfluxData by running the following command:

```
docker run --name influxdb -p 8086:8086 influxdb:2.3.0
```

With one command, you will have a running instance of InfluxDB ready to go.

Persisting data outside InfluxDB container

If you want to persist data outside the InfluxDB container, you can create a new directory and navigate to it with this command:

```
mkdir path/to/influxdb-docker-data-volume && cd $_
```

Then you can run the following Docker command to make sure InfluxDB stores data in your current directory by using the volume flag:

```
docker run \
    --name influxdb \
    -p 8086:8086 \
    --volume $PWD:/var/lib/influxdb2 \
    influxdb:2.3.0
```

Configuring InfluxDB with Docker

If you want to configure your InfluxDB Docker instance, you can do so using a mounted configuration file. First you need to make sure you are persisting your data outside the InfluxDB container itself by following the section above.

Next you need to generate a default configuration file and store it on the host file system with the following command:

```
docker run \
  --rm influxdb:2.3.0 \
  influxd print-config > config.yml
```

Now you can restart the InfluxDB container, and it will pull its configuration from the created file:

```
docker run -p 8086:8086 \
  -v $PWD/config.yml:/etc/influxdb2/config.yml \
  influxdb:2.3.0
```

Open a shell in the InfluxDB container to use Influx CLI

To use the InfluxDB CLI you'll need to create a shell in your InfluxDB container so you can interact with it by using the following command:

```
docker exec -it influxdb /bin/bash
```

Docker-compose setup

If you want to run applications that require multiple containers, Docker Compose is a useful tool. By creating a YAML file, you can configure and define the details of how you want your application to run and then start all your containers with a single command.

In the context of InfluxDB, you can use the InfluxDB IoT Center application as a real-world example of how you can use InfluxDB with Docker Compose for more advanced applications. IoT Center consists of 4 different containers — InfluxDB, Telegraf, an MQTT broker, and the IoT Center NodeJS application backend. Docker Compose allows you to define environment variables, port numbers, volumes, and dependencies for each container. Here's the full IoT Center Docker Compose YAML file:

```
version: "3"
services:
  influxdb_v2:
    image: influxdb:latest
    ports:
      - "8086:8086"
    environment:
      - INFLUXD_HTTP_BIND_ADDRESS=:8086
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=my-user
      - DOCKER_INFLUXDB_INIT_PASSWORD=my-password
      - DOCKER_INFLUXDB_INIT_ORG=my-org
      - DOCKER_INFLUXDB_INIT_BUCKET=iot_center
      - DOCKER_INFLUXDB_INIT_RETENTION=30d
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=my-token
    command: influxd --reporting-disabled

  mosquitto:
    image: eclipse-mosquitto:2.0.10
    ports:
      - "1883:1883"
    volumes:
      - ./app/server/mqtt/docker-mosquitto/mosquitto/:/mosquitto/
  telegraf:
    image: telegraf:latest
    volumes:
      - ./app/server/mqtt/telegraf.docker.conf:/etc/telegraf/telegraf.conf

  iot_center:
    build: ./app
    ports:
      - "5000:5000"
    environment:
      - INFLUX_URL=http://influxdb_v2:8086/
      - INFLUX_TOKEN=my-token
      - INFLUX_ORG=my-org
      - INFLUX_BUCKET=iot_center
      - MQTT_TOPIC=iot_center
      - MQTT_URL=mqtt://mosquitto:1883
    depends_on:
      - mosquitto
      - influxdb_v2
      - telegraf
```

Docker environment variables

When running InfluxDB on Docker, the following environment variables are required:

- `DOCKER_INFLUXDB_INIT_USERNAME` - The username to set for the system's initial super-user.
- `DOCKER_INFLUXDB_INIT_PASSWORD` - The password to set for the system's initial super-user.
- `DOCKER_INFLUXDB_INIT_ORG` - The name to set for the system's initial organization

- `DOCKER_INFLUXDB_INIT_BUCKET` - The name to set for the system's initial bucket

There are a number of other non-required environment variables available for customizing your InfluxDB installation to your preferences.

The optimal size of an InfluxDB instance is dependent on a lot of factors. This chapter covers how you should think about those factors and how to measure and estimate them to get as close to an accurate size prediction as possible.

Three workload factors, write volume, query volume, and storage, affect the sizing of an InfluxDB OSS node. You can estimate write volume and storage needs fairly accurately. However, the more complex your write workload is, the more human input you'll need in that estimation, so we will cover both cases. Storage is similar in this way but with the added complexity of compression. Queries are more challenging to estimate, but having a rough idea of your query volume is enough to get started

This chapter highlights two different approaches to hardware sizing, one that uses rough estimates, and another that factors in more detail.

InfluxData provides estimated hardware sizing guidelines that you can use to extrapolate sizing for your own workload. It is important to keep in mind that these guidelines are very rough. While InfluxData provides this rough guideline, this chapter is about how to estimate the way these factors apply to your specific circumstance.

System requirements

Memory, disk speed, and compute power

Estimated guidelines in the following table include writes per second, queries per second, and number of unique series, CPU, RAM, and IOPS (input/output operations per second).

vCPU or CPU	**RAM**	**IOPS**	**Values per second**	**Queries* per second**	**Unique series**
2-4 cores	2-4 GB	500	< 5,000	< 5	< 100,000
4-6 cores	8-32 GB	500-1000	< 250,000	< 25	< 1,000,000
8+ cores	32+ GB	1000+	> 250,000	> 25	> 1,000,000

* **Queries per second** for moderate operational/online dashboard queries. Queries vary widely in their impact on the

system. For simple or complex queries, we recommend testing and adjusting the suggested requirements as needed. See query guidelines for details.

Another important thing to note is these estimates don't differentiate between workloads that are read-heavy compared to writes. The table below illustrates this difference in the context of CPU cores with a real world benchmark.

Data shape/schema, method of writing it, query type, and query cadence all affect these benchmarks in vastly different ways. Nevertheless, a benchmark with information about the sample data and queries used, gives you an idea of how your workload might compare.

Details about these benchmarks:

- Telegraf system plugins (system, cpu, mem, disk, diskio, etc.) produced this data. To get an idea of what those metrics look like, visit their respective READMEs.
- The columns marked "Isolated" measure the ingest rate and read rate when the other is not happening against the node at all. In other words, when measuring writes, queries are off. When measuring queries, writes are off.
- The columns marked "Combined" are when both writes and reads are turned on. The way to read those columns is essentially that 3 different tests were conducted per row;
    - Writes on with queries off
    - Queries on with writes off
    - Writes and queries on

	Isolated		Combined	
CPU Cores	Writes (values/sec)	Reads (queries/sec)	Writes (values/sec)	Reads (queries/sec)
4	188,012	50	99,700	40
8	405,636	90	207,283	80
16	673,668	150	375,022	140
32	1,056,353	240	650,245	220

Note: The CPU favors queries.

Storage allocation

We recommend running InfluxDB on locally attached solid state drives (SSDs). This is true for any time series database as time series is naturally high velocity. Write volume, query volume, and on-going file compactions all impact disk I/O. Preventing storage from being a bottleneck requires storage that supports high velocity I/O.

For best results, InfluxDB servers should have a minimum of 1000 IOPS on storage to ensure recovery and availability. We recommend at least 2000 IOPS for rapid recovery of cluster data nodes after downtime.

See your cloud provider documentation for IOPS detail on your storage volumes.

Estimating volumes

This section describes how InfluxDB users can estimate their workload volumes to better understand the sizing needs of their instance(s). We will start with writes (ingestion volume).

Writes

There are several units we can use to measure writes:

- Records/points (lines of line protocol)
- Values (number of field values, of which there can be more than one in a record)
- Bytes

We need to measure this over the unit of time, i.e., what is our measure write volume over a given time interval?

Recall that we have been sizing in terms of "values" (see the tables above). We do this for a few reasons:

- Records can contain one or more values
- They are the easiest to estimate without touching a computer – you can merely think about how many there are

Estimating based on bytes is another option, discussed later in this chapter.

