

# How to Securely Deploy QuestDB on a Vultr Cloud Server

Learn step-by-step instructions for securely deploying QuestDB on a Vultr Cloud Server, including setup, configuration, and best security practices.

---

# Contents

01	Introduction	3
02	Prerequisites	3
03	Instal QuestDB	3
04	Basic Usage of QuestDB	6
05	Accessing the QuestDB Database using a Programming Language	8
06	Configuration of QuestDB	10
07	More Resources	12

# Introduction

---

QuestDB is a time-series database application that is fast and suitable for digesting large data. A time-series database is a database that is specialized for keeping and managing data indexed by time order. Data can be sensor data, stock prices, and so on. It's similar to [InfluxDB](#) and TimescaleDB.

Using one example, the sensor data might be the temperature in your house measured every day in one year. So there's a time component in it. Although you can keep the time-series data in a normal database like PostgreSQL, a time-series database like QuestDB is optimized for this use case.

This guide explains how to install and deploy QuestDB on Ubuntu 22.04.

## Prerequisites

---

- Deploy a [Vultr Cloud Compute](#) running on Ubuntu 22.04.
- Create a non-root user with sudo access and log in with SSH.
- [Update](#) the Ubuntu Server.
- Install [Curl](#).

## Instal QuestDB

---

QuestDB is a Java application. To run it, you need Java Virtual Machine (JVM) version 11. But you don't have to install JVM separately. There is a QuestDB binary package that shipped with JVM.

1. Download and extract the QuestDB binary package that has JVM.

```
$ wget https://github.com/questdb/questdb/releases/download/6.6.1/questdb-6.6.1-rt-linux-amd64.tar.gz
$ tar -zxvf questdb-6.6.1-rt-linux-amd64.tar.gz
```



```
hs_err_pid+%p.log -XX:+UnlockExperimentalVMOptions -XX:+AlwaysPreTouch -XX:
+UseParallelGC -p ./bin/questdb.jar -m io.questdb/io.questdb.ServerMain -d /home/
your_username/.questdb
```

As you can see the PID is the same as the one you saw previously.

You can also run QuestDB with Docker. But first, you must install Docker.

```
$ sudo apt install docker.io -y
```

Before running QuestDB with Docker, stop the QuestDB server first.

```
$ ./bin/questdb.sh stop
```

Then you can run QuestDB with Docker like this.

```
$ sudo docker run -p 9000:9000 -p 9009:9009 -p 8812:8812 -p 9003:9003 -v "$(pwd):/var/
lib/questdb" questdb/questdb:6.6.1
```

Unlike when you run it with the binary package, with Docker the QuestDB server runs in the foreground. So to stop it, you can Ctrl+C.

By default, the QuestDB application uses four ports whether you run it with Docker or directly with the binary package. The ports that QuestDB uses are port 9000, 9009, 8812, and 9003.

Port 9000 is for the REST API and web console. Web console is a web interface to interact with QuestDb.

Port 9009 is for InfluxDB line protocol. You can insert data to QuestDB like you do it in InfluxDB.

Port 8812 is for Postgres wire protocol. You can interact with Quest like you interact with PostgreSQL.

Port 9003 is a minimal health server. Any HTTP request that sent to this port will get the 200 status request unless the database server dies.

You don't have to open all ports. You may want to interact with QuestDb with REST API only. So you can only open port 9000.

The Docker command also mounts the `/var/lib/questdb` directory to your current directory. It's useful for editing the configuration file later.

## Basic Usage of QuestDB

The most friendly UI to connect to QuestDB is Web Console which runs on port 9000. It's a web application where you can type SQL commands on the browser.

Run the QuestDB server again.

```
$ ./bin/questdb.sh start
```

Then you can open the port 9000 to the public with [the iptables command](#).

```
$ sudo /sbin/iptables -I INPUT -p tcp -m tcp --dport 9000 -j ACCEPT
```

This is for convenience only so you can follow this tutorial. But you should protect your QuestDB server properly. You can use Nginx as a proxy with an authentication. You can also [setup firewall rules on Vultr](#).

When you open the web application, there is a text field. For this tutorial, you'll create dummy data of cryptocurrency prices for a year.

Type this SQL command in the field and click the `Run` button.

```
create table cryptocurrencies (id int, name string);
```

Then you insert some cryptocurrencies into the table.

```
insert into cryptocurrencies (id, name) values  
(1, 'Bitcoin'), (2, 'Ethereum');
```

Most of the time, you can use SQL commands but QuestDB has some commands that are not available in standard SQL databases like PostgreSQL. One of them is to get all rows from a table.

```
'cryptocurrencies';
```

Then you should create a table of the price and its data. Type this SQL command to generate dummy data.

```
create table prices as (  
  
SELECT x id,  
timestamp_sequence(to_timestamp('2022-01-01T00:00:00', 'yyyy-MM-ddTHH:mm:ss'),  
3600000000L * 24) as ts,  
rnd_long(10000, 20000, 0) AS price,  
1 as cryptocurrency_id  
from long_sequence(365) x  
  
UNION  
  
SELECT x id,  
timestamp_sequence(to_timestamp('2022-01-01T00:00:00', 'yyyy-MM-ddTHH:mm:ss'),  
3600000000L * 24) as ts,  
rnd_long(800, 1800, 0) AS price,  
2 as cryptocurrency_id  
from long_sequence(365) x  
  
)  
timestamp(ts) partition by month;
```

The SQL command above is not valid in PostgreSQL. Although there is the Postgres wire protocol you can use, that doesn't mean you can use all PostgreSQL commands. For example, the `generate_series` syntax from PostgreSQL is not available on QuestDB. The `rnd_long` syntax from QuestDB is not available on PostgreSQL.

