

Extensions

release date. 2026. 09. 11.

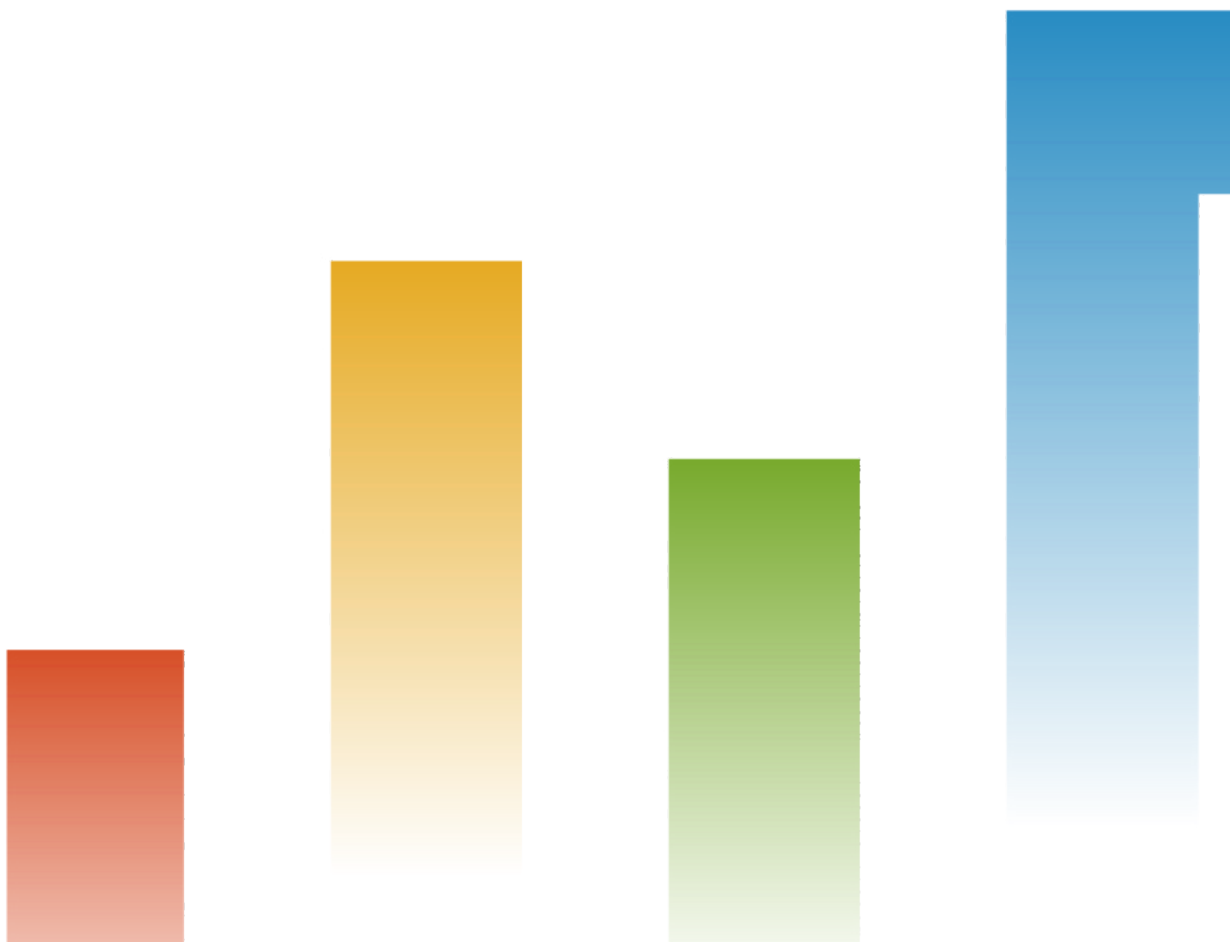


Table Of Contents

• Extensions	3
◦ Focus	4
■ Support Environment	5
■ Installation	6
■ Configuration.....	8
■ Usage examples	15
◦ Telegraf	18
■ Plug-in installation and operation	
■ Installing the plug-in.....	21
■ Configuration	28
■ Agent installation and operation	
■ Support Environment.....	30
■ Agent Installation	31
■ Troubleshooting	36
■ Configuration	37
■ Management.....	40
■ Usage examples	43
◦ OpenTelemetry	47
■ Install Agent.....	49
■ Agent Configuration	59
■ Release Notes	0
■ Download Agent.....	68

Extensions

It guides the tools that can expand for collecting data. In addition to the following materials, please send your feedback including material requests to docs@whatap.io.

 Focus

4 items

 Telegraf

3 items

 OpenTelemetry

4 items

What Is Focus?

Focus is a program that uploads random data not provided by the WhaTap monitoring in time series.

The operation team can upload the collected values by operating the self-developed check scripts or database queries periodically or once. Uploaded values can be viewed in the Analysis/Metrics Chart menu of the project. You can also add it to your Flex board.

To use Focus, select one of Server, Application (Java, PHP, NodeJS, Python), Database, and Kubernetes projects among WhaTap projects, and then specify the project access key string and collection server IP of the project as execution options.

Support Environment

It can be applied to Windows, Linux, FreeBSD, OSX, and ARMv5,6,7 and ARM64 Linux OS running on X86.

Product	Environment	Support Environment
Focus	Windows	Windows 2008R2 or later
	Linux	Debian 7.0 or later
		Ubuntu 12.04 or later
		CentOS, Red Hat 6.0 or later
		Amazon Linux 1.0 or later
		SUSE 12.1 or later
	FreeBSD	FreeBSD 10 or later
	OSX	Lion 10.7 or later
	Raspberry Pi OS	8 or later

ⓘ If the environment is not included in the support list, contact us at support@whatap.io.

Installation

WhaTap Focus has a single executable file, and it requires no separate installation or registration process. You can download the executable file and run it at any place.

Downloading the executable file

WhaTap Focus has a single executable file that can be downloaded and used immediately for each version.

64 bit Debian / Ubuntu / Red Hat / CentOS / Amazon Linux / SUSE

```
wget https://repo.whatap.io/focus/linux_amd64/focus
chmod +x ./focus
```

64 bit OSX

```
wget https://repo.whatap.io/focus/darwin_amd64/focus
chmod +x ./focus
```

64 bit FreeBSD

```
wget https://repo.whatap.io/focus/freebsd_amd64/focus
chmod +x ./focus
```

64 bit OSX

```
wget https://repo.whatap.io/focus/darwin_amd64/focus
chmod +x ./focus
```

64 bit Windows

```
Invoke-WebRequest -Uri https://repo.whatap.io/focus/windows_amd64/focus.exe -OutFile focus.exe
```

ArmV5 Linux

```
wget https://repo.whatap.io/focus/linux_arm_5/focus
chmod +x ./focus
```

ArmV6 Linux

```
wget https://repo.whatap.io/focus/linux_arm_6/focus
chmod +x ./focus
```

ArmV7 Linux

```
wget https://repo.whatap.io/focus/linux_arm_7/focus
chmod +x ./focus
```

Arm64 Linux

```
wget https://repo.whatap.io/focus/linux_amd64/focus
chmod +x ./focus
```

Focus file configuration

- Linux/FreeBSD/OSX/Raspberry Pi OS
- **focus**: Program for data collection and transmission.
- Windows
- **focus.exe**: Program for data collection and transmission.

Configuration

You can upload any collected data from the WhaTap project. Focus can be used even if the default agent has not been installed for the project.

Checking the project access key and collection server IP

In [Management](#) > [Agent Installation](#) for the selected project, check the **project code**, **collection server IP**, and **project access key**.

- [Linux Shell](#) Windows Powershell

SH

```
export WHATAP_LICENSE=xxxx-xxxxxx-xxxxxx
export WHATAP_HOST=xxxx.xxxx.xxxx
export WHATAP_PCODE=xxx
```

SH

```
$WHATAP_LICENSE=xxxx-xxxxxx-xxxxxx
$WHATAP_HOST=xxxx.xxxx.xxxx
$WHATAP_PCODE=xxx
```

Adding specific tags for filtering

You can set specific strings as tags for filtering. Through the settings, you can filter the data collected from the WhaTap monitoring service screen by the name and value for the tag.

- [Linux Shell](#) Windows Powershell

SH

```
1 # -tag.[User tag key name][Blank][User tag value]
2
3 ./focus -license $WHATAP_LICENSE \
```

```

4 -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
5 -tag.CustomTagName CustomTagValue \
6 -tag.MyServer1 server_01 \
7 ...

```

Powersehll

```

1 # -tag.[User tag key name][Blank][User tag value]
2
3 .\focus.exe -license $WHATAP_LICENSE `
4 -pcode $WHATAP_PCODE -server.host $WHATAP_HOST `
5 -tag.CustomTagName CustomTagValue `
6 -tag.MyServer1 server_01 `
7 ...

```

Sending alerts

Custom alerts can be sent immediately. After sending an alert, the Focus is ended.