The estimation methodology assumes you don't have real data to measure but rather a theoretical workload. This is common if you have never used InfluxDB before.

Values estimation

The first thing to do here is determine whether your write load is “regular” or “irregular” in nature. In other words, does each data source report data on a regular cadence or is it more event-driven, requiring more sporadic writes?

Regular

Most metrics cases have regular sampling/reporting intervals. There are many workloads that contain both regular and irregular reporting intervals. However, we suggest you estimate them separately and only combine them after concluding your estimate(s).

For regular data, you need to do a few things:

- Estimate the number of unique time series for which you want to record data
- Determine your sampling rate for these series (1 second, 10 seconds, 60 seconds, 5 minutes, etc.)
- Assuming each series has the same sample rate, convert the count of series to and their sample rates to a time unit of your choosing (seconds typically work well as a common denominator for comparing rates).

Example: 100,000 series have a 10 second sample rate and another 100,000 have a minutely sample rate. 100,000/10 = 10,000 per second. 100,000/60 = 1,666 per second. 10,000 + 1,666 = 11,666 series values per second.

- Note: If you have series with different sample rates, you’ll have to do these estimates for each sample rate group, convert each groups’ estimate to a common time unit, and add them up.

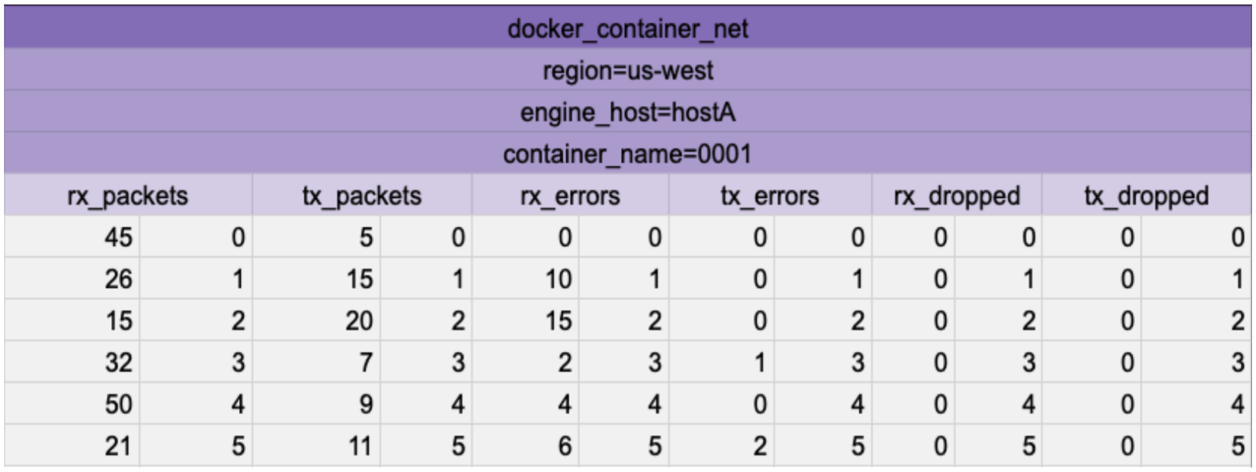
To estimate series count, consider:

- The types of data sources
- The number of each type
- For each type, how many individual metrics you are tracking.

The more precise your estimate, the more accurate your scaled number will be. Accuracy matters more as scale increases.

To illustrate series count, consider the Docker metrics below. The measurement is `docker_container_net` and this measurement has 3 tags, ``region``, ``engine_host``, and ``container_name``. These tags identify, at the most granular level, each Docker container.

For each Docker container, the measurement contains the six fields below the tags, `` `rx_packets` ``, `` `tx_packets` ``, etc. If this Docker container had five more measurements associated with it, each with six fields (like this one), that would be thirty-six individual metrics. If there are one thousand such containers, you would have a constant in-flow of 36,000 series.



docker_container_net											
region=us-west											
engine_host=hostA											
container_name=0001											
rx_packets		tx_packets		rx_errors		tx_errors		rx_dropped		tx_dropped	
45	0	5	0	0	0	0	0	0	0	0	0
26	1	15	1	10	1	0	1	0	1	0	1
15	2	20	2	15	2	0	2	0	2	0	2
32	3	7	3	2	3	1	3	0	3	0	3
50	4	9	4	4	4	0	4	0	4	0	4
21	5	11	5	6	5	2	5	0	5	0	5

(The value in the second column for each field value (i.e, nos. 0-5) corresponds to the measurement number.)

Let's assume that the only data you're tracking are these six measurements in 1,000 different containers. If you used a 1-minute sample rate, you can convert this into seconds: 36,000 metrics / 60 seconds = 36 metric values per second.

Irregular

Irregular sampling adds complexity. It is similar conceptually to measuring regular volume, but requires additional work. Two things must be done here:

- Determine seasonality – a pattern of your sample rate repeating itself, if possible
- Measure actual metrics written in that window

Attempting to take a sample rate from a random batch or an arbitrary minute or hour window may lead to an inaccurate estimation of real volume over a longer period of time.

Data scientists may be able to help gauge seasonality, but many people have to simply guess. If you can determine that the irregularity of sampling follows a repeating pattern, say every 7 days, measure how many metrics you write in that 7-day period. There are techniques available for completing this process, however, they are not InfluxDB-specific and are outside the scope of this book.

Bytes estimation

Estimating bytes is very similar to estimating values except you measure those values with a computer.

For regular workloads:

- Measure the size of a batch
- Count the number of batches
- Convert batch data to a common time unit

To measure the size of a batch:

- Write it to a file and check the size of the file
- Use provided metrics from your current DB (or InfluxDB) to gauge average batch sizes
- Or use any other method you deem best for your toolset

Note: InfluxDB supports gzipped data so if you intend to write gzipped data, measure it gzipped.

For irregular workloads, combine the strategies of measuring irregular values with the measuring of bytes.

Storage

Write volumes affect storage, so start planning storage needs only after determining write volumes. Storage incorporates compression, which adds complexity. For this stage of the sizing process, it is best to also have your write bytes estimated and not gzipped. Gzipping adds a layer of compression that complicates estimation.

The major parameters that affect storage are:

- Write bytes
- Retention (how long you're storing your data)
- Compressibility

Without compression, your storage would be your write bytes accumulated over the duration of your retention, but compression significantly reduces data storage needs.

The expected compression ratio of your specific data is impossible to know without actually storing your real workload in InfluxDB and letting it get fully compacted. That said, understanding the factors that affect the efficacy of compression will help guide your estimate.

Here is a list of the factors that affect data compaction, in order of importance:

- Value types
    - Bools compress better than numbers, which compress better than strings
- Float precision
    - The float compression algorithm is most efficient with floats of similar lengths so truncating is optimal
- Timestamp precision
    - Similar to the floats, the timestamp compression likes even intervals. High precision timestamps can reduce your compression ratio
- Data precision
    - Data closer together in time have better odds of being compacted in the same storage blocks
- Batch sizes
    - Less impactful when fully compacted but impacts the first phase of compactions

Value types:

- Numeric:
    - Most metric data is numerical and numerical data typically gets 70-90% compression compared to raw. Your data precision affects where it falls in that range.
- Bool:
    - Boolean series have a total value cardinality of 2, which makes for very effective compression, typically exceeding 90%.
- String:
    - On average, users achieve around 60% compression for string data.
- Timestamps:
    - Timestamps are too variable to predict, but less precise timestamps and consistent intervals result in better compression ratios.

After estimating your compression rates, you need to consider your data retention. Are you writing all data to one bucket? If so, how long will data stay in the bucket before eviction? For simplicity, sake let's assume you have one bucket with a one-year retention period.

To estimate your storage needs, take your estimated compressed data for a period of time (data usually compacts fully after a few days) and amortize it over a year. As long as you procure disk space that exceeds your annual data accumulation, you should be fine.

Queries

Determining sizing requirements for query workloads is difficult because both query types and the shape of the underlying data are highly variable.

Many factors impact queries, and the level of impact depends on additional factors. This makes predicting query volume and performance an ever-changing task.

