

Extensions

release date. 2026. 09. 11.

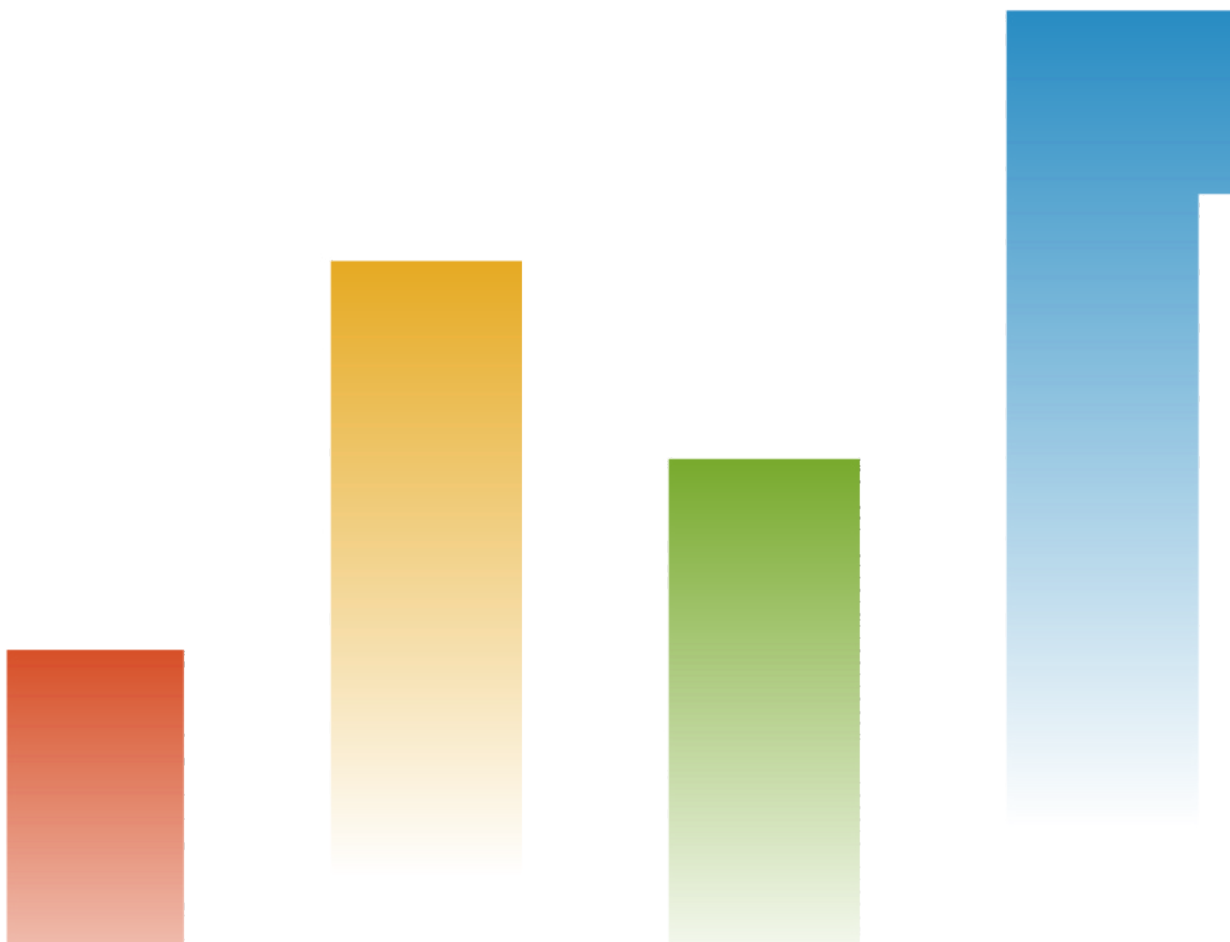


Table of Contents

• 拡張機能(Extensions)	3
◦ Focus	4
▪ サポート環境	5
▪ インストール	6
▪ 設定	8
▪ 使用例	15
◦ Telegraf	18
▪ プラグインのインストールと活用	
▪ プラグインインストール	21
▪ 設定	28
▪ エージェントのインストールと活用	
▪ サポート環境	30
▪ エージェントのインストール	31
▪ インストール問題の解決	36
▪ 設定	37
▪ 管理	40
▪ 使用例	43
◦ OpenTelemetry	47
▪ エージェントのインストール	49
▪ エージェントの設定	59
▪ リリースノート	0
▪ エージェントをダウンロード	68

