



# User guide for self-managed PostgreSQL

Updated 2024/01/19

This user guide describes how to tune a self-managed database instance using the DBtune software as a service. Self-managed instances include either an on-prem server or a self-managed cloud instance, such as EC2, GCE, Azure VM or cloud instances from any other cloud provider. DBtune works well on a variety of systems, including production enterprise systems.

Please [contact](#) the DBtune team if you would like to tune a fully-managed database instance instead, such as Amazon RDS, Azure Flexible Server or Google CloudSQL.

<b>System prerequisites</b>	<b>3</b>
Infosec whitelisting	3
System requirements	3
Packages to be installed	3
Time commitment to be expected from the user	3
<b>DBtune overview</b>	<b>4</b>
What is DBtune?	4
The challenge: Maximizing database efficiency beyond defaults	4
The problem: Neglected tuning leading to issues	4
The solution: DBtune’s adaptive AI approach	4
Why you should care: Practical benefits for your business	4
DBtune infrastructure	5
DBtune performance measurement	6
DBtune safety and automatic control policies	6
Total memory usage guardrail	6
Mitigation of temporary low performance during the tuning process	6
DBtune performance results	7
<b>How to get started</b>	<b>8</b>
Step 1: Login	8
Step 2: Add database instance to be tuned	8
Step 3: Download and run the DBtune Python client	10
Step 4: Monitor the tuning session	13
Step 5: Lay back, DBtune can take it from here	15
<b>How to deploy DBtune</b>	<b>16</b>
When to initiate a tuning session using DBtune	16
Scheduling a timed tuning session	16
Handling multiple workloads	17
Ensuring ongoing performance	17
Integrating DBtune with query tuning	17
Increase DBtune iteration time to capture	18
<b>Upgrading DBtune after trial</b>	<b>19</b>
<b>Additional material</b>	<b>20</b>
Interrupting the tuning session	20
Configuration management panel	20
Tuning the database instance once again	21
Modify memory usage guardrail threshold	22
<b>Security and privacy</b>	<b>22</b>

# System prerequisites

## Infosec whitelisting

Please ask your infosec department to whitelist these URLs for external access:

- AWS endpoints that is used in the DBtune client:  
<https://bwqh2n66kg.execute-api.eu-north-1.amazonaws.com/prod>
- AWS s3 bucket link used for downloading the client:  
<https://dbtune-eu-client-package-prod.s3.amazonaws.com>

## System requirements

- Superuser access to PostgreSQL
- PostgreSQL versions 10, 11, 12, 13, 14, 15, 16
- OS: Any Linux system
- Browser: Google Chrome, Firefox, Safari
- Server hosting the database: on-prem server or self-managed cloud instance, e.g., EC2, GCE, Azure VM

## Packages to be installed

- python3.8 # or later
- python3.8-pip # or later
- postgresql12 # or later
- postgresql-contrib # same version as postgres
- psutil==5.9.5
- psycopg2
- requests==2.29.0
- PyYAML==5.3.1
- screen or tmux

## Time commitment to be expected from the user

- First time users: about 30 minutes to start a tuning session.  
Second time users: 3 minutes.
- The user is not required to monitor the tuning session. However the DBtune web dashboard shows the tuning progress in real time.
  - The final optimal configuration is provided by DBtune after about 3.5 hours of tuning — It will automatically be installed on the server without user intervention.
  - The user can interrupt the tuning session and revert to the original configuration at any point. They can also cease the tuning session and deploy a new configuration to production when they believe they have reached a satisfactory or new optimal configuration before the full tuning session has ended.

# DBtune overview

## What is DBtune?

### The challenge: Maximizing database efficiency beyond defaults

Default database configurations often fall short of delivering optimal performance. The intricacies of data interaction, application behavior, and underlying hardware necessitate tailored configuration adjustments. These settings dictate how data flows through the system, impacting the performance of critical components such as RAM, CPU, and disk. This tuning process is crucial, aligning databases with real-world usage while making the most of available hardware. Parameters like `shared_buffers`, `work_mem`, and `random_page_cost` in PostgreSQL exemplify the need for this fine-tuning.

### The problem: Neglected tuning leading to issues

Neglecting database tuning can trigger a range of issues. These include inflated infrastructure costs, sluggish response times, increased downtime, decreased productivity, compromised data accuracy, dissatisfied users, missed opportunities, and potential financial setbacks. In today's competitive landscape, these consequences can significantly hinder an organization's growth and reputation.

### The solution: DBtune's adaptive AI approach

DBtune emerges as a solution to database configuration challenges. Unlike traditional manual tuning, vendor-specific tools, or DIY approaches, DBtune leverages advanced machine learning to dynamically adapt and optimize database configurations. The size, complexity, and workload of the database no longer impede effective tuning. The result? Noticeable improvements in system performance, increased transactions per second (TPS), shorter query runtimes, reduced technology expenses, and simplified management.