Nevertheless, we can try to determine a general query volume, e.g., small, medium, large, by considering the parameters that affect query performance. This requires a basic understanding of your query load's complexity. You can cross-reference that information with query quantity.

The table below provides a rough approximation of the query complexity scale *(Note: the benchmark tables above used "moderate" queries):*

Query complexity	Criteria
Simple	● 0-2 functions, 0 regex ● Small time bound (< 30 min for moderate workloads) ● No tag grouping ● Execute in a few milliseconds
Moderate	● 2+ functions and 1+ regex ● Contains grouping ● Execute in a few hundred milliseconds
Complex	● Many functions, multiple regex ● Multiple complex functions ● Large time range

	• Execute in multiple to many seconds

After estimating query complexity and volume, you can take advantage of query profiling to gain a deeper understanding of how to improve your queries. See the Flux Profiler package.

Things we know impact performance:

- Use of push-down functions
    - InfluxDB pushes functions like `range()`, `filter()`, `aggregateWindow()` down to storage, making them very fast. Here is a complete list
        - Caveat: push-down functions are pushed down until the query encounters a non-push-down function. At that point, the remainder of the functions in that query occur in memory, not the datastore. Be mindful of function order where appropriate to take advantage of push-downs.
    - Profiling can tell you which functions are push down functions.
- Query cardinality: Determined by how many index entries the query touches.
- Points scanned
- Points returned
- Points grouped
- Window periods
- Groups
- Window-groups

This concludes the primer for estimating the size of InfluxDB instances. If, at any point, you believe you are not getting effective usage of the resources you allotted to your instance(s), it might mean you have room to optimize.

For optimizing, see this blog post and the available resources on optimization in the InfluxDB docs:

- Optimize writes
- Optimize Flux queries

Chapter 6

Deploying at Scale - Ansible

Large companies often use InfluxDB in a globally distributed manner. While InfluxDB OSS is not clusterable today, users are deploying it as the data layer in disparate data locales. Since the introduction of Edge Data Replication in InfluxDB (OSS), distributed InfluxDB nodes can now safely and easily replicate select local data sets to an instance in the cloud, making the globally distributed architecture even more viable and therefore more common.

Deploying InfluxDB widely at your network edge provides detailed visibility into the data at those edge locations. Having a distributed InfluxDB architecture is valuable so we want to enable users to deploy many instances at once as easily as possible. Using a deployment and configuration management tool, like Ansible, makes it easy to scale InfluxDB deployments.

This chapter focuses on deploying InfluxDB with Ansible.

Housekeeping items:

- This chapter assumes you understand the fundamentals and terminology of both Ansible and AWS. The steps in the “Infrastructure” section are in the context of AWS but they should be similar to other cloud provider modules.
- This chapter covers the complete setup process, but every step may not be relevant to you. Skip the step(s) that do not apply to you. For example, if you already have an inventory of hosts ready to go, skip over the “Infrastructure” section and go right into “Deploy InfluxDB” section.
- The infrastructure creation procedure depends on the `amazon.aws` module and must be a version >= 2.2. Ideally you should use the latest version available (at time of writing, it is 4.0.0). This also depends on Python 3.6+, Boto 1.16+, and Botocore 1.19+.
- The sections below break up each procedure into logical parts and functions. This takes the form of a playbook that includes the appropriate order of operations. In practice, you may want all this information in the same playbook or to bring everything together with `include`. However, doing so will typically require some customizations to how tasks target inventory because some tasks target your controller host (localhost) and target/remote hosts. You can switch between those hosts in a single playbook but that is not required in the configuration described here.

Infrastructure

The first thing you need is the infrastructure on which to deploy your instances of InfluxDB. There are many ways to deploy infrastructure and specific ways to do so at the edge (i.e., Greengrass).

For simplicity, this section discusses a basic setup of EC2 instances intended to function as a distributed deployment.

For cleanliness, we've chosen to limit the yaml to tasks. The full play, with any boilerplate, is available at the end of this section.

Start by creating infrastructure. If you have your own way of creating a target host inventory, you can proceed to the next section.

This portion of the Ansible play creates the AWS servers necessary to run InfluxDB. Make sure you have the necessary AWS credentials and a VPC with permission to handle outside SSH/HTTP traffic.

First, create a security group in which to launch EC2 instances.

~~~~
Create security group
```
  tasks:
    - name: Create security group
      ec2_group:
         name: ansible-influxdb
         region: us-east-1
         rules:
            - proto: tcp
              from_port: 22
              to_port: 22
              cidr_ip: 0.0.0.0/0
```
~~~~

This creates a security group that allows SSH traffic for any instance that is a part of it. Note that we're using a source address of 0.0.0.0/0 which is not a security best practice. Infrastructure security is out of scope for this chapter and book.

To enable use of the API from a remote machine,add the InfluxDB port, 8086, like we did with SSH. If you configure InfluxDB to use a different port, reflect that here as well:

~~~~
```
            - proto: tcp
              from_port: 8086
~~~~

~~~~
              to_port: 8086
              cidr_ip: 0.0.0.0/0
```
~~~~

Create instances

~~~~
```
  tasks:
…
    - name: Provision instances
      ec2_instance:
         key_name: ansible
         group: ansible
         instance_type: t2.large
         image_id: ami-06640050dc3f556bb
         region: us-east-1
         wait: true
         exact_count: 10
```
~~~~

This creates ten t2.large instances of the latest (at time of writing) Ubuntu free tier image. Also, if needed, you may want to dynamically add the hosts to a host group that you will target.

Now that the remote hosts are created and accessible, we need to target them. For this, we rely on the AWS ec2_inventory plugin.

Once this is all set, you need to create the final file, which we're calling `infra.yaml`. It should look like this:

~~~~
```
---

- name: Create infrastructure
  hosts: localhost
  gather_facts: false
~~~~

```
  tasks:
    - name: Create security group
      ec2_group:
         name: ansible-influxdb
         region: us-east-1
         rules:
            - proto: tcp
              from_port: 22
              to_port: 22
              cidr_ip: 0.0.0.0/0
            - proto: tcp
              from_port: 8086
              to_port: 8086
              cidr_ip: 0.0.0.0/0

    - name: Provision instances
      ec2_instance:
         key_name: ansible
         group: ansible
         instance_type: t2.large
         image_id: ami-06640050dc3f556bb
         region: us-east-1
         wait: true
         exact_count: 10
```

Now you can deploy InfluxDB.

Deploy InfluxDB

The most basic deployment of InfluxDB via Ansible should be able to do the following:

- Download
- Install
- Run
- Restart when necessary

This basic set of instructions does not address operations like configuration. We'll cover configuring with Ansible separately to keep the basic (default deployment) cleaner.

Note: For the first procedure, you need to be able to promote yourself to root in order to deal with system level things.

Given our chosen EC2 image was Ubuntu, we will use the "apt" package management system.

First, we need to prep the host for download by importing the appropriate GPG signing key and adding the InfluxDB package to the package repository. You can accomplish this with two tasks:

~~~
```
  tasks:
    - name: Import InfluxDB GPG signing key
      become: true
      apt_key:
        url: https://repos.influxdata.com/influxdb.key
        state: present

    - name: Add InfluxDB repository
      become: true
      apt_repository:
        repo: deb https://repos.influxdata.com/ubuntu focal stable
        state: present
```
~~~

In both tasks, you need to `become` root to accomplish the task. We use the package for the "Focal" distro. The rest is fairly self-explanatory.

Next, we install InfluxDB:

~~~~
```

     - name: Ensure InfluxDB is installed

       become: true

       apt:

         name: influxdb

         state: present

```
~~~~

We use the `apt` module here but you can also use `package` the same way.

Next we run the service:

~~~~
```

     - name: Run InfluxDB

       become: true

       systemd:

         name: influxdb

         state: started

```
~~~~

We use the `systemd` module here which runs InfluxDB as root. If we wanted to run InfluxDB as another user (i.e., the AWS EC2 default Ubuntu user, `ubuntu`), we have to include a task that gives the service permission to read and write the necessary data and metadata files. We won't include those tasks here in the interest of avoiding bloat but the directories necessary are, by default, in `/var/lib/influxdb/`.

The entire deployment play, which we're calling `install.yaml`, should look like this:

~~~~
```

---


- name: Install InfluxDB

  hosts: all
~~~~

```
  gather_facts: false
  remote_user: ubuntu

  tasks:

    - name: Import InfluxDB GPG signing key
      become: true
      apt_key:
        url: https://repos.influxdata.com/influxdb.key
        state: present

    - name: Add InfluxDB repository
      become: true
      apt_repository:
        repo: deb https://repos.influxdata.com/ubuntu focal stable
        state: present

    - name: Ensure InfluxDB is installed
      become: true
      apt:
        name: influxdb
        state: present

    - name: Run InfluxDB
      become: true
      systemd:
        name: influxdb
        state: started
```

~~~
```
~~~

That’s it for a basic deployment of InfluxDB. For many users, this will suffice. That said, we covered configuration in a previous chapter so that users can create custom configurations for their instances. In addition, this chapter presents what an Ansible configuration could look like.

Configure InfluxDB

As discussed in the configuration chapter, there are multiple ways to configure InfluxDB. You can do so with a config file, runtime flags, and/or environment variables. We’ll cover configuration via a file here because it lends itself best to Ansible semantics.

Against your remote hosts, you’ll want something to the effect of (complete file):

~~~
```
---
- name: Install InfluxDB
  hosts: all
  gather_facts: false
  remote_user: ubuntu
  vars_files: defaults.yaml

  tasks:

   -  name: Set config path
      become: true
      file:
        path: “{{ influxdb_configuration _dir }}”
        state: directory

   -  name: Set configuration
      become: true
      template:
~~~

```
          src: influxdb.conf.j2

          dest: "{{ influxdb_configuration_dir }}/influxdb.conf"

          force: yes

          owner: influxdb

          group: influxdb

          mode: 0744

        when: influxdb_template_configuration

```
```

While this is straightforward, we make use of some files external in this playbook. The linked playbook at the beginning (and end) of this chapter has all the necessary files for you to work with. For now, note that this uses a file called `defaults.yaml` and sets it as the vars for the play to use. Then the play references an `influxdb.conf.j2` file that is a Jinja2 template where we inject values from the defaults.yaml file. You can use the values to change configuration settings for your deployed InfluxDB nodes, using the information in the "Configuration" chapter.

Once you have InfluxDB nodes deployed and configured, we can incorporate some instance setup into the Ansible procedure as well! InfluxDB has a setup API that will initialize your instance with an admin/"operator" user. You can also initialize instances with other things like dashboards, Flux tasks, etc. Below, we'll set up the instances like normal and include a configured replication and Flux task.

Note: The Flux task is not a recommended task. It's simply a placeholder to showcase initializing one with an Ansible play.

Setup InfluxDB

This portion is actually less Ansible and more Python but we show the Ansible portion because we use it to run the Python script. The included Python script is merely an example because we can't know exactly how users will want to set up their instances. The playbook repo contains the Python script necessary for this example so you don't need to doany coding.

The Ansible looks like this:

```
```

- name: Setup InfluxDB instances
```

```
  hosts: localhost
  gather_facts: no
  vars_files:
    - vars.yaml

  tasks:
    - name: Setup influxdb instance
      command: |
       ./main.py --influxdb-host {{ item }} --username {{ influx_username }}
--password {{ influx_password }} --organization {{ influx_org }} --bucket {{
influx_bucket }} \
       --cloud-host {{ influx_remote_host }} --cloud-org {{
influx_remote_org_id }} --cloud-token {{ influx_remote_token }} --cloud-bucket
{{ influx_remote_bucket }} \
       --task-file {{ task_file_path }}
     with_items: "{{ groups['all'] }}"

```

This play uses the `command` module, indicated in green, to target the control host and run the local Python script, `main.py`, which does the following:

- Hits the InfluxDB `/setup` API and creates the first user, who will have full admin rights
- Creates a remote connection to a cloud account
- Creates a replication stream of a bucket from the target node to that remote cloud account
- Creates a task from a file also provided in the repo (for example)
    - This task only works if you have “cpu” data (i.e., via Telegraf) but you can modify the task to do whatever you’d like.
    - If you’re unfamiliar with Flux, you can start to learn it for free at InfluxDB University. The complementary Time To Awesome book is also a great resource.

The `vars.yaml` file injects all the flag values in the command (like in the Configuration section) . The repo provides a blank version of this for you to fill in with your own values.

This chapter on Ansible and InfluxDB is more descriptive than prescriptive wherever possible because it’s unlikely you will run in production with exactly this set of plays. Nevertheless, it should provide an idea of how to customize this set or create your own.

Chapter 7

Monitoring

InfluxDB provides 3 data acquisition tools for collecting InfluxDB OSS performance, resource, and usage metrics:

1. Telegraf Prometheus Input Plugin.
2. InfluxDB scrapers.
3. Flux prometheus.scrape() function.

This chapter covers the first two solutions because they are the easiest way to monitor your InfluxDB OSS instance(s). The Telegraf Prometheus Input Plugin collects Prometheus metrics from a HTTP(s)-accessible endpoint, but offers all of the advantages and features of the Telegraf agent. InfluxDB scrapers collect Prometheus metrics from any HTTP(s)-accessible endpoint at 10 second intervals. The Flux Prometheus package includes functions that allow you to collect Prometheus metrics as well, but you need to create a task to regularly collect those Prometheus metrics.

The following suggested approaches leverage the Telegraf Prometheus Input Plugin and InfluxDB scraper tools, respectively.

1. InfluxDB Open Source (OSS) Metrics Template.

A template is a prepackaged InfluxDB configuration that contains multiple InfluxDB resources. This template includes (but is not limited to):

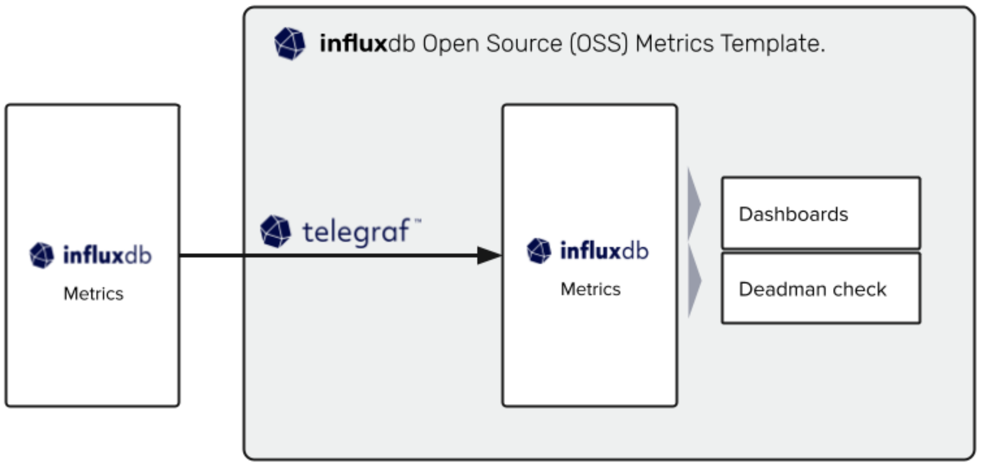
- A Telegraf Prometheus input plugin configuration that collects metrics from an InfluxDB OSS instance and writes them to a Free Tier InfluxDB Cloud account.
- A Dashboard, InfluxDB OSS Metrics, with some basic visualizations to start monitoring your InfluxDB OSS instance.
- A InfluxDB OSS Deadman check to check for data outages in your OSS instance.

We designed the template to quickly setup monitoring for your instance(s). You are free to modify and add to the dashboard, create new checks and notifications, and incorporate any other resources you find useful.

The primary advantages of this approach are:

- Ease of use and familiarity. You can just install the template and get going. If you're looking to monitor InfluxDB, you're likely writing data to InfluxDB with Telegraf alread.