拡張ツール(Extensions)

WhaTapサービスの収集データを拡張できるツールを紹介します。以下の資料に加え、追加内容のリクエストなどのフィードバックは、docs@whatap.ioまで送ってください。

Focus

4 項目

Telegraf

3 項目

OpenTelemetry

4 項目

Focusとは?

Focusは、WhaTapモニタリングで提供されていない任意のデータを時系列でアップロードするプログラムです。

運用チームが自社開発したチェックスクリプトやデータベースクエリを定期的または一回のみ操作して収集した値をアップロードできます。アップロードされた値は、そのプロジェクトの分析・メトリクスチャートメニューで確認できます。また、Flexボードに追加することができます。

Focusを使用するには、WhaTapプロジェクトの中からServer、Applicatoon(Java、PHP、NodeJS、Python)、Database、Kubernetesプロジェクトのいずれかを選択した後に、プロジェクトアクセスキー文字列と収集サーバーIPを実行オプションと指定して使用します。

サポート環境

X86環境で動作するWindows、Linux、FreeBSD、OSXおよびARMv5、6、7およびARM64 LinuxOSに適用できます。

サービス	環境	サポート環境
Focus	Windows	Windows 2008R2以降
	Linux	Debian 7.0以降
		Ubuntu 12.04以降
		CentOS、Red Hat 6.0以降
		Amazon Linux 1.0以降
		SUSE 12.1以降
	FreeBSD	FreeBSD 10以降
	OSX	Lion 10.7以降
	Raspberry Pi OS	8以降

ⓘ 対応一覧に含まれていない場合は、support@whatap.ioまでお問い合わせください。

インストール

WhaTap Focusは、単一実行ファイルで構成され、個別のインストールや登録プロセスが不要です。実行ファイルをダウンロードして任意の場所から実行できます。

実行ファイルをダウンロード

WhaTap Focusは、単一の実行ファイルで構成され、バージョンごとにダウンロードしてすぐに使用できます。

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ファイル構成



- **Linux / FreeBSD / OSX / Raspberry Pi OS**
- **focus:** データ収集および送信プログラムです。
- **Windows**
- **focus.exe:** データ収集および送信プログラムです。

設定

WhaTapプロジェクトから任意の収集データをアップロードできます。Focusは、プロジェクトの基本エージェントがインストールされていなくても使用できます。

プロジェクトアクセスキーと収集サーバーのIP確認

選択したプロジェクトの[管理](#) > [エージェントインストール](#) メニューからプロジェクトコード、収集サーバーIPとプロジェクトアクセスキーを確認します。

- [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
```

フィルターに特定のタグを追加

ユーザーは、フィルターで利用する特定の文字列をタグとして設定できます。設定により、WhaTapモニタリングサービス画面で収集されたデータをそのタグの名前と値でフィルタリングできます。

- [Linux Shell](#) Windows Powershell

SH

```
1 # -tag.[ユーザータグのキー名][空白][ユーザータグの値]
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.[ユーザータグのキー名][空白][ユーザータグの値]
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 ...

```

通知を送信する

ユーザーのカスタム通知を直ぐに送信できます。通知を送信すると、Focusは終了します。

- [Linux Shell](#) Windows Powershell

SH

```

#通知の重大度を選択します。
level={info|warn|fatal}
#通知のタイトル
title=必要な通知のタイトル
#通知の本文
message=目的の通知の本文
./focus -license $WHATAP_LICENSE \
-pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
-alert -level $level -title $title -message $message

```

Powersehll