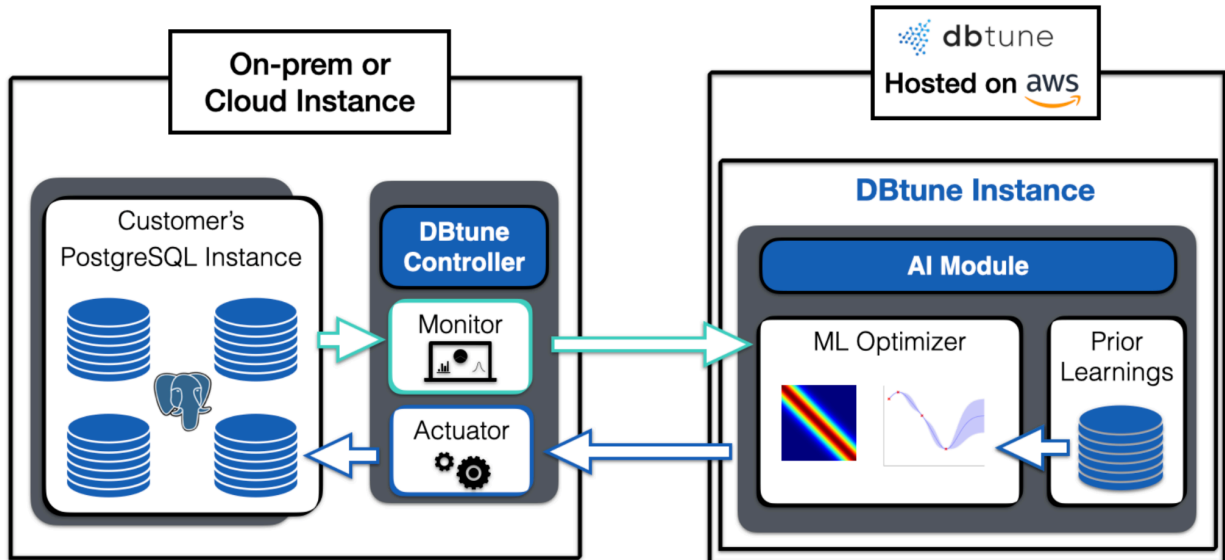
### Why you should care: Practical benefits for your business

DBtune isn't just another tool; it's an advantage for your business. By fine-tuning database configurations with precision, you can unlock previously untapped performance potential. This, in turn, can lead to cost savings by optimizing infrastructure spending, whether your databases are on-premise or in the cloud. Moreover, the automation of tuning tasks frees up your database administrators to focus on more strategic activities, such as refining query performance.

In a data-centric business landscape, DBtune empowers your database to operate at its best, improving performance, trimming costs, and allowing experts to steer your data initiatives forward.

## DBtune infrastructure

We present the high-level infrastructure view for the deployment of DBtune on a customer self-managed PostgreSQL instance. The figure below shows an overview of DBtune's architecture, which is an optimizer as a service (OaaS).



The DBtune OaaS maintains a repository of data collected from previous tuning sessions; This is represented by the prior learnings in the figure. This data is used to build models of how PostgreSQL responds to different knob configurations. For a new application, DBtune builds machine learning models as represented by the ML optimizer box in the figure, to guide experimentation and recommend optimal PostgreSQL configuration settings. Each recommendation and the subsequent observation of the behavior of the database instance provides DBtune with more information in a feedback loop that allows it to refine its models and improve their accuracy.

The system consists of two parts. The first is the client-side controller (on the left in the above diagram) that interacts with the target PostgreSQL instance to be tuned. It collects runtime information from the PostgreSQL instance using *psutil* and *pg\_stat\_statements*, installs new configurations in the *conf.d* directory, and collects performance measurements.

The second part is DBtune's tuning manager on the right side, called the AI module in the DBtune instance. It receives the information collected from the controller and stores it in its repository with data from previous tuning sessions. This repository does not contain any confidential information about the customer's database instance; it only contains knob configurations and performance data. The DBtune manager continuously analyzes new data and refines the internal ML models. These models automatically identify the recommended PostgreSQL configuration without human intervention.

The DBtune controller on the left is installed by the user, to connect to the customer's PostgreSQL instance. The DBtune controller is open-source under Apache 2.0 license. The OaaS on the right functions as a recommendation system for the database instance on the left. It finds the best configuration that optimizes throughput or latency depending on the user set optimization target. It learns to optimize by observing the database instance, so it generalizes for a new workload, for a new PostgreSQL version, and for a new hardware/cloud instance.

A full DBtune tuning session consists of approximately 30 iterations on average, i.e DBtune tries 30 different database configurations before providing a final instance configuration which delivers either peak TPS or minimum average query run time. This final configuration is then automatically installed on the server.

## DBtune performance measurement

Each DBtune iteration lasts about 7 minutes. Since DBtune will perform a total of 30 iterations, the total tuning session time will be about 3.5 hours. The total amount of time depends on if you set the system to allow or not allow the database to restart during tuning — More about the restart option below.

## DBtune safety and automatic control policies

DBtune is safe to use in production environments. Guardrails exist by design to protect against negative performance implications compared to the original baseline figures. DBtune's customers running in production have never experienced undesired downtime or undesirable degradation in performance due to the DBtune technology.

### Total memory usage guardrail

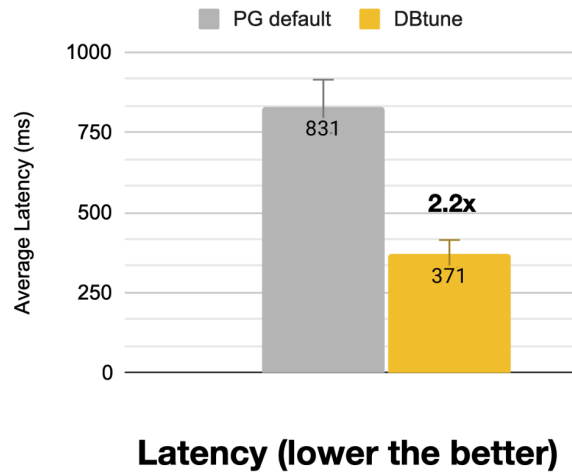
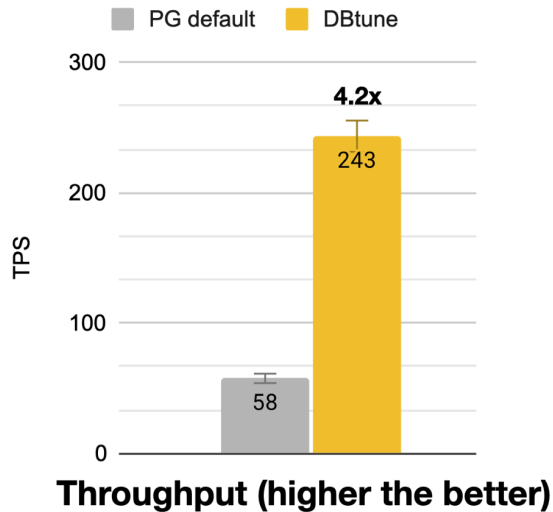
DBtune implements a guardrail to monitor the total RAM usage during the tuning session. As a default behavior DBtune will automatically react and move away from PostgreSQL configurations that make the database instance use more than 90% available memory.