- [Linux Shell](#) Windows Powershell

SH

```

#Select a severity of the alert.
level={info|warn|fatal}
#Alert title
title=Desired alert title
#Alert body
message=Desired alert body
./focus -license $WHATAP_LICENSE \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -alert -level $level -title $title -message $message

```

Powersehll

```

#Select a severity of the alert.
$level="{info|warn|fatal}"
#Alert title
$title="Desired alert title"

```

```
#Alert body
$message=Desired alert body
.\focus.exe -license $WHATAP_LICENSE `
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST `
  -alert -level $level -title $title -message $message
```

Collecting the SQL query result

It collects the execution results of custom SQL queries in time series indefinitely.

- [Linux Shell](#) Windows Powershell

SH

```
#Metric category
category=my_category
#Database driver
driver={mysql|postgres}
#Database connection info
dburl="ID:password@tcp(IP:Port)/Database"
#Sql Query
sqlquery="select some, columns from sometable"
.\focus -license $WHATAP_LICENSE \
  -category $category \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -rdb $driver -rdb.connect $dburl \
  -rdb.sql $sqlquery
```

Powershell

```
#Metric category
$CATEGORY="my_category"
#Database driver
$driver="{mysql|postgres}"
#Database connection info
$dburl="ID:password@tcp(IP:port)/Database"
#Sql Query
$sqlquery="select some, columns from sometable"
```

```
.\focus.exe -license $WHATAP_LICENSE `
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST `
  -category $CATEGORY `
  -rdb $driver -rdb.connect $dburl `
  -rdb.sql $sqlquery
```

Collecting the keyword matching logs of the log file

If a keyword is included when a log occurs in the specified text log file, the log line is collected.

- [Linux Shell](#) | Windows Powershell

SH

```
#Metric category
category=my_category
#Log file
LOG_FILE=Log file path
#Delimiter to use between keywords when entering multiple log keywords
LOG_KEYWORDS_SEPERATOR=,
#Enter log keywords by using the delimiter (,).
LOG_KEYWORDS=keyword1,keyword2
./focus -license $WHATAP_LICENSE \
  -category $category \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -tail $LOG_FILE \
  -tail.keys $LOG_KEYWORDS \
  -tail.seperator $LOG_KEYWORDS_SEPERATOR
```

Powersehll

```
#Metric category
$CATEGORY="my_category"
#Log file
LOG_FILE=Log file path
#Delimiter to use between keywords when entering multiple log keywords
LOG_KEYWORDS_SEPERATOR=,
#Enter log keywords by using the delimiter
LOG_KEYWORDS=keyword1,keyword2
```

```
.\focus.exe -license $WHATAP_LICENSE `
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST `
  -category $CATEGORY `
  -tail $LOG_FILE `
  -tail.keys $LOG_KEYWORDS `
  -tail.seperator $LOG_KEYWORDS_SEPERATOR
```

WhaTap log analyzing service

When a log occurs in the specified text log file, it is uploaded in real time to the WhaTap log analysis service.

- **Linux Shell** Windows Powershell

SH

```
#Category
category=my_category
#Can include the log file wildcard (*) and date pattern (http://strftime.org)
LOG_FILE=log file path
.\focus -license $WHATAP_LICENSE \
  -category $category \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -logsink $LOG_FILE
```

Powersehll

```
#Category
$CATEGORY="my_category"
#Can include the log file wildcard (*) and date pattern (http://strftime.org)
$LOG_FILE=log file path
.\focus.exe -license $WHATAP_LICENSE `
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST `
  -category $CATEGORY `
  -logsink $LOG_FILE
```

Collecting the OS resource usage

It collects the resource usage of the server on which Focus is running.

Linux Shell

```
#Metric category
category=my_category
#Whether or not to enable the disk monitoring
diskenabled=true|false
#Disk mount to monitor
diskmount=/mypartition
#Whether or not to enable the NIC monitoring
nicenabled=true|false
#NIC to monitor
nic=eth0

./focus -license $WHATAP_LICENSE \
  -category $category \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -sys \
  -sys.disk.enabled $diskenabled \
  -sys.disk $diskmount \
  -sys.net.enabled $nicenabled \
  -sys.net $nic
```

Collecting results of running the temporary programs and scripts

It runs programs and scripts and continuously collects results to stdout. WhaTap Focus collects the keys and values of the dictionary when any value is input to stdin in the json dictionary format. Because the pipe is to be used as an input method, it is required to disable buffering.

- For continuous collection instead of one-time collection, you can apply the following option.

**-onetime**

- To specify the collection time, you can enter it with the following option. `-now {unix epoch time(second)}`

Linux Shell

```
#Metric category
CATEGORY="my_category"

Program or script | \
Reprocessing in the json format | \
./focus -license $WHATAP_LICENSE \
  -category $category \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST
```

The following example executes the top command to continuously collect the CPU and memory usage of a specific process.

Linux Shell

```
#Metric category
CATEGORY="my_category"

export PID=PID of the process to collect
top -b -p $PID | awk '/'$PID'/{ printf "{\\"pid\\": %s, \\"cpuPercent\\": %s, \\"memoryPercent\\": %s, \\"cmd\\": \\"%s\\"}\n", $1, $9, $10, $12}; system("")' | \
./focus -license $WHATAP_LICENSE \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -category $CATEGORY
```

Usage Examples

Use Focus to guide you to frequently used commands and other use cases. This will be helpful to development and operation by allowing you to check the data in time series.

Checking the project access key and collection server IP

In [Management](#) > [Agent Installation](#) for the selected project, check the **project code**, **collection server IP**, and **project access key**.

- [Linux Shell](#) Windows Powershell

SH

```
export WHATAP_LICENSE=xxxx-xxxxxx-xxxxxx
export WHATAP_HOST=xxxx.xxx.xxx
export WHATAP_PCODE=xxx
```

SH

```
$WHATAP_LICENSE=xxxx-xxxxxx-xxxxxx
$WHATAP_HOST=xxxx.xxx.xxx
$WHATAP_PCODE=xxx
```

Adding specific tags for filtering

Referring to the following example, you can filter the data collected from the WhaTap monitoring service screen by the name and value for the tag.

SH

```
1 # -tag.[User tag key name][Blank][User tag value]
2
3 ./focus -license $WHATAP_LICENSE \
4   -pcode $WHATAP_PCODE -server:host $WHATAP_HOST \
```

```

5 -tag.CustomTagName CustomTagValue \
6 -tag.MyServer1 server_01 \
7 ...

```

Collecting the CPU and memory for each Top command PID

The following example executes the top command to continuously collect the CPU and memory usage of a specific process.

Linux Shell

```

#Metric category
CATEGORY="my_category"

export PID=PID of the process to collect
top -b -p $PID | awk '/$PID/{ printf "{\"pid\": %s, \"cpuPercent\": %s, \"memoryPercent\": %s, \"cmd\": \"%s\"}\n", $1, $9, $10, $12}; system(\"")' | \
./focus -license $WHATAP_LICENSE \
-pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
-category $CATEGORY

```

Using NETSTAT

The following example executes the netstat command to collect the number of TCP connections by status.

Linux Shell

```

#Metric category
CATEGORY="my_category"

netstat -nat | tail -n+3 | awk '{print $6}' | sort | uniq -c | awk 'BEGIN { printf "{" } {if (NR!=1) {printf ", "}}{printf "\"%s\":\"%s\",$2,$1} END { print "}" }' | \
./focus -license $WHATAP_LICENSE \
-pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
-category $CATEGORY -onetime

```

Using VMSTAT

The following example executes the vmstat command to collect itemized metrics.

Linux Shell

```
#Metric category
CATEGORY="my_category"

vmstat -n 5 | awk 'NR>2 {printf "{ \"r\":%s,\"b\":%s,\"swpd\":%s,\"free\":%s,\"buff\":%s,\"cache\":%s,\"si\":%s,\"so\":%s,\"bi\":%s,\"bo\":%s,\"in\":%s,\"cs\":%s,\"us\":%s,\"sy\":%s,\"id\":%s,\"wa\":%s,\"st\":%s }\n", $1,$2, $3,$4, $5,$6, $7,$8, $9,$10, $11,$12, $13,$14, $15,$16,$17}\n' | \
./focus -license $WHATAP_LICENSE \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -category $CATEGORY
```

Using DU

The following example executes the du command to collect the capacity of a directory.

Linux Shell

```
#Metric category
CATEGORY="my_category"
TARGET=Directory to collect capacity in

du -sb $TARGET --max-depth=0 | awk 'BEGIN { printf "{" } {if (NR!=1) {printf ", "}}{printf "\"%s\":%s", $2,$1} END { print "}" }' | \
./focus -license $WHATAP_LICENSE \
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
  -category $CATEGORY -onetime
```

What Is Telegraf?

Telegraf is a plug-in driven server agent for collecting and sending metrics and events from databases, systems, and IoT sensors. Telegraf has been written in Go, compiles into a single binary file without any external dependency, and requires the minimum memory space.

- **** Collection and transmission of all types of data****

- **Database**

- It collects and sends metrics by connecting data sources such as MongoDB, MySQL, and Redis.

- **System**

- It collects metrics from modern cloud platforms, containers, and orchestrator stacks.