- ○ Telegraf offers a lot of features for data collection like batching, caching, buffering, and more.



2. InfluxDB scrapers and [Edge Data Replication]() (EDR).

This approach requires you to:

- ○ Set up an InfluxDB scraper.
- ○ Write InfluxDB OSS metrics to a new bucket.
- ○ Use the Edge Data Replication feature to send the data to a Free Tier InfluxDB Cloud account.
- ○ Create a dashboard and task(s) in InfluxDB Cloud to monitor the health of the InfluxDB OSS instance and receive alerts.

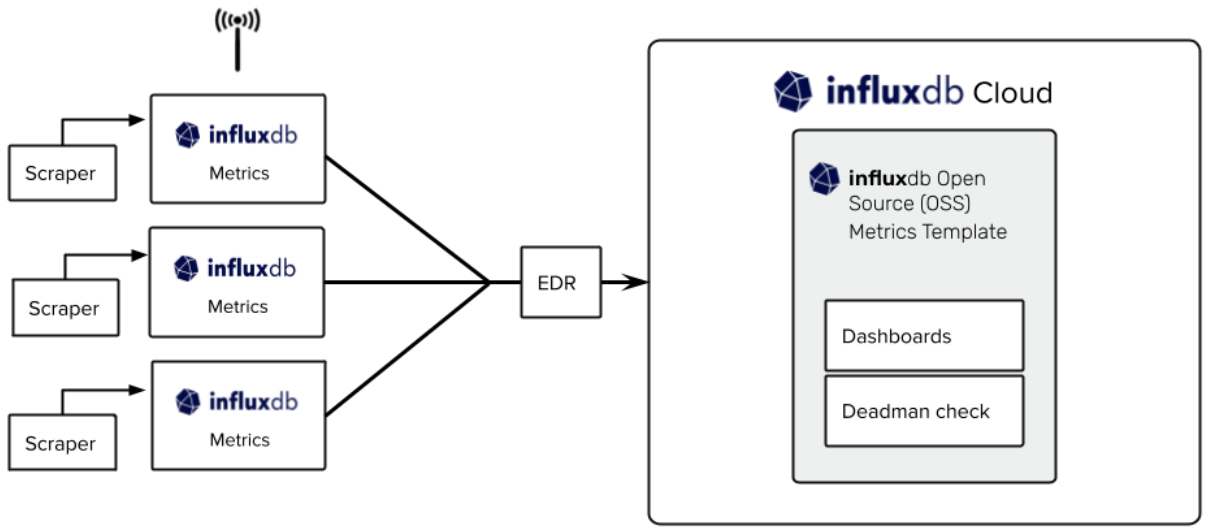
Optional approach:

- ○ Use the InfluxDB Open Source (OSS) Metrics Template in addition to the InfluxDB scraper and EDR feature to install the preconfigured dashboard and task for those metrics.
- ○ When using Edge Data Replication, make sure to write data to a bucket called oss_metrics. This is the bucket that the dashboard and deadman check queries.

The primary advantages of this approach are:

- ○ InfluxDB Scrapers allow you to collect performance metrics from your InfluxDB OSS instances without the need to host or maintain any Telegraf agents

    - This approach takes advantage of the Edge Data Replication (EDR). EDR is perfect for IoT use cases, where you have hundreds of InfluxDB instances running at the edge or fog and want to monitor those.



Metrics Deep Dive

The InfluxDB OSS metrics endpoint provides the following performance, resource, and usage Prometheus metrics types:

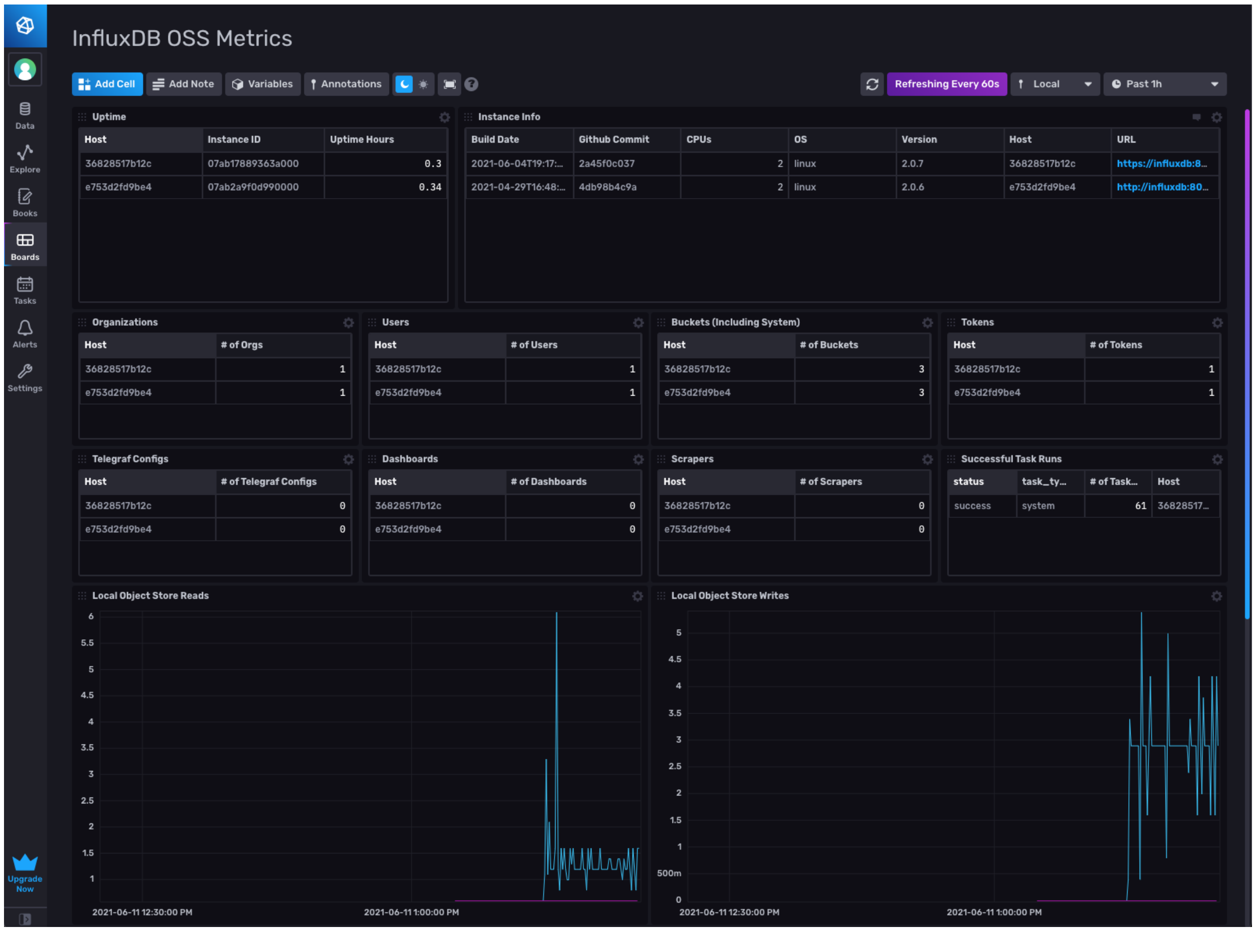
- Boltdb
- Go runtime
- HTTP API
- InfluxDB objects and queries
- QC (query controller)
- InfluxDB services
- InfluxDB storage
- InfluxDB tasks

Here is a summary of the most important metrics to focus on, what they are, and what they indicate about your InfluxDB OSS instance:

Metric Name	Description	Indicator

Visualization and Alerting Best Practices

While the InfluxDB Open Source (OSS) Metrics Template includes a Telegraf configuration to collect and write InfluxDB OSS metrics, you don't have to use Telegraf to take advantage of the preconfigured dashboard and deadman check. The preconfigured dashboard, InfluxDB OSS Metrics, contains some basic visualizations to help you monitor your InfluxDB OSS instance.