### Mitigation of temporary low performance during the tuning process

DBtune implements a guardrail to monitor the performance of each database configuration that is used on the customer instance during the tuning session. If the performance of a configuration is below 40% of the best found configuration so far, then DBtune automatically abandons that configuration by moving to the next iteration and recommending a new alternative configuration.

## DBtune performance results

We report an example of the improvement in throughput (TPS) and latency (in milliseconds) that can be expected by the DBtune user on their PostgreSQL instance. The experiment is performed using the standard Wikipedia OLTPBench benchmark.

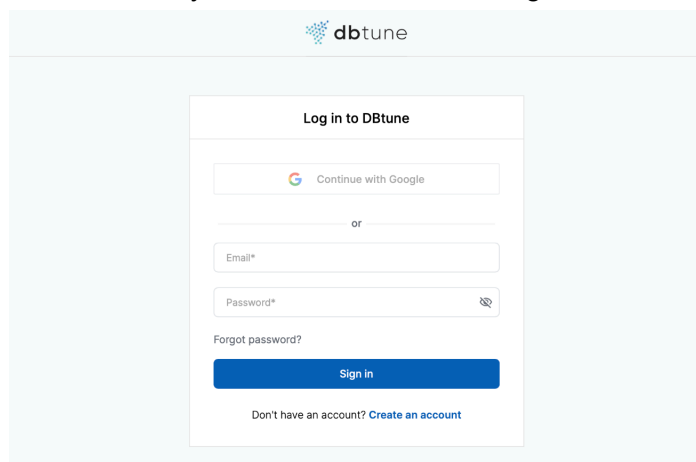


The configuration provided by PostgreSQL is named "default" in the plot. This is the configuration that is delivered in the PostgreSQL released package. We observe that DBtune is able to speed up the performance by 4.2x from 58 to 243 transactions per second. A similar outcome is observable for the latency metric, where DBtune is able to increase the performance of the baseline by 2.2x by reducing the average latency of the system from 831 ms to 371 ms.

# How to get started

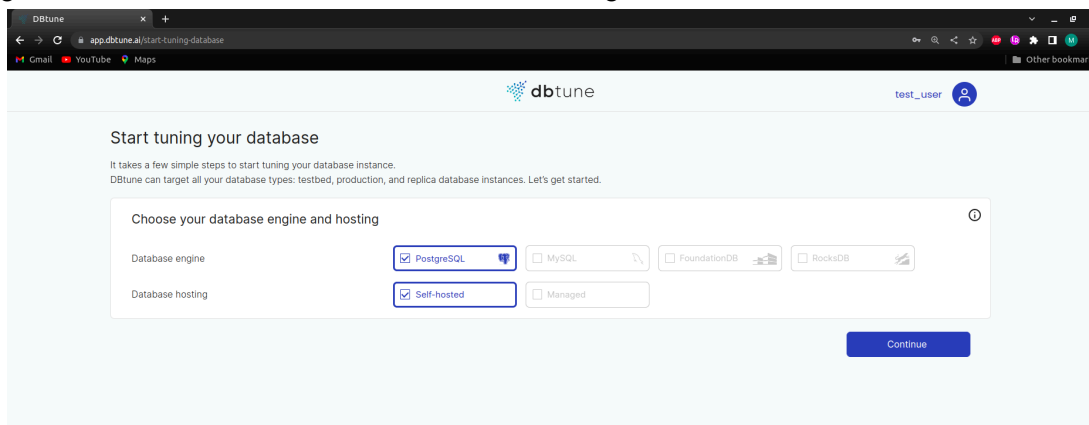
## Step 1: Login

- Visit [app.dbtune.com](http://app.dbtune.com).
- Create an account, confirm your email address and log in.



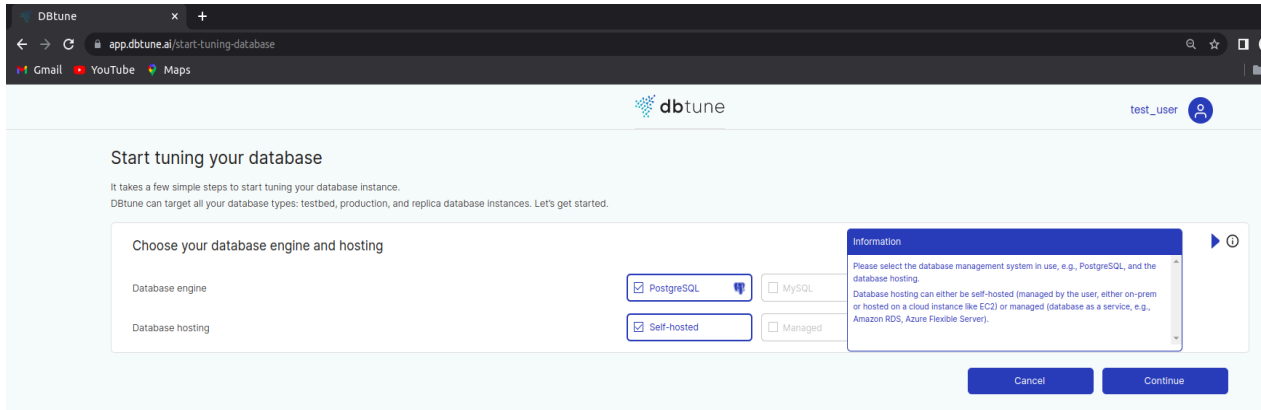
## Step 2: Add database instance to be tuned

After login, first-time users will be taken to the following screen.