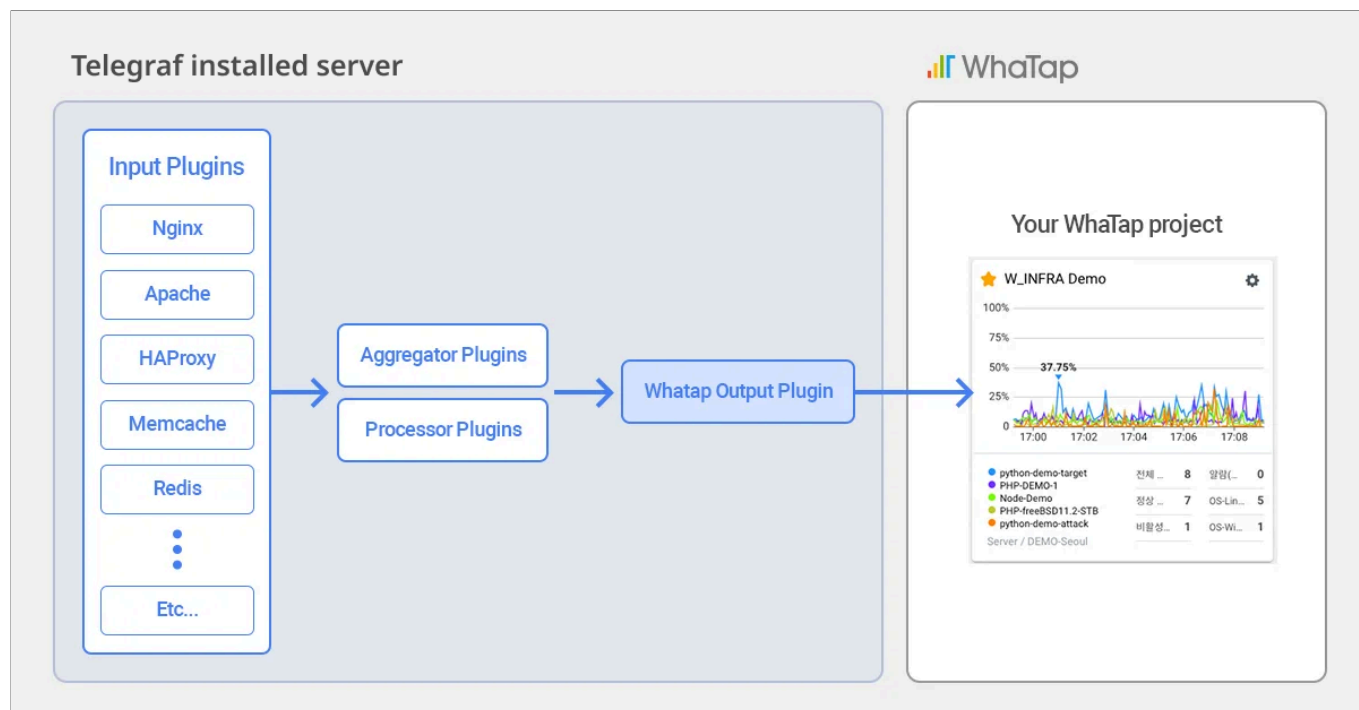
- **IoT Sensor**

- It collects important stateful data (pressure levels, temperature levels, etc.) from IoT sensors and devices.

- **Telegraf Plug-in**

Telegraf is a plug-in driven agent that collects, processes, aggregates, and creates metrics. It supports four categories of plug-ins: input, output, aggregator, and processor.

You can forward the metrics collected by Telegraf to the WhaTap collection server by using the Telegraf's WhaTap output plug-in and installing the WhaTap Telegraf agent.



⚠ Telegraf plug-ins and agents must have been installed and configured directly on the service.

The guide is provided on how to install plug-ins or agents and the use cases are provides as follows.

📄 Plug-in installation and operation

2 items

📄 Agent installation and operation

5 items

Usage examples

It guides you to use cases.

Installing The Plug-in

whatap output plugin

You can use the Telegraf's WhaTap output plug-in (TCP) to forward metrics collected in Telegraf to the WhaTap collection server. The collected data can be checked in the WhaTap's project.

! Telegraf plug-ins must have been installed and configured directly on the service.

Support Environment

The following operating systems are available for installation via download. For other operating systems, proceed with [Compiled installation](#).

- macOS
- Red Hat & CentOS
- Ubuntu & Debian
- FreeBSD
- Windows

Installing via download

This installation file has been created in advance by adding the WhaTap plug-in based on the Telegraf release version. If Telegraf has been already installed, you can enable the WhaTap plug-in by replacing only the executable.

Telegraf release version

- [1.21.1](#) 1.16.0 1.15.4 1.14.5 1.13.2 1.12

- **Red Hat & CentOS**

- <https://repo.whatap.io/telegraf/telegraf-release-1.21.1/linux/amd64/telegraf> (executable file)
- <https://repo.whatap.io/telegraf/telegraf-release-1.21.1/linux/amd64/telegraf-1.21.1-1.amd64.rpm>
- https://repo.whatap.io/telegraf/telegraf-release-1.21.1/linux/amd64/telegraf-1.21.1_linux_amd64.tar.gz

- **Ubuntu & Debian**

- <https://repo.whatap.io/telegraf/telegraf-release-1.21.1/linux/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.21.1/linux/amd64/telegraf_1.21.1-1_amd64.deb
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0/linux/amd64/telegraf-1.21.1_linux_amd64.tar.gz

- **macOS**

- <https://repo.whatap.io/telegraf/telegraf-release-1.16.0/darwin/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0/darwin/amd64/telegraf-1.16.0_darwin_amd64.tar.gz

- **Red Hat & CentOS**

- <https://repo.whatap.io/telegraf/telegraf-release-1.16.0/linux/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0/linux/amd64/telegraf-1.16.0-1.x86_64.rpm
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0/linux/amd64/telegraf-1.16.0_linux_amd64.tar.gz

- **Ubuntu & Debian**

- <https://repo.whatap.io/telegraf/telegraf-release-1.16.0/linux/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0/linux/amd64/telegraf_1.16.0-1_amd64.deb
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0/linux/amd64/telegraf-1.16.0_linux_amd64.tar.gz

- **FreeBSD**

- <https://repo.whatap.io/telegraf/telegraf-release-1.16.0/freebsd/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0/freebsd/amd64/telegraf-1.16.0_freebsd_amd64.tar.gz

- **Windows**

- <https://repo.whatap.io/telegraf/telegraf-release-1.16.0/windows/amd64/telegraf.exe> (executable file)
- <https://repo.whatap.io/telegraf/telegraf-release-1.16.0/windows/amd64/>

[telegraf-1.16.0_windows_amd64.zip](#)

- **macOS**

- <https://repo.whatap.io/telegraf/telegraf-release-1.15.4/darwin/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4/darwin/amd64/telegraf-1.15.4_darwin_amd64.tar.gz

- **Red Hat & CentOS**

- <https://repo.whatap.io/telegraf/telegraf-release-1.15.4/linux/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4/linux/amd64/telegraf-1.15.4-1.x86_64.rpm
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4/linux/amd64/telegraf-1.15.4_linux_amd64.tar.gz

- **Ubuntu & Debian**

- <https://repo.whatap.io/telegraf/telegraf-release-1.15.4/linux/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4/linux/amd64/telegraf_1.15.4-1_amd64.deb
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4/linux/amd64/telegraf-1.15.4_linux_amd64.tar.gz

- **FreeBSD**

- <https://repo.whatap.io/telegraf/telegraf-release-1.15.4/freebsd/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4/freebsd/amd64/telegraf-1.15.4_freebsd_amd64.tar.gz

- **Windows**

- <https://repo.whatap.io/telegraf/telegraf-release-1.15.4/windows/amd64/telegraf.exe> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4/windows/amd64/telegraf-1.15.4_windows_amd64.zip

- **macOS**

- <https://repo.whatap.io/telegraf/telegraf-release-1.14.5/darwin/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.14.5/darwin/amd64/telegraf-1.14.5_darwin_amd64.tar.gz

- **Red Hat & CentOS**

- <https://repo.whatap.io/telegraf/telegraf-release-1.14.5/linux/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.14.5/linux/amd64/telegraf-1.14.5-1.x86_64.rpm
- https://repo.whatap.io/telegraf/telegraf-release-1.14.5/linux/amd64/telegraf-1.14.5_linux_amd64.tar.gz
- **Ubuntu & Debian**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.14.5/linux/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.14.5/linux/amd64/telegraf_1.14.5-1_amd64.deb
 - https://repo.whatap.io/telegraf/telegraf-release-1.14.5/linux/amd64/telegraf-1.14.5_linux_amd64.tar.gz
- **FreeBSD**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.14.5/freebsd/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.14.5/freebsd/amd64/telegraf-1.14.5_freebsd_amd64.tar.gz
- **Windows**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.14.5/windows/amd64/telegraf.exe> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.14.5/windows/amd64/telegraf-1.14.5_windows_amd64.zip
- **macOS**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.13.2/darwin/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.13.2/darwin/amd64/telegraf-1.13.0~842282d_darwin_amd64.tar.gz
- **Red Hat & CentOS**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.13.2/linux/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.13.2/linux/amd64/telegraf-1.13.0~842282d-0.x86_64.rpm
 - https://repo.whatap.io/telegraf/telegraf-release-1.13.2/linux/amd64/telegraf-1.13.0~842282d_linux_amd64.tar.gz
- **Ubuntu & Debian**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.13.2/linux/amd64/telegraf> (executable file)