```

#通知の重大度を選択します。
$level="{info|warn|fatal}"
#通知の題名
$title="希望する通知の題名"

```

```
#通知の本文
$message="希望する通知の本文"
.\focus.exe -license $WHATAP_LICENSE `
  -pcode $WHATAP_PCODE -server.host $WHATAP_HOST `
  -alert -level $level -title $title -message $message
```

SQL Query結果の収集

ユーザー定義SQL Queryの実行結果を時系列で無期限に収集します。

- [Linux Shell](#) Windows Powershell

SH

```
#メトリクスカテゴリ
category=my_category
#データベースドライバ
driver={mysql|postgres}
#データベース接続情報
dburl="ID:パスワード@tcp(IP:ポート番号)/データベース"
#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

```
#メトリクスカテゴリ
$CATEGORY="my_category"
#データベースドライバ
$driver="{mysql|postgres}"
#データベース接続情報
$dburl="ID:パスワード@tcp(IP:ポート番号)/データベース"
#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
```

ログファイルキーワードマッチングログの収集

指定したテキストログファイルでログが発生する場合、キーワードを含めると、対応するログラインが収集されます。

• [Linux Shell](#) Windows Powershell

SH

```
#メトリクスカテゴリ
category=my_category
#ログファイル
LOG_FILE=ログファイルパス
#2つ以上のログキーワードを入力する際にキーワード間で使用する区切り文字
LOG_KEYWORDS_SEPERATOR=,
#ログキーワードを「,」でつないで入力
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

```
#メトリクスカテゴリ
$CATEGORY="my_category"
#ログファイル
LOG_FILE=ログファイルパス
#2つ以上のログキーワードを入力する際にキーワード間で使用する区切り文字
LOG_KEYWORDS_SEPERATOR=,
#ログキーワードを区切り文字でつないで入力
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ログ分析サービス

指定したテキストログファイルでログが発生した場合、WhaTapログ分析サービスにリアルタイムでアップロードします。

- **Linux Shell** Windows Powershell

SH

```

#カテゴリ
category=my_category
#ログファイルwildcard(*)、日付パターン(http://strftime.org)を含む
LOG_FILE=ログファイルパス
./focus -license $WHATAP_LICENSE \
-category $category \
-pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
-logsink $LOG_FILE

```

Powersehll

```

#カテゴリ
$CATEGORY="my_category"
#ログファイルwildcard(*)、日付パターン(http://strftime.org)を含む
$LOG_FILE=ログファイルパス
.\focus.exe -license $WHATAP_LICENSE `
-pcode $WHATAP_PCODE -server.host $WHATAP_HOST `
-category $CATEGORY `
-logsink $LOG_FILE

```

OSリソースの使用量の収集

Focusが実行しているサーバーのリソース使用量を収集します。

Linux Shell

```
#メトリクスカテゴリ
category=my_category
#ディスクモニタリングが有効かどうか
diskenabled=true|false
#モニタリングするディスクマウント
diskmount=/mypartition
#NICモニタリングが有効かどうか
nicenabled=true|false
#モニタリングするNIC
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
```

ランダムなプログラムとスクリプトの実行結果の収集

任意のプログラムおよびスクリプトを実行し、stdoutで出力される結果を継続的に収集します。WhaTap Focusはjson dictionary形式で、stdinとして入力されると、dictionaryのkey、valueを収集します。入力手段としてパイプを使用するので、バッファリングを無効にする必要があります。

- ① 継続的に収集するのではなく、1回限り収集する場合は、次のオプションを適用できます。 `-onetime`
- 収集時間を指定する場合は、次のオプションで指定できます。 `-now {unix epoch time(second)}`

Linux Shell

```
#メトリクスカテゴリ  
CATEGORY="my_category"
```

```
プログラムまたはスクリプト | \  
json dictionary形に再処理 | \  
./focus -license $WHATAP_LICENSE \  
-category $category \  
-pcode $WHATAP_PCODE -server.host $WHATAP_HOST
```