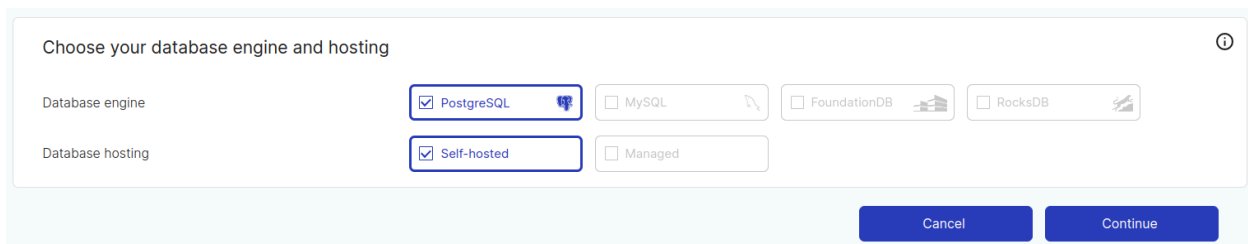
The infocards can help you through the process:





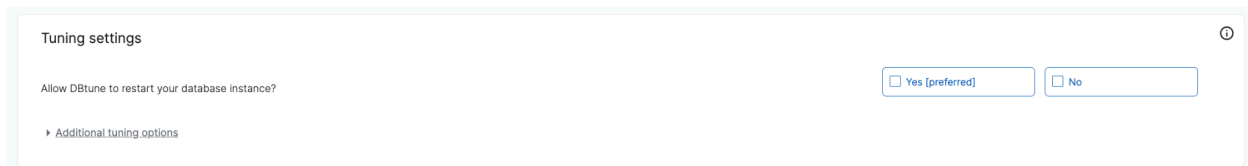
**Select the database management system and the database hosting:**

Database hosting can either be self-hosted (managed by the user, either on-prem or hosted on a cloud instance like EC2) or managed (database as a service, e.g., Amazon RDS, Azure Flexible Server).



**Choose if the database can be restarted:**

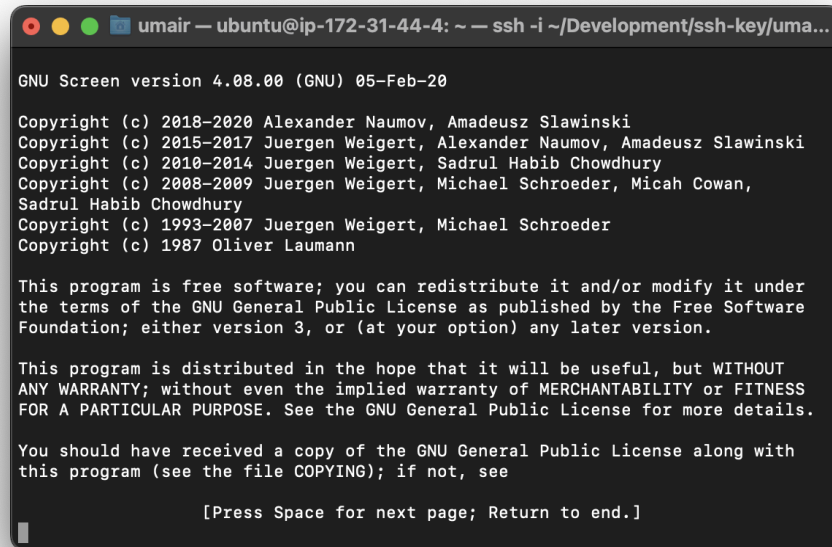
There are a number of parameters that can not be changed unless there is a restart of the database itself. The tuning session benefits from allowing the DB instance to restart as a greater number of parameters can be included. These cannot be changed and tuned if the option of no restart is selected. In the case restart is enabled, the database instance will be unavailable for a short period of time when each new configuration is deployed at each restart about every 5 minutes during the tuning session. The instance will be unavailable for a few seconds up to a few minutes depending on the user application.



**Step 3: Download and run the DBtune Python client**

Prepare a shell on your server machine:

1. Since DBtune will be running for a few hours, it is recommended to use a terminal multiplexer such as “screen” so as not to accidentally disconnect the tuning session if the CLI is closed. Type the command “screen” on the terminal of your client machine, you will get the following output:



```
umair — ubuntu@ip-172-31-44-4: ~ — ssh -i ~/Development/ssh-key/uma...
GNU Screen version 4.08.00 (GNU) 05-Feb-20

Copyright (c) 2018-2020 Alexander Naumov, Amadeusz Slawinski
Copyright (c) 2015-2017 Juergen Weigert, Alexander Naumov, Amadeusz Slawinski
Copyright (c) 2010-2014 Juergen Weigert, Sadrul Habib Chowdhury
Copyright (c) 2008-2009 Juergen Weigert, Michael Schroeder, Micah Cowan,
Sadrul Habib Chowdhury
Copyright (c) 1993-2007 Juergen Weigert, Michael Schroeder
Copyright (c) 1987 Oliver Laumann

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 3, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program (see the file COPYING); if not, see

[Press Space for next page; Return to end.]
```

2. Login as root user or use the command “*sudo su*”.
3. Download the DBtune client using the *wget*, *curl* or *download* command on the web interface. The command will download the DBtune client as a tar file and uncompress it. The client you download is specific to your user account and to the database instance you are about to connect it to, and it reflects the choices on tuning target and the restart option. To change any of these you need to download a new client.

Download the DBtune client, provide the database instance connection details and start the client ⓘ

Select a command on the right to download the client on the machine hosting the database instance. Follow the instructions in the `dbtune_config.yml` file in the `dbtune_client` folder to connect the client with the instance.