- https://repo.whatap.io/telegraf/telegraf-release-1.13.2/linux/amd64/telegraf_1.13.0~842282d-0_amd64.deb
- https://repo.whatap.io/telegraf/telegraf-release-1.13.2/linux/amd64/telegraf-1.13.0~842282d_linux_amd64.tar.gz
- **FreeBSD**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.13.2/freebsd/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.13.2/freebsd/amd64/telegraf-1.13.0~842282d_freebsd_amd64.tar.gz
- **Windows**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.13.2/windows/amd64/telegraf.exe> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.13.2/windows/amd64/telegraf-1.13.0~842282d_windows_amd64.zip
- **macOS**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.12/darwin/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.12/darwin/amd64/telegraf-1.12.0~842282d_darwin_amd64.tar.gz
- **Red Hat & CentOS**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.12/linux/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.12/linux/amd64/telegraf-1.12.0~842282d-0.x86_64.rpm
 - https://repo.whatap.io/telegraf/telegraf-release-1.12/linux/amd64/telegraf-1.12.0~842282d_linux_amd64.tar.gz
- **Ubuntu & Debian**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.12/linux/amd64/telegraf> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.12/linux/amd64/telegraf_1.12.0~842282d-0_amd64.deb
 - https://repo.whatap.io/telegraf/telegraf-release-1.12/linux/amd64/telegraf-1.12.0~842282d_linux_amd64.tar.gz
- **FreeBSD**

- <https://repo.whatap.io/telegraf/telegraf-release-1.12/freebsd/amd64/telegraf> (executable file)
- https://repo.whatap.io/telegraf/telegraf-release-1.12/freebsd/amd64/telegraf-1.12.0~842282d_freebsd_amd64.tar.gz
- **Windows**
 - <https://repo.whatap.io/telegraf/telegraf-release-1.12/windows/amd64/telegraf.exe> (executable file)
 - https://repo.whatap.io/telegraf/telegraf-release-1.12/windows/amd64/telegraf-1.12.0~842282d_windows_amd64.zip

Compiled installation

Preparations

- Go SDK configuration
- go dep installation
- Source File

Unzip in `${GOPATH} /src/github.com/influxdata/telegraf/`.

ⓘ They are source files that include the WhaTap plug-in in the Telegraf release version.

- https://repo.whatap.io/telegraf/telegraf-release-1.12_whatap.tar.gz
- https://repo.whatap.io/telegraf/telegraf-release-1.13.2_whatap.tar.gz
- https://repo.whatap.io/telegraf/telegraf-release-1.14.5_whatap.tar.gz
- https://repo.whatap.io/telegraf/telegraf-release-1.15.4_whatap.tar.gz
- https://repo.whatap.io/telegraf/telegraf-release-1.16.0_whatap.tar.gz

Installation procedure

ⓘ For the detailed builds, see the [Telegraf github](#) page.

Under `${GOPATH}/src/github.com/influxdata/telegraf` proceed with the build with the Makefile.

1. Proceed with the dependency installation. Dependency installation uses the go dep. Accordingly, the go dep must have been installed first.

```
# make deps
```

2. Build the executable file. Under the `${GOPATH}/bin` directory, the executable file (telegraf) is generated.

```
# make go-install
```

3. If the installation package is required, you can create it with the `make package` command. Proceed with cross-compilation to generate the package files for each operating system.

```
# make package
```

Configuration

In the config file ([telegraf.conf](#)) of Telegraf, enable the setting of [outputs.whatap](#).

! Telegraf plug-ins must be configured directly on the service.

A config file containing the WhaTap plugin can be created as follows:

```
# telegraf --sample-config > telegraf.conf
```

WhaTap project information

In [Home screen](#), select a project and then set the project data. You can check it in the agent installation guide of the project to connect.

- **Project Access Key:** Project access key to identify the project
- **Project Code:** Numeric code to identify the project
- **Collection Server Info (IP, Port):** IP and port of the WhaTap collection server that delivers the collected data

config setting (toml)

/etc/telegraf/telegraf.conf

```
# # Configuration for WhaTap
[[outputs.whatap]]
#   ## You can create a project on the WhaTap site(https://www.whatap.io)
#   ## to get license, project code and server IP information.
#
#   ## WhaTap license. Required
license = "x2tggnopk2t9-xxxxxxxxx-aaaaaaaaa"
#
#   ## WhaTap project code. Required
pcode = 118
#
```

```
# ## WhaTap server IP. Required
# # Put multiple IPs with / as delimiters. e.g. "1.1.1.1/2.2.2.2"

servers = ["tcp://1.2.3.4:6600", "tcp://5.6.7.8:6600"]

# ## Connection timeout.
# # timeout = "60s"
```

- `[[outputs.whatap]]` Delete the comment (#).
- Delete the comment (#) of `license` . Enter the access key of the project. It is a character type.
- Delete the comment (#) of `pcode` . Enter the project code. It is numeric.
- Delete the comment (#) of `servers` . Set the collection server data (IP, port). The protocol supports only TCP. Register as "tcp://ip:port" in text format. Register in array format. Separate with comma (,) and then register values in array format.

 To apply the settings, restart Telegraf.

Support Environment

Before installing the WhaTap Telegraf agent, check the support environment.

- **OS**
 - Linux Red Hat 6.x x64 or later
 - Ubuntu 12.x x64 or later
 - FreeBSD 10.x x64 or later
- **Telegraf**
 - Telegraf 1.3 or later

① The version that supports the socket_writer output plugin is allowed.

Agent Installation

The following guides you to the basic installation method for using the WhaTap Telegraf agent.

Metrics collected by Telegraf can be forwarded to the WhaTap collection server. The collected data can be checked in the WhaTap's project. This page explains how to install the whatap-telegraf agent to relay the data from Telegraf to the collection server.

Enable the `socket_writer` output plugin of the installed Telegraf to pass the data to the whatap-telegraf (Local TCP) agent.

To use the WhaTap service, after [sign in](#), you must create a project and install the agent to the target server. For more information about sign in, see [the following](#).

ⓘ Telegraf agents must be installed and configured directly on the service.

Creating a project

Create a project before installing the agent.

1. Log in to the WhaTap Monitoring Service.
2. In the left side menu, click **All Projects** > **+ Project**.
3. On the **Select product** screen, select the product you want to install.
4. Enter or select the following items:
 - **Project name**: Enter the project name.
 - **Data server region**: Select the data server region. A region is a group of data centers that provide cloud services. Your data will be stored in the selected region.
 - **Time zone**: Set the time zone to be used for alerts and report generation.
 - **Notification language setting**: Set the language for alert notifications. (Supports Korean and English)
 - **Project groups**: Group multiple projects for easier management. Select a group if applicable.

- **Project description:** Add additional details or a description for your project.

5. After completing all settings, click the **Creating a project** button.

ⓘ If you add a project while an organization is selected, you must set the **Groups of organization** field.
For more information about groups, see the related document.

Checking the project access key

Project access key is the unique ID for activating the WhaTap services.

In the installation guide section, select **Getting the access key**. After automatic reception of **project access key**, proceed to the next step.

✓ After a project has been created, the **Agent installation** page appears automatically. If the **Agent installation** does not appear, select **All projects** on the left and then select a newly created project.

Installation order and file configuration

1. Install the WhaTap **repository**.
2. Install the whatap-telegraf **Linux package** (yum, apt-get).
3. Set the **project access key** and **** WhaTap server information**** settings.
4. Set the Telegraf's `socket_writer` output plugin to link the data.

ⓘ Configuring the Telegraf agent file

ⓘ

- **whatap_telegraf**
By using the service running file, this program sends the data from Telegraf to the collection server.