InfluxDB OSS Metrics

Uptime

Host	Instance ID	Uptime Hours
36828517b12c	07ab17889363a000	0.3
e753d2fd9be4	07ab2a9f0d990000	0.34

Instance Info

Build Date	Github Commit	CPUs	OS	Version	Host	URL
2021-06-04T19:17:...	2a45f0c037	2	linux	2.0.7	36828517b12c	https://influxdb:8...
2021-04-29T16:48:...	4db98b4c9a	2	linux	2.0.6	e753d2fd9be4	http://influxdb:80...

Organizations

Host	# of Orgs
36828517b12c	1
e753d2fd9be4	1

Users

Host	# of Users
36828517b12c	1
e753d2fd9be4	1

Buckets (Including System)

Host	# of Buckets
36828517b12c	3
e753d2fd9be4	3

Tokens

Host	# of Tokens
36828517b12c	1
e753d2fd9be4	1

Telegraf Configs

Host	# of Telegraf Configs
36828517b12c	0
e753d2fd9be4	0

Dashboards

Host	# of Dashboards
36828517b12c	0
e753d2fd9be4	0

Scrapers

Host	# of Scrapers
36828517b12c	0
e753d2fd9be4	0

Successful Task Runs

status	task_ty...	# of Task...	Host
success	system	61	36828517...

A screenshot of the InfluxDB OSS Metrics dashboard from the InfluxDB Open Source (OSS) Metrics Template.

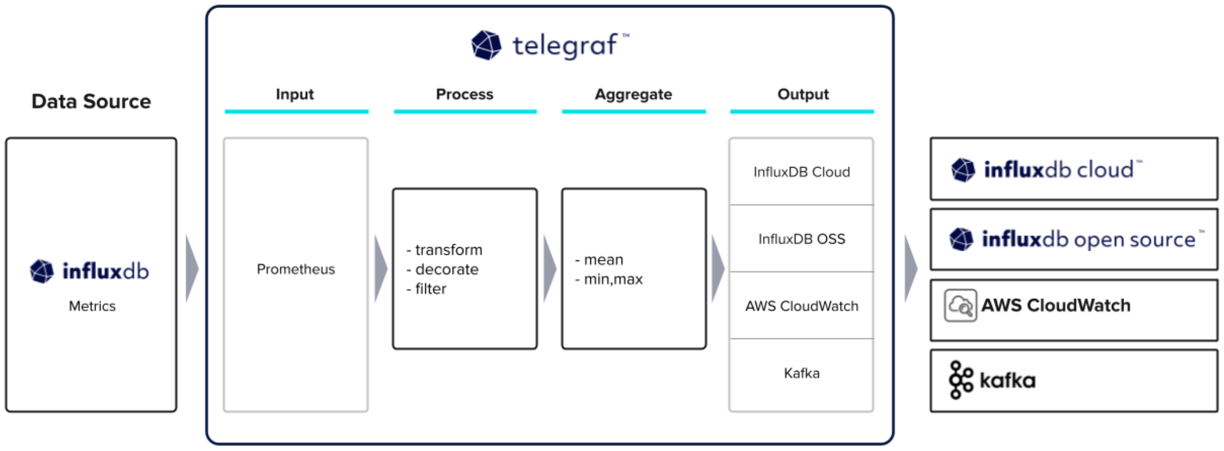
To view the corresponding Flux query for any of the cells in the dashboard, click the gear icon on the upper right corner of the cell. The following table is a summary of each cell from left to right, top to bottom. Each cell contains metrics data for each InfluxDB host:

Cell Name	Description	Purpose
Uptime	Queries for influxdb_uptimes_seconds, grouped by host, and calculated on an hourly basis.	For identifying hosts that experience an outage. All hosts should have the same Uptime Hours value.

Instance Info	Lists the build date, github commit, CPU's, OS, version, host, and URL.	For providing more context about the InfluxDB instances at a glance.
Organizations	Lists the number of organizations per host.	
Users	Lists the number of users per host.	
Buckets (Including System)	Lists the number of buckets per host.	
Telegraf Configs	Lists the number of tokens per host.	
Dashboards	Lists the number of dashboards per host	
Scrapers	Lists the number of scrapers per host	
Local Object Store Reads	Queries for boltdb_reads_total, grouped by host.	For monitoring query performance and guard against I/O contention.
Local Object Store Writes	Queries for boltdb_writes_total, grouped by host.	

Telegraf

We can use Telegraf to scrape monitoring metrics from InfluxDB. The benefit of using Telegraf for this is that it allows you to send monitoring metrics directly to another InfluxDB instance or to send them to a completely different platform/endpoint. The next section provides the building blocks to create a Telegraf config. A full version of the config is available here.



Prometheus Input Plugin

InfluxDB exposes health metrics via a HTTP(s)-accessible endpoint:

``http://<INFLUXDB_HOST_ADDRESS>:8086/metrics``

All metrics exposed by this endpoint are written using the Prometheus data model. We use the Prometheus Input Plugin to collect metrics from this endpoint. Because the Prometheus input plugin is designed to collect data from any prometheus endpoint, this guide highlights the significant plugin parameters.

~~~
```TOML
# Read metrics from one or many prometheus clients
[[inputs.prometheus]]
  ## An array of urls to scrape metrics from.
  urls = ["http://<INFLUXDB_HOST_ADDRESS>metrics"]

  ## Metric version controls the mapping from Prometheus metrics into
  ## Telegraf metrics. When using the prometheus_client output, use the same
  ## value in both plugins to ensure metrics are round-tripped without
  ## modification.
  ##
  ##   example: metric_version = 1; deprecated in 1.13
~~~

```
  ##            metric_version = 2; recommended version
  metric_version = 1

  ## Optional TLS Config
  # tls_ca = /path/to/cafile
  # tls_cert = /path/to/certfile
  # tls_key = /path/to/keyfile
  ## Use TLS but skip chain & host verification
  # insecure_skip_verify = false
```

The break down:

`urls` - This parameter is where you specify the InfluxDB metrics endpoint. Note: You can specify multiple URLs within the square brackets to monitor several InfluxDB instances simultaneously `(more on this later).`

```TOML
  urls = ["http://192.168.1.220:8086/metrics",
        "http:/host1:8086/metrics",
        "http://host2:8086/metrics"]
```

metric_version - The prometheus plugin provides two conversions of the Prometheus Data model into line protocol. It is now widely accepted and recommended to use **version 2** but the InfluxDB Open Source (OSS) Metrics Template still requires you to use **version 1**. This guide doesn't cover the differences between the conversions, but you can find an explanation here.

Optional TLS Config - Lastly if you host and run InfluxDB with TLS enabled then Telegraf provides parameters for configuring plugin certificates. In most cases, when using self-signed certificates, set *insecure_skip_verify* to **true.**

```TOML
insecure_skip_verify = true
```

Output Plugins

After collecting the metrics using the Prometheus input plugin you need to send these monitoring metrics somewhere. One of the primary destinations is another remote instance of InfluxDB, such as InfluxDB Cloud. To do this, use the InfluxDB v2.x Output Plugin. Here are the output plugins parameters abstracted from the InfluxDB Open Source (OSS) Metrics Template:

~~~
```TOML

# # Configuration for sending metrics to InfluxDB

[[outputs.influxdb_v2]]

  ## The URLs of the InfluxDB cluster nodes.

  ##

  ## Multiple URLs can be specified for a single cluster, only ONE of the

  ## urls will be written to each interval.

  ##   ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]

  urls = ["$INFLUX_URL"]


  ## Token for authentication.

  token = "$INFLUX_TOKEN"


  ## Organization is the name of the organization you wish to write to; must
exist.

  organization = "$INFLUX_ORG"


  ## Destination bucket to write into.

  bucket = "oss_metrics"


  ## The value of this tag will be used to determine the bucket.  If this

  ## tag is not set the 'bucket' option is used as the default.

  # bucket_tag = ""


  ## If true, the bucket tag will not be added to the metric.
~~~

```
  # exclude_bucket_tag = false

  ## Optional TLS Config for use on HTTP connections.
  # tls_ca = "/etc/telegraf/ca.pem"
  # tls_cert = "/etc/telegraf/cert.pem"
  # tls_key = "/etc/telegraf/key.pem"
  ## Use TLS but skip chain & host verification
  # insecure_skip_verify = false
```

The breakdown:

urls - The URL(s) of the InfluxDB node(s) you want to send the monitoring metrics to.

token - This parameter requires you to generate an authentication token for the remote InfluxDB node. We recommend that you generate a write only token because Telegraf requires no other privileges to your InfluxDB instance.

organization - The name of the organization within your remote instance of InfluxDB

bucket - The destination bucket name for your metrics. Optionally, you can use a **bucket_tag** instead. This allows you to designate a metric tag to control which bucket your data is written to. This is useful when you are collecting metrics from multiple InfluxDB nodes and want to write each node's health metrics to a different bucket.

```TOML
  ## The value of this tag will be used to determine the bucket.  If this
  ## tag is not set the 'bucket' option is used as the default.
  bucket_tag = "host"

  ## If true, the bucket tag will not be added to the metric.
  # exclude_bucket_tag = false
```

In this case Telegraf writes metrics to buckets where the bucket name matches the host value.