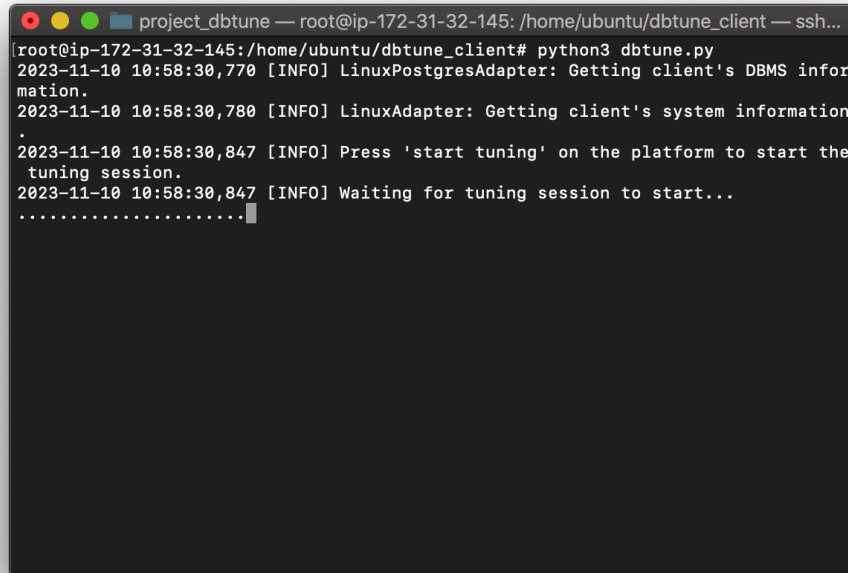
Log in as the root user or run the command “`sudo su`” before starting the client. Start the client with the command `./dbtune`.

[wget](#) [curl](#) [Download](#)



- port
  - The port postgres is running on.
- dbname
  - The name of the database instance.
- user
  - The database username
- password
  - The password for the database user
- restart\_command
  - If you have installed postgres from source and the command “sudo systemctl restart postgresql” does not restart postgres, provide a command that does in this field. A working restart command is necessary if you allow the client to restart postgres, or if the postgres extension pg\_stat\_statements is not currently enabled and you want to tune for query runtime. Enabling pg\_stat\_statements requires one restart, which the client will ask if you want to allow, and if you do then it will do it for you even if you choose not to allow restarts in the previous step from the DBtune web interface.

5. Start the client by running ‘python3 dbtune.py’.



```

project_dbtune — root@ip-172-31-32-145: /home/ubuntu/dbtune_client — ssh...
[root@ip-172-31-32-145: /home/ubuntu/dbtune_client# python3 dbtune.py
2023-11-10 10:58:30,770 [INFO] LinuxPostgresAdapter: Getting client's DBMS information.
2023-11-10 10:58:30,780 [INFO] LinuxAdapter: Getting client's system information
.
2023-11-10 10:58:30,847 [INFO] Press 'start tuning' on the platform to start the tuning session.
2023-11-10 10:58:30,847 [INFO] Waiting for tuning session to start...
.....

```

6. Enter a database instance name and press the **Start tuning** button on the web interface

Name your database instance ⓘ

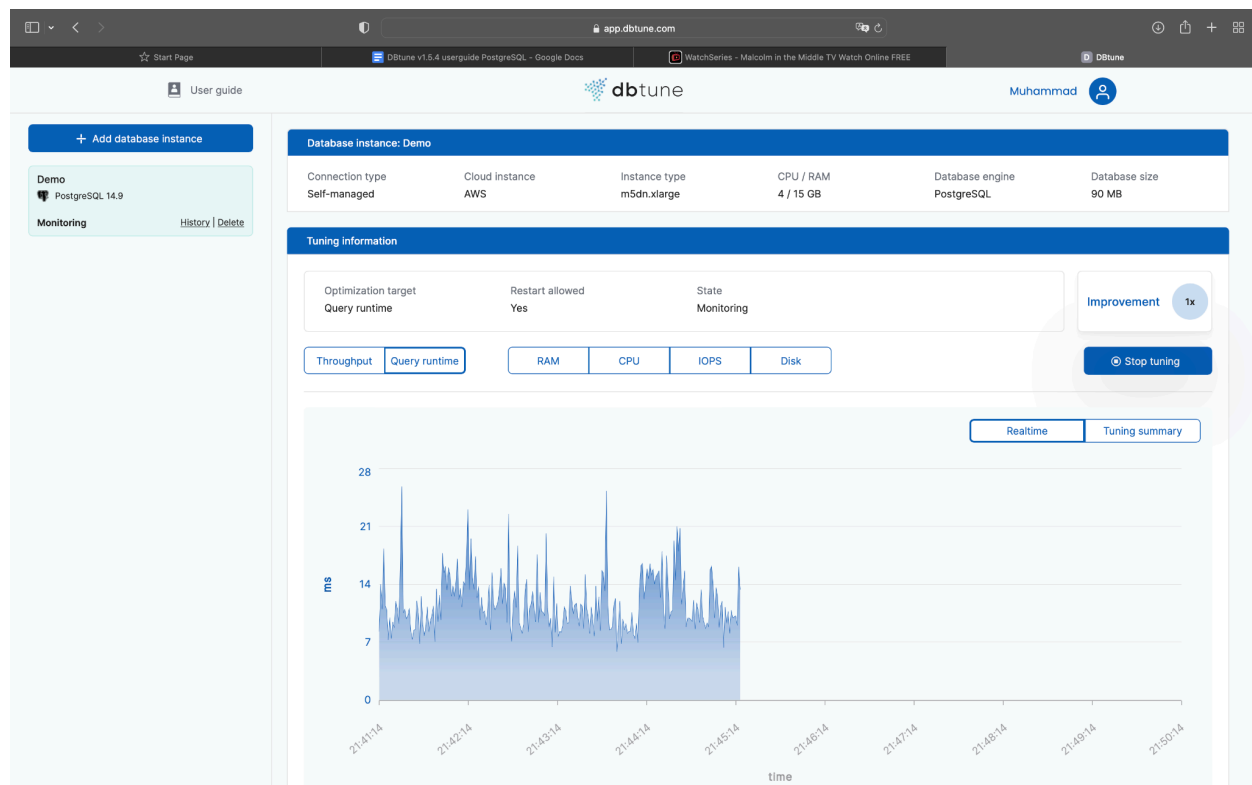
Choose a name for your database instance. You can always change this name later.