- **/etc/init.d/whatap-teleggraf** (FreeBSD /usr/local/etc/whatap_telegraf)
Service script file.
- **whatap.conf**
It is the configuration file. This is the file where the address of the collection server and the server's project access key are entered.
- **whatap-teleggraf-#.log**
Agent log file. ([/usr/whatap/teleggraf/logs](#))

Installing the package

- [Red Hat/CentOS](#) [Debian/Ubuntu](#) [Amazon Linux](#) [FreeBSD](#)

1. Install the WhaTap **repository**.

```
$ sudo rpm -Uvh https://repo.whatap.io/centos/5/noarch/whatap-repo-1.0-1.noarch.rpm
```

2. Install the **package** with the following command.

```
$ sudo yum install whatap-teleggraf
```

3. Run the **configuration script** to start the service.

```
echo "license=[granted project access key]" | sudo tee /usr/whatap/teleggraf/whatap.conf
echo "whatap.server.host=[WhaTap server address]" | sudo tee -a /usr/whatap/teleggraf/whatap.conf
sudo service whatap-teleggraf restart
```

1. Install the WhaTap **repository**.

```
$ wget https://repo.whatap.io/debian/release.gpg -O - | sudo apt-key add -
$ wget https://repo.whatap.io/debian/whatap-repo_1.0_all.deb
$ sudo dpkg -i whatap-repo_1.0_all.deb
```

```
$ sudo apt-get update
```

2. Install the **package** with the following command.

```
$ sudo apt-get install whatap-telegraf
```

3. Run the **configuration script** to start the service.

```
echo "license=[granted project access key]" | sudo tee /usr/whatap/telegraf/whatap.conf
echo "whatap.server.host=[WhaTap server address]" | sudo tee -a /usr/whatap/telegraf/whatap.conf
sudo service whatap-telegraf restart
```

1. Install the WhaTap **repository**.

```
$ sudo rpm --import https://repo.whatap.io/centos/release.gpg
$ echo "[whatap]" | sudo tee /etc/yum.repos.d/whatap.repo > /dev/null
$ echo "name=whatap packages for enterprise linux" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null
$ echo "baseurl=https://repo.whatap.io/centos/latest/\$basearch" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null
$ echo "enabled=1" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null
$ echo "gpgcheck=0" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null
```

2. Install the **package** with the following command.

```
$ sudo yum install whatap-telegraf
```

3. Run the **configuration script** to start the service.

```
echo "license=[granted project access key]" | sudo tee /usr/whatap/telegraf/whatap.conf
echo "whatap.server.host=[WhaTap server address]" | sudo tee -a /usr/whatap/telegraf/whatap.conf
sudo service whatap-telegraf restart
```

1. Install the WhaTap **repository**.

```
$ wget https://s3.ap-northeast-2.amazonaws.com/repo.whatap.io/freebsd/10/whatap-telegraf-0.0.4.txz
```

```
$ pkg install whatap-telegraf-0.0.4.txz
```

2. Run the **configuration script** to start the service.

```
echo "license=[granted project access key]" | tee /usr/whatap/telegraf/whatap.conf
echo "whatap.server.host=[WhaTap server address]" | tee -a /usr/whatap/telegraf/whatap.conf
sudo service whatap_telegraf restart
```

Telegraf integration

By setting `socket_writer` output plugin in [Telegraf.conf](#), the data collected via whatap-telegraf is sent.

TOML

```
# # Generic socket writer capable of handling multiple socket types.
[[outputs.socket_writer]]

# ## URL to connect to
address = "tcp://127.0.0.1:6600"

# # data_format = "influx"
data_format = "json"
```

- `[[outputs.socket_writer]]` Uncomment the line
- Enter the TCP connection setting in the address field.
- Set the data type in json format.

You have finished installation of the Telegraf agent. In [the following](#) check the installation troubleshooting.

Troubleshooting

Restart the whatap-telegraf service if it is not running or encounters an error.

⚠️ Telegraf plug-ins and agents must have been installed and configured directly on the service.

- **Red Hat/CentOS**

```
$ service whatap-telegraf restart
```

- **Debian/Ubuntu**

```
$ sudo service whatap-telegraf restart
```

- **FreeBSD**

```
$ service whatap_telegraf restart
```

Configuration

The Whatap agent writes the necessary settings for each agent in [whatap.conf](#). The agent locates and loads the configuration file through the environment variables. The options that can be set through the [whatap.conf](#) file are as follows:

! The Telegraf agent must be manually configured.

Agent name identification

WhaTap basically uses the data collected from the server for identifying the infrastructure server that is the target of monitoring data collection.

- ! • The agent names must be unique on a project basis.
- If the agent ID and name have been changed, previous data is not connected.

- **object_name** *String*

Default `{type} - {hostname} - {ip2} - {ip3} - {docker}`

It is an agent naming (ONAME) scheme to identify applications. An OID is created based on the ONAME.

Name	Description
<code>{type}</code>	Use the value set in app_name. The default value is "Telegraf."
<code>{ip#}</code>	When IPs are divided, the #th digit is used.
<code>{hostname}</code>	The server host name is used.
<code>{docker}</code>	Use the IP of the Docker container.

- **app_name** *String*

It is a component used in the agent naming (ONAME) scheme to identify applications. It corresponds to the `{type}` among the `object_name` options.

❗ It is applied after restarting the `whatap-telegraf` service.

Agent communication settings

- **license** `String`

To install the agent, set the project access key granted from the server. The project access key includes the agent's project and the encryption key for secured communication.

- **whatap.server.host** `ip_address`

Default `127.0.0.1,127.0.0.1`

Set the server to transfer the data collected by the agent. If there are multiple IPs due to the collection server redundancy, they can be specified by separating with comma (,). The collection server's proxy daemon must be serviced in the listening state for the specified IP.

- **whatap.server.port** `tcp_port`

Default `6600`

Set the collection server's port. Because only one port can be specified, the collection servers specified in `whatap_server_host` must use the same port.

- **tcp_so_timeout** `Millisecond`

Default `60000`

Set the socket timeout of the TCP session that communicates with the collection server.

- **tcp_connection_timeout** `Millisecond`

Default `5000`

Set the connection time out of the TCP session that communicates with the collection server.

- **net_send_max_bytes** `Byte`

Default `5242880`

Set the maximum size of the data to send to the collection server at one time.

- **net_send_buffer_size** Byte

Default `1024`

Maximum data in bytes to transfer data.

Telegraf

- **telegraf_tcp_port** tcp_port

Default `6600`

TCP port to connect with Telegraf socker_writer output plugin.

! It is applied after restarting the `whatap-telegraf` service.

- **telegraf_delta_fields** String

Default `1024`

Add an extra field after calculating the increment value of the field with cumulative values. Set the value as the name.fields format. Multiple items can be separated with comma (,).

e.g. The increment values for accepts and requests in nginx.accepts, nginx.requests nginx are delivered with the names of acceptsDelta and requestsDelta fields.

Log

- **log_keep_days** Day

Default `7`

Set the retention period of log files.

Management

Let's learn about additional options you can set through the [whatap.conf](#) file to manage the Telegraf agents.

Update

- [Red Hat/CentOS](#) [Debian/Ubuntu](#) [FreeBSD](#)

1. Delete cache data for updating package information.

```
$ yum clean all
```

2. Update the whatap-teleggraf package.

```
$ yum update whatap-teleggraf
```

3. To install a version, specify the version data during installation.

```
$ sudo yum install whatap-teleggraf-{version}
```

1. Delete cache data for updating package information.

```
$ sudo apt-get update
```

2. Update the whatap-teleggraf package.

```
$ sudo apt-get install --only-upgrade whatap-teleggraf
```

3. To install a version, specify the version data during installation.

```
$ sudo apt-get install whatap-teleggraf={version}
```

1. Install the package again.

SH

```
# wget https://s3.ap-northeast-2.amazonaws.com/repo.whatap.io/freebsd/10/whatap-telegraf-0.0.4.txz
# pkg install whatap-telegraf-0.0.4.txz
```

2. To install a version, `{version}` enter the version data before installation.

SH

```
# wget https://s3.ap-northeast-2.amazonaws.com/repo.whatap.io/freebsd/10/whatap-telegraf-{version}.txz
# pkg install whatap-telegraf-{version}.txz
```

Pause

- [Red Hat/CentOS](#) [Debian/Ubuntu](#) [FreeBSD](#)

SH

```
$ sudo service whatap-telegraf stop
```

SH

```
$ sudo service whatap-telegraf stop
```

SH

```
$ service whatap_telegraf stop
```

Close Pending

- Delete the package (yum, apt-get).
- [Red Hat/CentOS](#) [Debian/Ubuntu](#) [FreeBSD](#)

```
```bash title='SH'
```

```
$ sudo yum remove whatap-telegraf
...
```

```
```bash title='SH'  
$ sudo apt-get purge whatap-telegraf  
...
```

```
```bash title='SH'  
$ pkg delete whatap-telegraf
...
```

- If necessary, delete the log files and others under [/usr/whatap/telegraf](#).

# Usage Examples

For more information about Telegraf, see the [Telegraf documentation](#) for influxdata.

ⓘ Telegraf plug-ins and agents must have been installed and configured directly on the service.

## SNMP

SNMP performance data can be collected and charted in real time using the SNMP input plug-in. The collected data can be retrieved by `telegraf_snmp` and `telegraf_interface` categories.

Linux Shell

```
cat >/etc/telegraf/telegraf.d/snmp_device.conf <<EOL
[[inputs.snmp]]
 ## Agent addresses to retrieve values from.
 ## format: agents = ["<scheme://><hostname>:<port>"]
 ## scheme: optional, either udp, udp4, udp6, tcp, tcp4, tcp6.
 ## default is udp
 ## port: optional
 ## example: agents = ["udp://127.0.0.1:161"]
 ## agents = ["tcp://127.0.0.1:161"]
 ## agents = ["udp4://v4only-snmp-agent"]

 agents = ["udp://1.1.1.1:161"]

 ## Timeout for each request.
 # timeout = "5s"

 ## SNMP version; can be 1, 2, or 3.
 version = 2

 ## SNMP community string.
 community = "xxx@xxxxx"
```

```

Agent host tag
agent_host_tag = "agent_host"