Optional TLS Config - Lastly if you host and run InfluxDB with TLS enabled then Telegraf provides parameters for configuring plugin certificates. In most cases, when using self-signed certificates, set *insecure_skip_verify* to **true.**

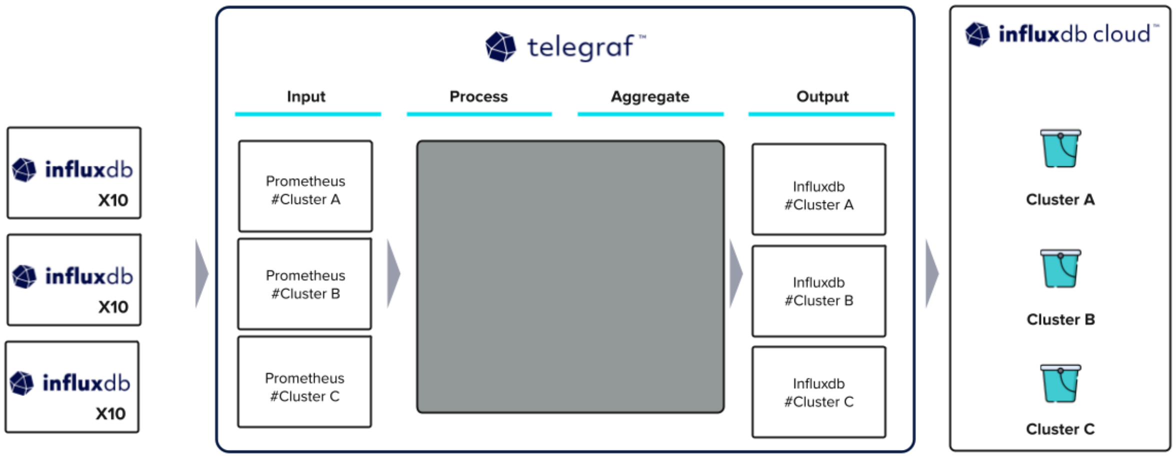
The InfluxDB v2.x Output Plugin is one of 34 output plugins that allow you to customize where you send monitoring metrics. We cannot cover every output plugin in this guide, but here are some popular endpoints to investigate further:

1. Amazon CloudWatch
2. Apache Kafka
3. Datadog
4. Google Cloud PubSub
5. Microsoft Azure Monitor

Best Practices

Telegraf is a highly performant agent with a minimal system footprint. When monitoring only one InfluxDB node, using Telegraf's default parameters is more than sufficient. When monitoring multiple nodes with one Telegraf agent there are some agent-level and plugin parameters to keep in mind:

- Divide and Conquer: Although both the Prometheus input and InfluxDB plugins can read and write to multiple endpoints it may be more effective to duplicate plugins. For example, suppose you need to monitor 30 instances of InfluxDB. You can divide data collection between 3 identical input plugins, with the difference being that each plugin is responsible for monitoring 10 specific nodes. As a general scalability rule, this approach plays well with other Telegraf best practices.



- **Interval, flush and jitter:** It is worth noting that the monitoring metrics count is quite sizable (avg. 492 metrics per default flush interval). When collecting from multiple nodes this can impact your network bandwidth. To help reduce bottlenecks you can specify interval and flush times at the plugin level.

  ~~~
  ```TOML
  # Read metrics from one or many prometheus clients
  [[inputs.prometheus]]
  interval = 10s
    urls = ["http://server1:8086/metrics"]
    metric_version = 1


  [[inputs.prometheus]]
  interval = 15s
    urls = ["http://server2:8086/metrics"]
    metric_version = 1
  ```
  ~~~

  You can also do this at the agent level with collection jitter. In this approach, each plugin sleeps for a random time within the jitter range before collecting data.

  ~~~
  ```TOML
  [agent]
  collection_jitter = "5s"
  [[inputs.prometheus]]
    urls = ["http://server1:8086/metrics"]
    metric_version = 1


  [[inputs.prometheus]]
    urls = ["http://server2:8086/metrics"]
    metric_version = 1
  ```
  ~~~

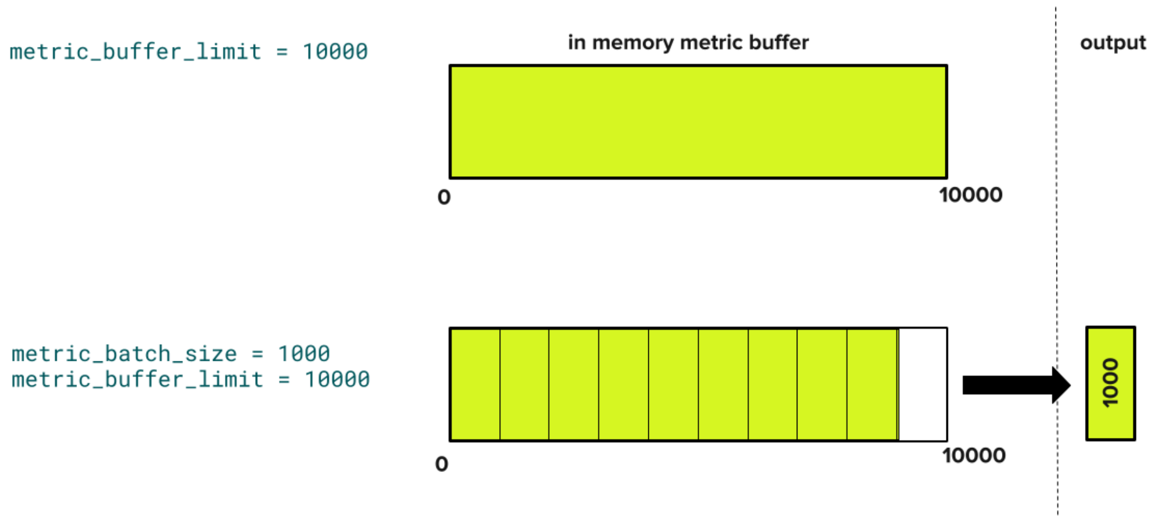
  Telegraf also provides the same amount of granular control in its output plugins. Suppose you send monitoring metrics to InfluxDB for storage and to AWS Cloudwatch for other workloads. You can deploy flush jitter at the agent level to stagger data delivery from each output plugin.

~~~
```TOML
[agent]
collection_jitter = "5s"
flush_jitter = “5s”
[[inputs.prometheus]]
  urls = ["http://server1:8086/metrics"]
  metric_version = 1


[[inputs.prometheus]]
  urls = ["http://server2:8086/metrics"]
  metric_version = 1
```
~~~

You can also control the flush interval at the plugin level.