Start tuning →

7. Once the tuning has started you can detach the screen:
  - Press `Ctrl + a` (release) and then `d` to detach: the tuning will keep on running in the background;
  - You can reattach the screen using the following command: `screen -r`;
  - For more details on how to use `screen` see this [link](#).

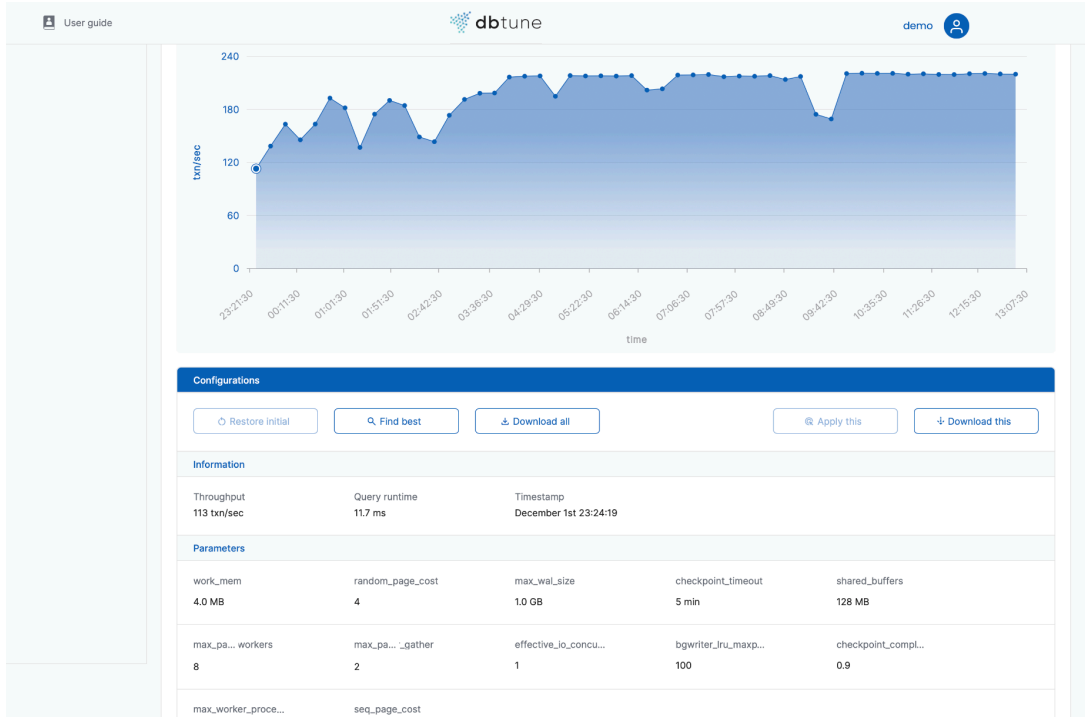
## Step 4: Monitor the tuning session

After starting the tuning session the user will be redirected to the dashboard



You can monitor the throughput, the average query runtime, disk I/O, RAM usage, CPU, and IOPS usage in real time as the tuning progresses.

The currently active configuration is shown in the configurations panel. Under *Tuning summary* you can see all the configurations that were explored by DBtune. Click on a point in the chart to visualize the configuration in the bottom panel.



## **Step 5:** Sit back, DBtune can take it from here

The tuning session is running, no additional effort is required from the user.

DBtune is learning how to optimize your database with respect to your specific workload and instance resources. The tuning session will last about 3.5 hours. After that DBtune will automatically install on the instance being tuned the PostgreSQL configuration that delivers the optimal results for your chosen target objective (throughput or query runtime).

If you require the optimization to run faster, please contact the DBtune team at [info@dbtune.com](mailto:info@dbtune.com). In this user guide we have been conservative in defining the tuning session for the user. We can adapt to your needs and make the tuning faster.

# How to deploy DBtune

## When to initiate a tuning session using DBtune

DBtune is an online system designed to dynamically observe and optimize workloads in real-time through 30 optimization iterations. Achieving the best results involves triggering a tuning session during the period of maximum load and peak usage. This aligns with times when end-users perceive system sluggishness and are actively seeking improvements.

Example: Imagine a bank operational from 9 am to 5 pm. The daily transaction load slows down the database. To enhance performance, DBtune should be triggered between 9 am and 5 pm, the heavy workload window. Conversely, tuning during nighttime hours lacks significance due to the absence of performance bottlenecks.

Tip: If the peak workload ends at 5 pm, start DBtune no later than 1:30 pm. This ensures the entire DBtune tuning session operates within the period of peak load.

Based on extensive experience with numerous users, there is typically only one time window that the user wants to optimize. For instance, in the example above, the focus is on the 9 am to 5 pm window. Performance during the 5 pm to 9 am time frame isn't problematic for the bank, rendering optimizations unnecessary.

DBtune's goal is to tune the time window that matters. Users should initiate DBtune when the intensive database usage window is active, completing the tuning session entirely within that period, i.e. the full tuning session needs to start and complete between 9 am and 5 pm in the example above.

## Handling multiple workloads

While the standard use case involves optimizing a single workload, scenarios can arise where distinct workloads occur at different times of the day or the week.

Example: Consider the bank scenario: apart from weekday transactions, a complex reporting workload arises on Saturdays. This workload is very different from the one that the system is experiencing during the weekdays because it is composed of complex queries that manipulate large swaths of data in the database. This distinct nature necessitates separate tuning sessions.