次の例は、topコマンドを実行して、特定のプロセスのCPUとメモリ使用量を継続的に収集します。

Linux Shell

```
#メトリクスカテゴリ  
CATEGORY="my_category"
```

```
export PID=収集するプロセスのPID  
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
```

使用例

Focusを使用して、繋栄に使用されるコマンドやその他の活用例について案内します。これにより、開発や運用時にデータを時系列で確認できるようになり、開発や運用に役立ちます。

プロジェクトアクセスキーと収集サーバーのIP確認

選択したプロジェクトの[管理](#) > [エージェントインストール](#) メニューからプロジェクトコード、収集サーバーIPとプロジェクトアクセスキーを確認します。

- [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
```

フィルターに特定のタグを追加

次の例を参照して、WhaTapモニタリングサービス画面で収集されたデータをそのタグの名前と値でフィルタリングできます。

SH

```
1 # -tag.[ユーザータグのキー名][空白][ユーザータグの値]
2
3 ./focus -license $WHATAP_LICENSE \
4   -pcode $WHATAP_PCODE -server.host $WHATAP_HOST \
```

```

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

```

TOPコマンドのpid別CPU、Memory収集

次の例は、topコマンドを実行して、特定のプロセスのCPUとメモリ使用量を継続的に収集します。

Linux Shell

```

#メトリクスカテゴリ
CATEGORY="my_category"

export PID=収集するプロセスのPID
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

```

NETSTAT活用

次の例は、netstatコマンドを実行して、TCP Connectionステータス別の番号を収集します。

Linux Shell

```

#メトリクスカテゴリ
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

```

VMSTAT活用

次の例は、vmstatコマンドを実行して、項目された指数を収集します。

Linux Shell

```
#メトリクスカテゴリ
CATEGORY="my_category"

vmstat -n 5 | awk 'NR>2 {printf "{\r\n":%s,\r\n":%s,\r\nswpd":%s,\r\nfree":%s,\r\nbuff":%s,\r\n cache":%s,\r\n si":%s,\r\n so":%s,\r\n bi":%s,\r\n bo":%s,\r\n in":%s,\r\n cs":%s,\r\n us":%s,\r\n sy":%s,\r\n id":%s,\r\n wa":%s,\r\n 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
```

DU活用

次の例は、duコマンドを実行して、任意のディレクトリ容量を収集します。

Linux Shell