- **Metric buffer limit & max batch size:** Telegraf has an in-memory buffer. All collected metrics enter the buffer before being “flushed” to their relevant outputs. Buffering provides limited durability because Telegraf keeps metrics in the buffer for as long as the Telegraf agent is alive or until the buffer reaches its defined maximum. Controlling the buffer limit and max batch size are subjective to your architecture needs. For instance, if you have a weak or unpredictable network connection and can afford a larger Telegraf memory footprint, we recommend setting a higher buffer limit. For networks with limited bandwidth you may also want to define a smaller batch size. Batching governs how many metrics Telegraf sends to the output plugin at one time. For example, if your batch size is set to 20, but you collect 30 metrics per interval, then Telegraf automatically sends the first 20 metrics to the output endpoint(s) because it already reached the maximum batch size. The next batch contains the remaining 10 metrics. Triggering the flush interval empties the entire buffer in batches corresponding to the buffer limit. For example, a buffer of 80 releases 4 successive batches of 20 metrics each.



- **Sensitive configuration parameters:** Storing credential information within a config is an important security feature to consider. One of the best ways to mitigate risk is to use environment variables as part of your best practice. This provides both greater scope to apply additional security best practices and further flexibility in your Telegraf config.

  ~~~
  ```TOML
  [[outputs.influxdb_v2]]
  alias = "k8"
    urls = ["${INFLUX_HOST}"]
    token = "${INFLUX_TOKEN}"
    organization = "${INFLUX_ORG}"
    bucket = "${INFLUX_BUCKET}"
  ```
  ~~~

  The example above anonymizes each parameter connecting to the InfluxDB instance with an environment variable. Note: You could also deploy a scripting log to change these parameters reactively and to automatically restart the Telegraf service based upon a desired system state change.

These best practices provide the foundation to effectively scale Telegraf within your system architecture. Other features to consider with Telegraf include routing and monitoring the Telegraf agent's performance, but detailed explanations of these features is beyond the scope of this guide. You can find more information on these subjects here.

InfluxDB Scrapers

The primary advantage to using InfluxDB Scrapers to collect performance metrics from your InfluxDB OSS instances is that you don't need to host or maintain any Telegraf agents. Instead, you need to set up an Edge Data Replication (EDR) stream to write this data to InfluxDB Cloud. You could just write performance metrics to OSS and forego the EDR stream, but this type of self monitoring has limitations. The primary limitation is that if your OSS instance becomes unavailable, then your monitoring pipeline will, too. You also won't have access to the recent monitoring data for incident discovery and reporting.

There are multiple ways to create a scraper with InfluxDB:

- Through the UI
- With the InfluxDB OSS v2 API
- With Flux

For all of these methods, we'll assume that you're running InfluxDB on http://localhost:8086/. You must also create a destination bucket to write the Prometheus metrics to. Some of these methods require the following information:

- An all access token
- Your organization ID
- Your destination bucket ID

To create a scraper through the InfluxDB UI follow these steps:

1. Navigate to the **Scrapers** tab under the **Data** page. Navigate to the **Data** page from the second option on the left hand navigation bar.
2. Click **+Create Scraper**, name your scraper, select the bucket and the Target URL that you want to write data to. By default the Target URL will be `http://localhost:8086/metrics`.
3. Click **Create**.

To create a scraper with the InfluxDB v2 OSS API execute the following cURL request:

~~~
```cURL
curl -X 'POST' \
  'http://localhost:8086/api/v2/scrapers' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Token <all access token>' \
~~~

~~~
  -d '{"bucketID": "<bucketID>")",  "name": "<example-scraper-name>", "orgID":
"<orgID>", "url": "http://localhost:8086/metrics", "allowInsecure":
"false","type":  "prometheus"}'
~~~

To create a scraper with Flux execute the following query:

~~~
```Flux
import "array"

import "http"

import "json"


http.post(

  url: "http://localhost:8086/api/v2/scrapers",

  headers: {Accept: "application/json", Authorization: "Token <all access
token>", "Content-Type": "application/json"},

  data: json.encode(v: {

  "bucketID": "<bucketID>")",

  "name": "<example-scraper-name>",

  "orgID": "<orgID>",

  "url": "http://localhost:8086/metrics",

  "allowInsecure": "false",

   "type":  "prometheus"})

)

```
~~~

Verify that you're successfully scraping data by executing the following query:

~~~
```Flux
from(bucket: "<destination bucket>")

  |> range(start: -10m)

  |> filter(fn: (r) => r["_measurement"] == "influxdb_uptime_seconds")

// obtaining the last value because we want to know how long InfluxDB has been
running
~~~

```
  |> last()
// optional convert the seconds to minutes with the following line
  |> map(fn: (r) => ({ r with _value: r._value/60.0 }))
  |> yield(name:  "uptime in minutes")
```
```

## Edge Data Replication

After successfully creating a scraper, use the Edge Data Replication (EDR) feature to replicate data from your OSS instance to a Free Tier InfluxDB Cloud account to receive alerts on poor performance or outages. If you want to take advantage of the dashboards and tasks in the InfluxDB Open Source (OSS) Metrics Template to your Cloud instance, install the template first. Then use the oss_metrics bucket as your destination Cloud bucket for EDR. You can also delete the Telegraf configuration. Follow these steps to configure a replication stream:

1. Download and install the Influx CLI.
2. Create an OSS CLI configuration with:

   ```
   ```bash
   influx config create --active \
     -n <example-config-name> \
     -u <OSS URL>\
     -t <OSS auto generated all access token>\
     -o <OSS org>
   ```
   ```

   **IMPORTANT NOTE:** To create an EDR stream you must use a OSS CLI config with the auto-generated all access token. It will have the name “<org>’s Token” and be at the bottom of the **Tokens** page. This “operator token” is created automatically during setup. You will not be able to create an EDR stream with a new all access token. You must reinstall and set up InfluxDB to obtain an original token if you’ve accidentally deleted it.
3. Verify that you have a CLI config by listing your CLI configs with:

   ```
   ```
   influx config ls
   ```
   ```
4. Verify that your connection is OK with:
   ```

```
influx ping
```

5. Create a remote CLI connection with:

   ```bash
   influx remote create \
     --name <example-remote-name> \
     --remote-url <Cloud URL e.g.
   https://us-west-2-1.aws.cloud2.influxdata.com> \
     --remote-api-token <Cloud token> \
     --remote-org-id <Cloud orgID>
   ```

6. List the remote connections to obtain the remote-id:

   ```bash
   influx remote ls \
   ```

7. Create a replication stream with:

   ```bash
   influx replication create \
     --name <example-replication-stream-name>\
     --remote-id <gathered in step 6>\
     --local-bucket-id <OSS bucketID to replicate> \
     --remote-bucket-id <destination Cloud bucketID>
   ```

Verify that you're successfully replicating data to InfluxDB Cloud by executing the following query:

```Flux
from(bucket: "<destination Cloud bucket>")
  |> range(start: -10m)
  |> filter(fn: (r) => r["_measurement"] == "influxdb_uptime_seconds")
```

~~~
// obtaining the last value because we want to know how long InfluxDB has been running

  |> last()

// optional convert the seconds to minutes with the following line

  |> map(fn: (r) => ({ r with _value: r._value/60.0 })

  |> yield(name:  "uptime in minutes")

```
~~~

Now you can set up a notification endpoint as described here and receive alerts on the performance and availability of your OSS instance.

Chapter 8

Conclusion

If you're an engineer tasked with maintaining the infrastructure and service deployments of an InfluxDB OSS 2.x, hopefully this OSS Operational Guide answered all of your questions pertaining to:

- Basic installation
- Hardware sizing recommendations
- Platform optimizations
- Deployment
- Service runtime tools
- Self monitoring
- Troubleshooting
- Backup and Restore

If you need additional support, reach out for help on our community site or Slack channel. Finally, you might be interest in these additional resources to learn more about using InfluxDB:

- InfluxDB Documentation: the official documentation for InfluxDB.
- InfluxDB University: A catalog of free training to help you gain skills using InfluxDB.
- Time to Awesome: A book on InfluxDB–helping IoT application developers build on top of InfluxDB and experience time to awesome.
- InfluxDB Blog: Blog posts highlighting popular use cases and how-to guides.

InfluxDB documentation, downloads & guides

[Get InfluxDB]

[Try InfluxDB Cloud for Free]

[Get documentation]

[Additional tech papers]

[Join the InfluxDB community]