In such cases, users trigger DBtune twice: once during the weekday 9-to-5 window and again during the Saturday window. The outcomes, say, *config1* and *config2*, are optimal configurations for their respective time frames. The two configurations have to be then scheduled to appear on the server with access to the `conf.d` directory. A script written by the user schedules these

configurations to apply to the system via the `conf.d` directory, thus accommodating diverse workloads. The script simply copies the configuration files provided by DBtune in the `conf.d` directory at the right time so that the optimal configuration for each workload is used.

## Ensuring ongoing performance

DBtune delivers optimal performance after a tuning session. However, as databases are dynamic, evolve and grow, regular usage of DBtune is crucial to maintain peak performance.

Example: Consider a scenario where a DBtune session optimizes a database instance on June 1st. After a week, performance dips noticeably due to application changes rendering the current configuration ineffective.

Tip: Monitor your database's performance and, upon detecting degradation, launch a new DBtune session. If issues relate to database configuration, DBtune will swiftly restore optimal performance.

The frequency of necessary re-tuning varies based on your application. The following events typically signal the need for a new DBtune session:

1. Changes in one or more queries due to application updates.
2. Upgrading PostgreSQL to a newer version.
3. Migrating the database to different hardware or cloud instances.
4. Substantial growth in the database's size.

## Integrating DBtune with query tuning


After conducting query tuning, initiate a DBtune session to further optimize performance. As query tuning is often iterative, running DBtune after each round of tuning enhances query plans. If results fall short of expectations, restart the iterative process: first, perform query tuning, followed by parameter tuning with DBtune. The compounded performance of query tuning and parameter tuning can be much higher than the single improvements.



# Upgrading DBtune after trial

This trial is valid for one month — During the trial the user can tune one database instance as many times as needed.

The banner below is present when the user is using a free trial. To complete a purchase click on "Upgrade now".

 Welcome to your free trial.  
With this trial you can tune one database for a limited amount of time.

[Upgrade now](#)

If you prefer to try out DBtune first on a synthetic workload follow the steps in this [guide](#).

Enter the payment details to subscribe — The payment is powered by Stripe.

## Enter payment details

Complete your purchase to automatically tune your database for optimal performance!  
Mandatory fields are marked with an asterisk "\*\*

<b>Billing details</b>	<b>Order details</b>
Email address*	Subtotal \$100/month
<input type="text"/>	VAT 0%
Contact number	
<input type="text"/>	
Country*	<b>Total \$100/month</b>
Select a country	Powered by <b>stripe</b>
Street address*	
<input type="text"/>	
State*	
Zip / postal code	
<input type="text"/>	
<input type="text"/>	
<b>Check out</b>	

When you press the "Check out" button, Stripe asks the card details to finalize the purchase.

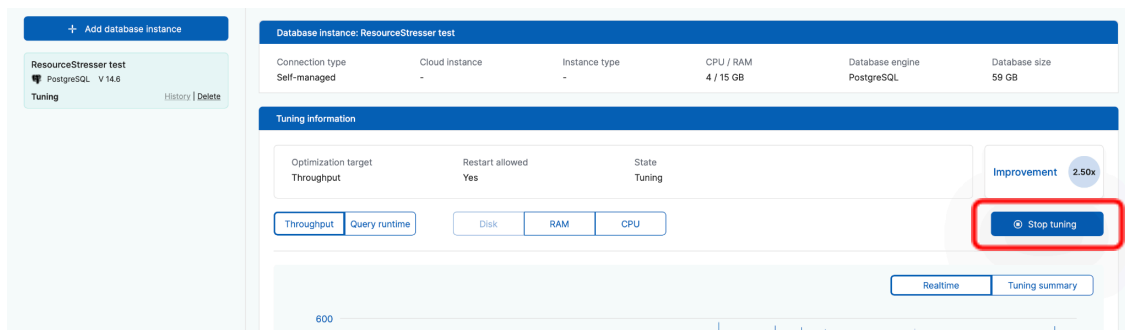
The subscription includes a monthly fee of \$100 per database instance. Currently, only one database instance subscription at a time can be purchased. If you'd like to acquire multiple database instances please contact our sales team at [info@dbtune.com](mailto:info@dbtune.com). Discounts on purchases of 5 database instances and above are available with our sales representatives.

## Additional material

### Interrupting the tuning session

To stop the tuning sessions the user has two equivalent options:

1. **From the CLI:** Press Ctrl-c to abort the tuning session. You will be prompted to choose between reinstating the initial configuration and installing the best configuration found so far.
2. **From the dashboard screen:** Press the **"Stop tuning"** button in the web interface. This will install the best configuration found so far on the system.



### Configuration management panel

The configuration management panel can be found below the performance monitoring plot on the DBtune dashboard screen. This allows you to perform specific actions regarding the database configurations explored by DBtune.



1. **"Restore initial" button:**  
Stops the tuning session and reinstates the default system configuration. Note that once the tuning session is aborted or completed, this action can not be performed.
2. **"Find best" button:**

It allows the user to trace the best performing configuration found by DBtune. After pushing the button the user will identify with a circle the best performing configuration in the performance plot as shown below.



3. **"Download all" button:**

Allows the user to download the history of configurations applied on the user's database along with the achieved database throughput and/or query runtime.