Number of retries to attempt.
retries = 3

The GETBULK max-repetitions parameter.
max_repetitions = 10

SNMPv3 authentication and encryption options.
##
Security Name.
sec_name = "myuser"
Authentication protocol; one of "MD5", "SHA", "SHA224", "SHA256", "SHA384", "SHA512" or "".
auth_protocol = "MD5"
Authentication password.
auth_password = "pass"
Security Level; one of "noAuthNoPriv", "authNoPriv", or "authPriv".
sec_level = "authNoPriv"
Context Name.
context_name = ""
Privacy protocol used for encrypted messages; one of "DES", "AES", "AES192", "AES192C", "AES256", "AES256C", or "".
Protocols "AES192", "AES192C", "AES256", and "AES256C" require the underlying net-snmp tools
to be compiled with --enable-blumenthal-aes (http://www.net-snmp.org/docs/INSTALL.html)
priv_protocol = ""
Privacy password used for encrypted messages.
priv_password = ""

Add fields and tables defining the variables you wish to collect. This
example collects the system uptime and interface variables. Reference the
full plugin documentation for configuration details.
[[inputs.snmp.field]]
 oid = "RFC1213-MIB::sysUpTime.0"
 name = "uptime"

[[inputs.snmp.field]]
 oid = "RFC1213-MIB::sysName.0"
 name = "source"
 is_tag = true

[[inputs.snmp.table]]

```

```

oid = "IF-MIB::ifXTable"
name = "interface"
inherit_tags = ["source"]

[[inputs.snmp.table.field]]
oid = "IF-MIB::ifDescr"
name = "ifDescr"
is_tag = true

[[aggregators.derivative]]
period = "60s"
max_roll_over = 1

fieldpass = ["*Octets", "*Pkts"]
drop_original = false

[aggregators.derivative.tags]
aggr = "derivative"

[[processors.starlark]]
source = ""
def apply(metric):
 for (k, v) in metric.fields.items():
 if k.endswith('Octets_rate'):
 metric.fields[k] *= 8

 return metric

'''

EOL
service telegraf restart

```

## SNMP Trap

Using the SNMP Trap input plug-in, SNMP Trap events can be collected in real time and propagated via emails, text messages, and messengers. The collected data can be retrieved by the `telegraf_snmp_trap` category.

Linux Shell

```
cat >/etc/telegraf/telegraf.d/snmp_trap.conf <<EOL
[[inputs.snmp_trap]]
 service_address = "udp://:162"
EOL
service telegraf restart
```

# Introduction To OpenTelemetry

 OpenTelemetry monitoring is a Beta version so unexpected errors may occur. It is not recommended for using it for sensitive data or in the operation environment. If you have any feedback or issues, feel free to contact the support team at [support@whatap.io](mailto:support@whatap.io).

This document is intended for users who want to collect additional data through OpenTelemetry while using applications of the WhaTap monitoring service.

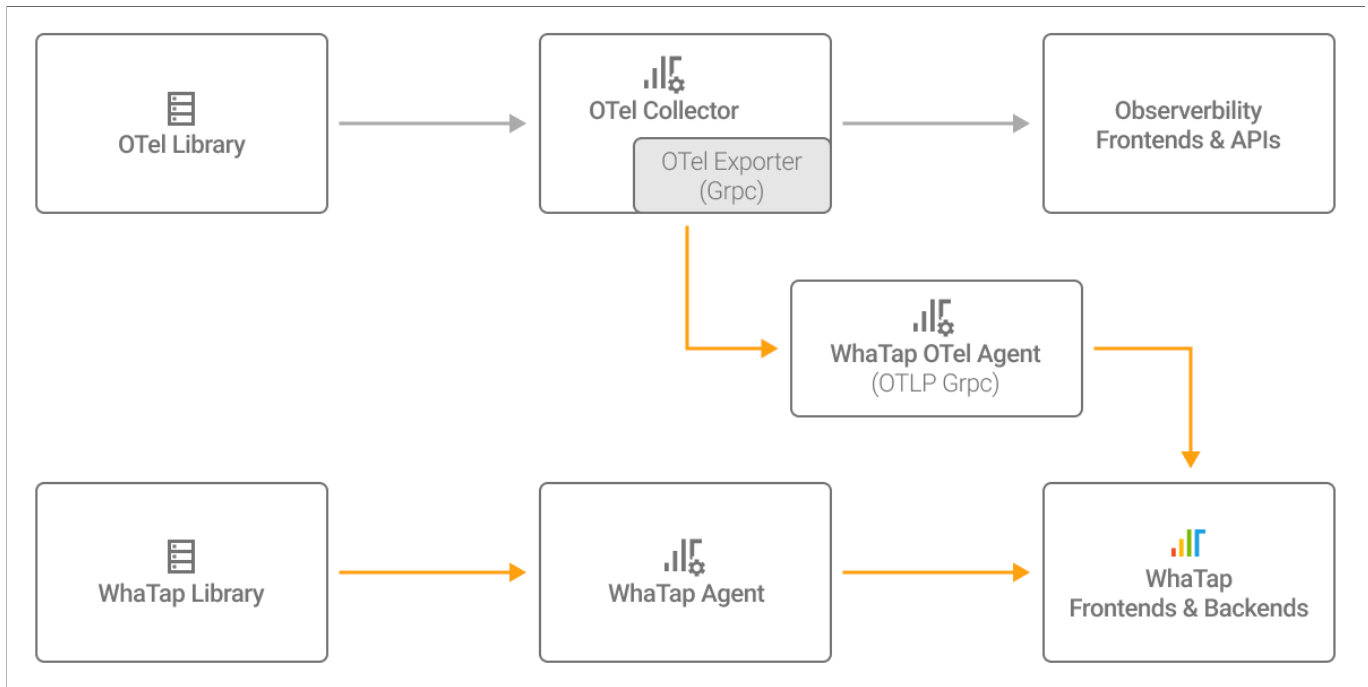
## Using the WhaTap monitoring through OpenTelemetry

[OpenTelemetry](#) is an open source observability framework for instrumenting, generating, collecting, and exporting telemetry data (metrics, logs, and traces). Measurement data collected through OpenTelemetry can be transmitted to the WhaTap monitoring platform to analyze the performances and behaviors of applications or services and then resolve problems.

If active applications or services are collected using the OpenTelemetry library, you can select a method to get the traces, metrics, and log data into the WhaTap monitoring platform. Highly visible observability monitoring is provided using the dashboards and analysis menus on the WhaTap monitoring platform.

## OpenTelemetry agent

The OpenTelemetry agent processes and delivers to the WhaTap monitoring platform the data collected through the OTLP Exporter of OpenTelemetry Collector.



For the procedure to install and run the agent, see the following:

1. [Install and run the WhaTap agent](#) on the instance where OpenTelemetry Collector has been installed.
2. The launched whatap-otel service runs the Grpc server for OTLP Exporter.
  - The listening port is 6600 by default.
  - As in the default setting for OpenTelemetry Collector, the gzip compression setting is `True`.

## Support Environment

The operating systems on which the OpenTelemetry agent can be installed are:

- CentOS/Red Hat 6.x (64bit) or later
- Debian/Ubuntu 12.04 (64 bit) or later
- Amazon linux (64bit)
- Alpine Linux (64 bit)

# Install Agent

⚠ OpenTelemetry monitoring is a Beta version so unexpected errors may occur. It is not recommended for using it for sensitive data or in the operation environment. If you have any feedback or issues, feel free to contact the support team at [support@whatap.io](mailto:support@whatap.io).

It guides you how to install the OpenTelemetry agent.

## Download agent

Execute the following command to download the agent to the instance where the OpenTelemetry Collector has been installed.

- [Red Hat/CentOS](#) [Debian/Ubuntu](#) [Amazon Linux](#) [Alpine Linux](#) [Alpine Linux Docker](#)

*# Register package repository*

```
$ sudo rpm -Uvh https://repo.whatap.io/centos/5/noarch/whatap-repo-1.0-1.noarch.rpm
```

*# Install package*

```
$ sudo yum install whatap-otel
```

*# Register package repository*

```
$ wget https://repo.whatap.io/debian/release.gpg -O - | sudo apt-key add -
```

```
$ wget https://repo.whatap.io/debian/whatap-repo_1.0_all.deb
```

```
$ sudo dpkg -i whatap-repo_1.0_all.deb
```

```
$ sudo apt-get update
```

*# Install package*

```
$ sudo apt-get install whatap-otel
```

*# Register package repository*

```
$ sudo rpm --import https://repo.whatap.io/centos/release.gpg
```

```
$ echo "[whatap]" | sudo tee /etc/yum.repos.d/whatap.repo > /dev/null
```

```
$ echo "name=whatap packages for enterprise linux" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null
```

```
$ echo "baseurl=https://repo.whatap.io/centos/latest/\$basearch" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null
$ echo "enabled=1" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null
$ echo "gpgcheck=0" | sudo tee -a /etc/yum.repos.d/whatap.repo > /dev/null

Install package
$ sudo yum install whatap-otel
```

```
$ wget https://s3.ap-northeast-2.amazonaws.com/repo.whatap.io/alpine/x86_64/whatap-otel.tar.gz
$ tar -xvzf whatap-otel.tar.gz -C /
```

FROM alpine

FROM alpine

RUN apk update && apk upgrade

*# Install WhaTap Go monitoring agent*

RUN wget https://s3.ap-northeast-2.amazonaws.com/repo.whatap.io/alpine/x86\_64/whatap-otel.tar.gz

RUN tar -xvzf whatap-otel.tar.gz -C /

## Default agent settings

To configure the agent, the WhaTap project's access key and collection server IP address are required. In [WhaTap monitoring service](#), move to the project for the application product. In **Management > Agent Installation**, check the access key and collection server IP address for preparation.

## Configuring the whatap.conf file

Set the access key and collection server IP address in the agent configuration ([whatap.conf](#)) file.