This command created 365 rows of random prices for Bitcoin and another 365 rows for Ethereum. The `3600000000L * 24` argument in `timestamp_sequence` means 1 day. It's a step in `timestamp_sequence` in microseconds. The `rnd_long` function generates a random number between the first argument and the second argument. The third argument is a null rate or a rate to get a NaN value in your random function. You didn't use the third argument's functionality. The `partition` part in the SQL command is useful for performance. Here, you use the month partition. It means data for each month will be stored in different files.

After executing this command, you'll get 730 rows.

You can find the average price of Bitcoin with this SQL command.

```
select avg(price) from prices where cryptocurrency_id = 1;
```

Then you can find the maximum price for each cryptocurrency with this SQL command.

```
select name, max(price)
from 'prices'
join(
  select id, name from 'cryptocurrencies'
) c
on prices.cryptocurrency_id = c.id;
```

If you want to access QuestDB with REST API, you can use `curl`. The port is still the same, 9000. But the endpoint is different. It's `/exec`.

```
$ curl -G --data-urlencode "query=select avg(price) from prices where cryptocurrency_id = 2;" http://<your_server_ip_address>:9000/exec
```

The `-G` option means putting the query in the URL as the parameter of the GET request. The parameter is the query string. But you have to encode it first with the `--data-urlencode` option.

You'll get this JSON data.

```
{"query":"select avg(price) from prices where cryptocurrency_id = 2;","columns":
[{"name":"avg","type":"DOUBLE"}],"dataset":[[1326.767123287671]],"count":1}
```

## Accessing the QuestDB Database using a Programming Language

To access QuestDB with a programming language like Python, you can use QuestDB as if you use PostgreSQL with a catch. Some PostgreSQL commands

don't work in QuestDB, and vice versa. It means you'd better use raw SQL instead of Object Relational Mapping (ORM) like SQLAlchemy. So the best option is to use a library like `psycopg2` if you use Python.

```
$ sudo apt update
$ sudo apt install python3-venv -y
$ cd ~/
$ python3 -m venv questdb-venv
$ source questdb-venv/bin/activate
(questdb-venv) $ pip install psycopg2-binary
```

Then you can write a script to insert data into the database you're still running. Create a file named `insert_data.py` and add the following content to it.

```
import psycopg2 as pg

with pg.connect(
    user="admin",
    password="quest",
    host="127.0.0.1",
    port="8812",
    database="you_can_write_anything_here") as connection:
    with connection.cursor() as cur:

        id = 3
        name = "Dogecoin"
        cur.execute('insert into cryptocurrencies values (%s, %s);', (id, name))

        print("Added Dogecoin")
```

Run the script.

```
(questdb-venv) $ python insert_data.py
```

Your script accessed QuestDB with the Postgres wire protocol. So it used port 8812, not 9000. By default, the user and the password are `admin` and `quest`. But you'll learn to change it in the next section covering the configuration of QuestDB.

QuestDB doesn't have a database name so you can put anything for the database name field.

To update data, create a script named `update_data.py` and add the following content to it.

```
import psycopg2 as pg

with pg.connect(
    user="admin",
    password="quest",
    host="127.0.0.1",
    port="8812",
    database="qdb") as connection:
    with connection.cursor() as cur:

        id = 3
        name = "BNB"
        cur.execute('update cryptocurrencies set name = %s where id = %s;', (name, id))

        cur.execute('select * from cryptocurrencies')
        rows = cur.fetchall()
        for row in rows:
            print(row)
```

Run the script.

```
(questdb-venv) $ python update_data.py
```

## Configuration of QuestDB

The location of the configuration file of the QuestDB database is dependent on the root directory of QuestDB. By default, it's located in `$HOME/.questdb`.

Edit the configuration file.

```
$ vim ~/.questdb/conf/server.conf
```

The configuration file has hundreds of lines. Some interesting settings are the ports for the protocols.

```
#http.net.bind.to=0.0.0.0:9000
#pg.net.bind.to=0.0.0.0:8812
#line.udp.bind.to=0.0.0.0:9009
# http.min.net.bind.to=0.0.0.0:9003
```

You can uncomment them and change the port value. If you want to change the username and password in the Postgres wire protocol, you can find them as well.

```
#pg.password=quest
#pg.user=admin
```

After changing the configuration file, don't forget to restart the server.

```
$ cd questdb-6.6.1-rt-linux-amd64
$ ./bin/questdb.sh stop
$ ./bin/questdb.sh start
```

You can also change the root directory of QuestDB by using the `-d` option.

```
$ ./bin/questdb.sh start -d /opt/questdb_root_dir
```

If you launch QuestDB with Docker, you have to bind the root directory using the `-v` option.

```
$ sudo docker run -p 9000:9000 -p 9009:9009 -p 8812:8812 -p 9003:9003 -v "$(pwd):/var/lib/questdb" questdb/questdb:6.6.1
```

In your current directory, there will be the `conf` directory. You can find `server.conf` inside that directory.

```
$ vim conf/server.conf
```

## More Resources

---

- [QuestDB GitHub Repository](#)
- [QuestDB Tutorial](#)



VULTR