4. **"Apply this" button:**

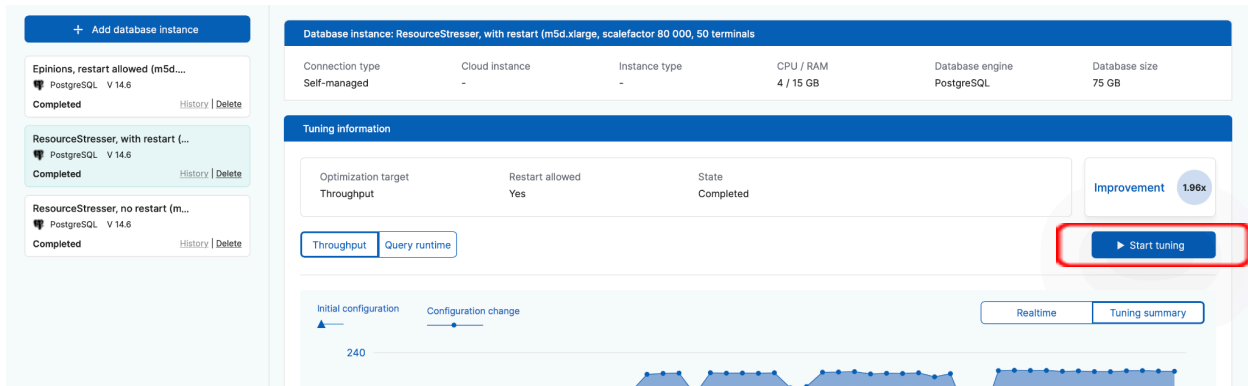
Allows the user to install a specific configuration from the tuning history on the user database instance. Note that you need to select a configuration from the tuning summary plot before using this feature. Once the tuning session is aborted or completed, this action can not be performed.

5. **"Download this" button:**

Allows you to download the PostgreSQL configuration file named **"99\_dbtune.conf"** for the selected configuration.

## How to run subsequent tuning sessions

Once the initial tuning session is completed, the **"Stop tuning"** button is replaced by a **"Start tuning"** button. Before starting a new tuning session, you need to run the DBtune client. When you press the "Start tuning" button, a popup window will give you the command to run on your CLI. Run the command on your cloud instance as sudo user and use a terminal multiplexer (see step 3 above), to start a new tuning session from the beginning. Previous tuning sessions will appear under the **"History"** tab on the left.



## Modify memory usage guardrail threshold

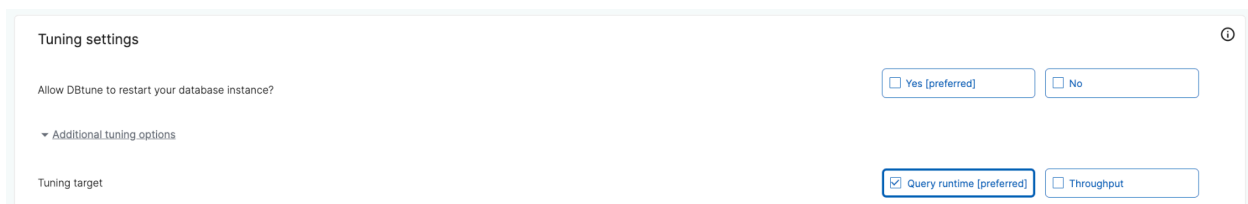
As a default behavior DBTune will automatically react and move away from PostgreSQL configurations that make the database instance use more than 90% available memory. This threshold can be customized in the DBTune YAML configuration files to accommodate specific user requirements. This can be achieved by adding the following line to the `dbtune_config.yml` file located in the downloaded `dbtune_client`:

```
tuning_session:
  Maximum_memory_usage_allowed: 90
```

## Choose alternative optimization target

The default target optimization is average query runtime which is a form of latency (measured in milliseconds). Tuning for average query runtime will also have the effect of improving the number of transactions per second (TPS) which is a form of throughput.

It is possible to put more emphasis on explicitly tune for throughput in the *Additional tuning options* in the *Start tuning database* page.



The target options are either:

1. **Throughput**: measured in transactions per second (TPS), or

2. **Average query runtime:** this represents latency, which is measured in milliseconds (ms).

The recommended target is average query runtime, which we have seen works in most cases for both metrics. In case you want to be more intentional about the objective that you chose, we recommend choosing:

1. Throughput if your machine/workload is hitting a bottleneck (either close to 100% CPU or 100% I/O), or in other words, an increased workload wouldn't lead to increased throughput. In this scenario DBtune can help you to increase the overall throughput of your workload and achieve improved performance at peak.
2. Average query runtime if your workload is composed of complex queries and you wish to reduce the average runtime of one query. The average query runtime can improve even when the machine is not bottlenecked quite as intensely. Bigger database instances will drive bigger query runtime improvement.

Note that typically, improving one optimization target will also benefit the other target. However, this choice allows the optimizer to focus on an optimization objective.

# Security and privacy

Port used: 443

Infosec whitelisting:

- AWS endpoints that is used in the DBtune client:  
<https://bwqh2n66kg.execute-api.eu-north-1.amazonaws.com/prod>
- AWS s3 bucket link used for downloading the client:  
<https://dbtune-eu-client-package-prod.s3.amazonaws.com>

Comprehensive list of data that is being fetched from the machine/PG instance:

## 1. Client system information

- Hardware
  - NUMOFCPU
  - TOTALMEMORY
  - AVAILABLEMEMORY
  - CLOUDPROVIDER
  - INSTANCETYPE
  - DISKSIZE
  - HDTYPE
- Software
  - DBVERSION
  - OSTYPE
  - MAXCONNECTIONS
  - DATABASESIZE

## 2. Performance metrics retrieved

- Performance
  - Throughput
  - Query runtime

## 3. Configuration parameters tuned

- Work\_mem
- Max\_wal\_size
- Seq\_page\_cost
- Shared\_buffers
- Random\_page\_cost
- Checkpoint\_timeout
- Max\_parallel\_workers
- Max\_worker\_processes
- Bgwriter\_lru\_maxpages
- Effective\_io\_concurrency

- Checkpoint\_completion\_target
- Max\_parallel\_workers\_per\_gather

#### 4. Client monitoring stats (posted every second)

- Cpu\_stats
  - Cpu\_util
- Memory\_stats
  - free
  - slab
  - used
  - total
  - active
  - cached
  - shared
  - buffers
  - percent
  - inactive
  - Available
- Io\_stats
  - Busy\_time
  - Read\_time
  - Read\_bytes
  - Read\_count
  - Write\_time
  - Write\_bytes
  - Write\_count
  - Read\_merged\_count
  - write\_merged\_count
- Db\_stats
  - Throughput
  - query\_runtime