```
echo "accesskey={ACCESS_KEY}" >> /usr/whatap/otel/whatap.conf
echo "whatap.server.host={COLLECTOR_SERVER_IP_ADDRESS}" >> /usr/whatap/otel/whatap.conf
```

## Configuring with environment variables

You can set the access key and collection server IP address as environment variables.

```
WHATAP_ACCESSKEY={ACCESS_KEY}
WHATAP_SERVER_HOST={COLLECTOR_SERVER_IP_ADDRESS}
WHATAP_OTEL_GRPC_SERVER_PORT={gRPC_SERVER_LISTENING_PORT}
```

ⓘ If both environment variables and the [whatap.conf](#) file have been configured, the settings in the [whatap.conf](#) file take precedence.

## Docker installation

If you have the OpenTelemetry Collector installed, you can run whatap-otel with a Docker container to receive data from the collector to the GRPC 6600 port.

```
docker pull whatap/whatap-otel
```

docker-compose.yml

```
version: '3.7'

services:
 whatap_otel:
 image: whatap/whatap-otel:latest

 environment:
 WHATAP_ACCESSKEY: x605111111111-z111111111-x1111111111m
 WHATAP_SERVER_HOST: 1.1.1.1/2.2.2.2
 WHATAP_OTEL_GRPC_SERVER_PORT: 6600
 otel_metric_server_enabled: true
 otel_log_server_enabled: true
 log_stdout_enabled: true
 #debug_otel_grpc_server_enabled: true

 ports:
 - 6600:6600
```

## Kubernetes installation

The following is a deployment example that creates the OpenTelemetry Collector and whatap-otel together in a single Pod.

otelcol-whatap-deploy.yaml

```

metadata:
 name: otel-collector-deploy
spec:
 replicas: 1
 selector:
 matchLabels:
 app: otel-collector
 template:
 metadata:
 name: otel-collector-pod
 labels:
 app: otel-collector
 spec:
 containers:
 - name: otel-collector
 image: otel/opentelemetry-collector-contrib:0.106.0
 ports:
 - containerPort: 4317 # OTLP gRPC
 - containerPort: 4318 # OTLP HTTP
 volumeMounts:
 - name: config-volume-otelcol
 mountPath: /etc/otel/config.yaml
 subPath: otel-collector-config.yaml
 args: ["--config", "/etc/otel/config.yaml"]
 - name: whatap-otel-agent
 image: whatap/whatap-otel:latest
 ports:
 - containerPort: 6600
 env:
 - name: "WHATAP_ACCESSKEY"
 value: "x60511111111-z11111111-x1111111111m"
 - name: "WHATAP_SERVER_HOST"

```

```

 value: "1.1.1.1/2.2.2.2"
 - name: "otel_metric_server_enabled"
 value: "true"
 - name: "otel_log_server_enabled"
 value: "true"
 - name: "log_stdout_enabled"
 value: "true"
 - name: NODE_IP
 valueFrom: {fieldRef: {fieldPath: status.hostIP}}
 - name: NODE_NAME
 valueFrom: {fieldRef: {fieldPath: spec.nodeName}}
 - name: POD_NAME
 valueFrom: {fieldRef: {fieldPath: metadata.name}}
 - name: whatap.micro.enabled
 value: "true"
volumes:
 - name: config-volume-otelcol
 configMap:
 name: otelcol-config

```

otelcol-whatap-deploy.yaml

```

apiVersion: v1
kind: ConfigMap
metadata:
 name: otelcol-config
data:
 otel-collector-config.yaml: |
 receivers:
 otlp:
 protocols:
 grpc:
 endpoint: 0.0.0.0:4317
 http:
 endpoint: 0.0.0.0:4318
 exporters:
 otlp:
 endpoint: "127.0.0.1:6600"
 tls:
 insecure: true

```

```

logging:
 loglevel: debug
processors:
 batch:
service:
pipelines:
 traces:
 receivers: [otlp]
 processors: [batch]
 exporters: [otlp, logging]
 metrics:
 receivers: [otlp]
 processors: [batch]
 exporters: [otlp, logging]
 logs:
 receivers: [otlp]
 processors: [batch]
 exporters: [otlp, logging]

```

## Configuring the Opentelemetry Collector

Configure the OTLP Exporter to forward data to whatap-otel.

otel-collector-config.yaml

```

1 receivers:
2 otlp:
3 protocols:
4 grpc:
5 endpoint: 0.0.0.0:4317
6 processors:
7 batch:
8
9 exporters:
10 logging:
11 # Configure the OTLP exporter to target whatap-otel.
12 otlp:
13 endpoint: 127.0.0.1:6600
14 tls:

```

```

15 insecure: true
16
17 service:
18 pipelines:
19 traces:
20 receivers:
21 - otlp
22 processors:
23 - batch
24 exporters:
25 - logging
26 - otlp
27 telemetry:
28 logs:
29 level: debug #activate debug mode

```

- End point ( `endpoint` ) setting: `127.0.0.1:6600`

The default port for local agents is 6600.

- `tls.secure` setting: `insecure: true`
- The gzip compression settings are used by default. (gzip is used)

ⓘ For more information about the OpenTelemetry Collector installation, see the following link.

- [Install the Collector](#)
- [Building a custom collector](#)

## Other platforms

### Tyk Gateway

Enable the OpenTelemetry function of [Tyk Gateway](#) and then check the transaction information through the OpenTelemetry Collector.

1. Connect the OpenTelemetry Collector to the Tyk Gateway.

tyk.conf

```
{
 "log_level": "debug",
 "listen_port": 8081,
 ...

 "opentelemetry": {
 "enabled": true,
 "endpoint": "111.222.333.444:4317",
 }
}
```

1. When the whatap-otel agent is connected to the OpenTelemetry Collector, trace data is collected.

## Kong Gateway

By configuring the built-in OpenTelemetry plugin in Kong Gateway, you can monitor transaction data through the OpenTelemetry Collector.

Specify the OpenTelemetry Collector endpoint in the OpenTelemetry plugin. When the Whatap-otel agent connects to the Collector, trace data will be collected.

1. kong.conf

Enable trace settings.

```
tracing_instrumentations = all
tracing_sampling_rate = 1
```

2. OpenTelemetry Plugin Configuration

Register the endpoint URL of the OpenTelemetry Collector.

The default OTLP receiver settings are as follows:

- Protocol: http
- Port: 4318

- URL:
  - Trace: /v1/traces
    - <http://x.x.x.x:4318/v1/traces>
  - Log: /v1/logs
    - <http://x.x.x.x:4318/v1/logs>

- Example default settings

```
traces_endpoint: http://otelcollector:4318/v1/traces
logs_endpoint: http://otelcollector:4318/v1/logs
```

- Example kong.yaml configuration

```
plugins:
- name: opentelemetry
 config:
 traces_endpoint: http://otelcollector:4318/v1/traces
 logs_endpoint: http://otelcollector:4318/v1/logs
 resource_attributes:
 service.name: kong-otel
```

- Example plugin configuration via Admin API

```
curl -X POST http://localhost:8001/services/plugins \
--header "accept: application/json" \
--header "Content-Type: application/json" \
--data '
{
 "name": "opentelemetry",
 "config": {
 "traces_endpoint": "http://otelcollector:4318/v1/traces",
 "logs_endpoint": "http://otelcollector:4318/v1/logs",
 "resource_attributes": {
 "service.name": "kong-dev"
 }
 }
}
```

```
}
,
```

- **Supported version:** Kong Gateway 3.3.x or higher

ⓘ For more detailed configuration instructions, refer to the following documentation.

- [Set up Kong Gateway](#)
- [OpenTelemetry Configuration](#)

## Starting the monitoring

- If the application server is restarted after all settings are made, the agent collects data. Go to the **Dashboard** > **Application dashboard** menu and then perform monitoring for the collected data in real time.
- For more information about the agent configuration, see [the following](#).

# Agent Configuration

⚠ OpenTelemetry monitoring is a Beta version so unexpected errors may occur. It is not recommended for using it for sensitive data or in the operation environment. If you have any feedback or issues, feel free to contact the support team at [support@whatap.io](mailto:support@whatap.io).

You can configure the agent to collect and process various metrics from OpenTelemetry.

## Agent

- **otel\_grpc\_server\_enabled** Boolean

Default `true`

Enable the Grpc server to receive the data delivered via the OpenTelemetry protocol (OTLP).

- **otel\_grpc\_server\_port** String

Default `6600`

Listening port of the Grpc server.

⚠ After changing the port number, be sure to restart the whatap-otel service.

- **otel\_grpc\_server\_gzip\_enabled** Boolean

Default `true`

It enables `gzip` by encrypting the data from the Grpc server.

- **debug\_otel\_grpc\_server\_enabled** Boolean

Default `false`

It enables the output of debugging logs for processing the OpenTelemetry Trace data.

## Agent Log

- **log\_keep\_days** Day **Int**

Default `7`

Set the retention period of log files.

- **log\_stdout\_enabled** **Boolean**

Default `false`

Add the option to output to Stdout as the same as the file log.

## Trace

Agent setting option for processing the OpenTelemetry Trace data. WhaTap gets values from the `attributes` object of **Span**, an OpenTelemetry Trace element, and uses them as monitoring information.

See the following code to see how the `attributes` object of **Span** relates to the WhaTap's agent configuration.

OpenTelemetry - Span example

```

1 {
2 "name": "/v1/sys/health",
3 "context": {
4 "trace_id": "7bba9f33312b3dbb8b2c2c62bb7abe2d",
5 "span_id": "086e83747d0e381e"
6 },
7 "parent_id": "",
8 "start_time": "2021-10-22 16:04:01.209458162 +0000 UTC",
9 "end_time": "2021-10-22 16:04:01.209514132 +0000 UTC",
10 "status_code": "STATUS_CODE_OK",
11 "status_message": "",
12 "attributes": {
13 "net.transport": "IP.TCP",
14 "net.peer.ip": "172.17.0.1",
15 "net.peer.port": "51820",
16 "net.host.ip": "10.177.2.152",
17 "net.host.port": "26040",

```

```

18 "http.method": "GET",
19 "http.target": "/v1/sys/health",
20 "http.server_name": "mortar-gateway",
21 "http.route": "/v1/sys/health",
22 "http.user_agent": "Consul Health Check",
23 "http.scheme": "http",
24 "http.host": "10.177.2.152:26040",
25 "http.flavor": "1.1"
26 },
27 "events": [
28 {
29 "name": "",
30 "message": "OK",
31 "timestamp": "2021-10-22 16:04:01.209512872 +0000 UTC"
32 }
33]
34 }

```

whatap.conf

```

otel_attr_tx_host_keys=http.host
otel_attr_tx_method_keys=http.method
otel_attr_tx_useragent_keys=http.user_agent

```

- ❗ • For more information about the WhaTap trace analysis, see [the following](#).
- For more information about the trace data processing for OpenTelemetry, see [the following link](#).
- For more information about the attributes provided by OpenTelemetry, see the following links:
  - [Database](#)
  - [Error](#)
  - [Host](#)
  - [HTTP](#)
  - [Network](#)
  - [URL](#)
  - [User agent](#)

- **otel\_trace\_server\_enabled** Boolean

Default `true`

It enables to process the OpenTelemetry Trace data.

- **otel\_trace\_queue\_size** Int

Default `5000`

It sets the queue size for processing the OpenTelemetry Trace data.

- **otel\_span\_collector\_max** Int

Default `100000`

Set the maximum size (number) of the Span data buffers for the OpenTelemetry Trace data.

- **otel\_attr\_dbcon\_keys** String

Default `sql.system,db.connection_string,db.system`

Enter the attribute name to be used as DB Connection information. Using this information, WhaTap displays it as the **DB Connection** step.

- **otel\_attr\_sql\_keys** String

Default `db.statement,sql.query`

Enter the attribute name to use as SQL query information. Using this information, WhaTap displays it as the **SQL** step.

- **otel\_attr\_httpc\_keys** String

Default `url.full,http.url,http.target`

Enter the attribute name to use as the HTTP call. Using this information, WhaTap displays it as the **HTTP Call** step.

- **otel\_attr\_tx\_url\_keys** String

Default `http.url,http.route,http.target,tyk.api.path,http.url`

Enter the attribute name to use as the service URL for transactions.

- **otel\_attr\_tx\_host\_keys** String

Default `http.host`

Enter the attribute name to use as the host name for transactions.

- **otel\_attr\_tx\_method\_keys** *String*  
Default `http.method,http.request.method`  
Enter the attribute name to use as the HTTP method.
- **otel\_attr\_tx\_status\_keys** *String*  
Default `http.status_code`  
Enter the attribute name to use as the HTTP status code for transactions.
- **otel\_attr\_tx\_useragent\_keys** *String*  
Default `http.user_agent`  
Enter the attribute name to use as the User-Agent for transactions.
- **otel\_attr\_tx\_clientip\_keys** *String*  
Default `net.peer.ip,net.socket.peer.addr`  
Enter the attribute name to use as the client IP for transactions.
- **otel\_attr\_tx\_referer\_keys** *String*  
Enter the attribute name to use as the HTTP Referer for transactions.
- **otel\_attr\_error\_keys** *String*  
Default `error.type`  
Enter the attribute name to use as the error information for transactions.
- **otel\_events\_exception\_keys** *String*  
Default `exception.type`  
Specify the attribute name of the event data to use as the error type.
- **otel\_events\_error\_message\_keys** *String*  
Default `exception.message`  
Specify the attribute name of the event data to use as the error message.
- **otel\_events\_error\_stack\_keys** *String*  
Default `exception.stacktrace`  
Specify the attribute name of the event data to use as the error stack.

- **otel\_tx\_path\_setby\_name\_enabled** Boolean

Default `false`

Specify the transaction name as the span name.

- **otel\_mtrace\_stepid\_setby\_trace\_spanid\_enabled** Boolean

Default `false`

Set the SpanID of the trace span to be recognized as ParentSpanID for external connection. Forcibly set it when there is no HTTP call span.

- **otel\_virtual\_httpc\_enabled** Boolean

Default `false`

For distributed tracing, forcibly set it when there is no HTTP call span. Set the HTTP call address by adding `otel_virtual_httpc_name` to the transaction name (URL).

- **otel\_virtual\_httpc\_name** String

Default: `external http call`

Forcibly specify the name when there is no HTTP call span.

- **transaction\_status\_error\_enable** Boolean

Default `false`

Set to display as an error if the status code is 400 or higher.

- **tx\_status\_meter\_enabled** Boolean

Default `true`

It collects the number of cases for each status code of the entire URL every 10 seconds.

- **stat\_txstatus\_enabled** Boolean

Default `true`

The number of status codes is collected every 5 minutes for each URL.

## Gateway

Agent setting option for processing the OpenTelemetry Metric data.

- **otel\_gateway\_enabled** Boolean

Default `false`

It sets whether or not to collect the query statistics data.

- It collects the elapsed time for each transaction, gateway, and API call for the entire URL every 10 seconds.
- It collects the elapsed time for each transaction, gateway, and API call for each URL and status every 5 minutes.

- **otel\_attr\_tx\_gw\_time\_keys** String

Default `duration.kong`

Specify the attribute name (key) that has the gateway's elapsed time.

- **otel\_attr\_tx\_api\_time\_key** String

Default `duration.proxy`

Set the elapsed time of the API call with the value of a specified attribute. Specify the attribute name (key) that has the API call's elapsed time.

- **otel\_replace\_tx\_resp\_time\_enabled** Boolean

Default `duration.proxy`

For transactions, the response time is calculated based on the start time and end time. The feature to change the response time for transactions is enabled with the key value of the specified attribute.

- For the response time, set the key value of a specific attribute to `otel_attr_replace_tx_resp_time_keys`.

- **otel\_attr\_replace\_tx\_resp\_time\_keys** String

Default `duration.kong`

Specify the key name of the attribute to be set as the response time.

## Metric

Agent setting option for processing the OpenTelemetry Metric data.

! For more information about the metric data processing for OpenTelemetry, see [the following link](#).

- **otel\_metric\_server\_enabled** Boolean

Default false

It enables to process the OpenTelemetry Metric data.

- **otel\_metric\_queue\_size** Int

Default 5000

It sets the queue size for processing the OpenTelemetry Metric data.

## Log

Agent setting option for processing the OpenTelemetry Log data.

! For more information about the log data processing for OpenTelemetry, see [the following link](#).

- **otel\_log\_server\_enabled** Boolean

Default false

It enables to process the OpenTelemetry Log data.

- **otel\_log\_queue\_size** Int

Default 5000

It sets the queue size for processing the OpenTelemetry Log data.

# Release Notes

**Latest Version: v0.2.7**

Loading data. Please wait a moment.

# Download Agent

## Latest Version 0.2.7

OS	File Format	Architecture	Download
Red Hat / CentOS / Amazon Linux	RPM	x86 x64	<a href="#">Download</a>
		aarch64	<a href="#">Download</a>
Ubuntu 12 and above / Debian	DEB	amd64	<a href="#">Download</a>
		arm64	<a href="#">Download</a>
Alpine Linux	tar gzip	x86 x64	<a href="#">Download</a>
		aarch64	<a href="#">Download</a>

ⓘ For details on how to view data collected with **OpenTelemetry** in the WhaTap monitoring service, see the [following](#).