```
#メトリクスカテゴリ
CATEGORY="my_category"
TARGET=容量を収集したいディレクトリ

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
```

Telegrafとは

Telegrafは、データベース、システム、IoTセンサーからメトリクスとイベントを収集および送信するためのプラグインベースのサーバーエージェントです。TelegrafはGoで作成されており、外部の依存関係のない単一のバイナリとしてコンパイルされ、最小限のメモリリソースを利用します。

- **全種類のデータ収集・送信**

- **データベース**

- MongoDB、MySQL、Redisなどのデータソースに接続し、メトリクスを収集および送信します。

- **システム**

- 最新のクラウドプラットフォーム、コンテナ、オーケストレータースタックからメトリクスを収集します。

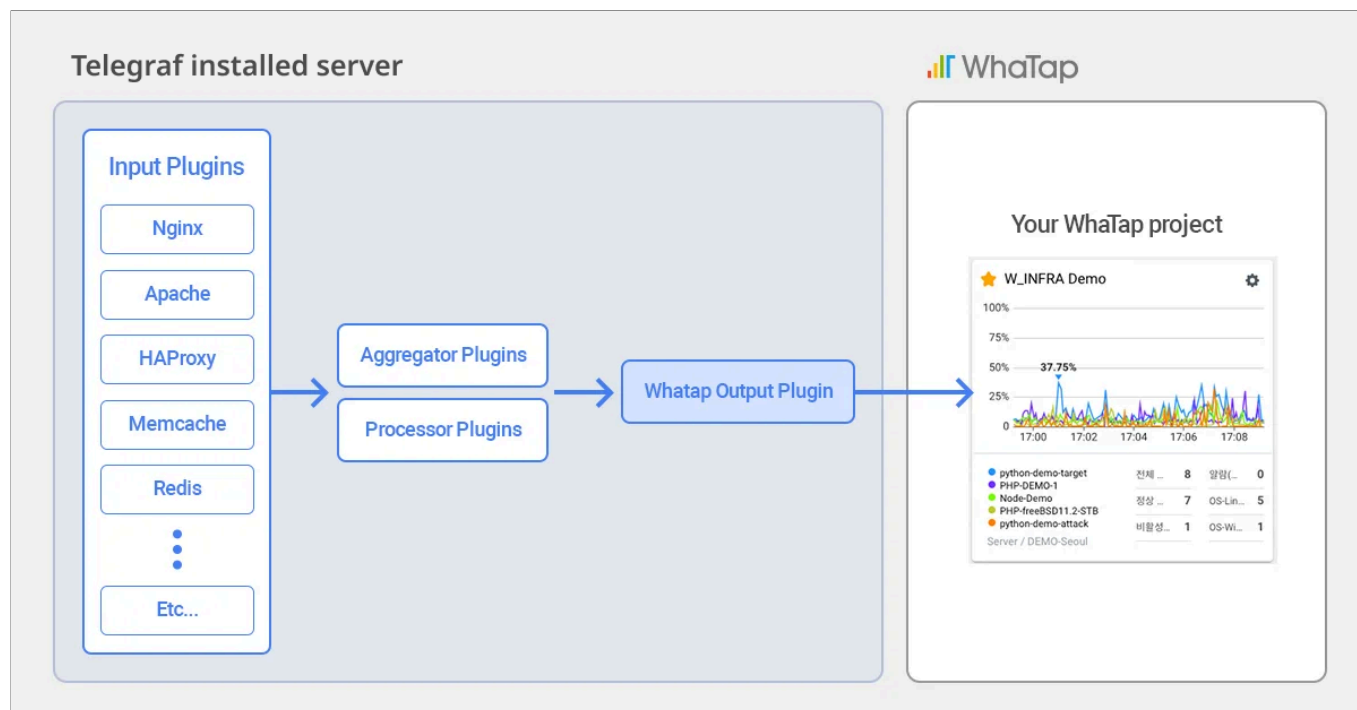
- **IoTセンサー**

- IoTセンサーとデバイスから重要なステータスの保存データ(圧力レベル、温度レベルなど)を収集します。

- **Telegrafプラグイン**

Telegrafは、メトリクスを収集、処理、集計、および作成するプラグインベースのエージェントです。入力、出力、アグリゲーター、プロセッサーを含む4つのカテゴリのプラグインを提供します。

TelegrafのWhaTap output pluginを活用し、WhaTapのTelegrafエージェントをインストールして、Telegrafで収集するメトリクスをWhaTap収集サーバーに送信できます。



ⓘ Telegrafプラグインまたはエージェントは、ユーザーがサービスに直接インストールおよび設定する必要があります。

プラグインまたはエージェントのインストール方法と使用例を次のように提供します。

📄 プラグインのインストールと活用

2 項目

📄 エージェントのインストールと活用

5 項目

使用例

使用例を紹介します。

プラグインインストール

whatap output plugin

Telegrafのwhatap output plugin(tcp)を使用して、Telegrafで収集したメトリクスをWhaTap収集サーバーに送信することができます。収集されたデータは、WhaTapのプロジェクトで確認できます。

⚠ Telegrafプラグインは、ユーザーがサービスに直接インストールして設定する必要があります。

サポート環境

次の一覧にあるOSは、ダウンロードおよびインストールできます。その他のOSは、[コンパイルインストール](#)に進みます。

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

ダウンロードインストール

Telegraf releaseバージョンを基準にwhatap pluginを追加して、事前に作成したインストールファイルです。Telegrafが既にインストールされている場合は、実行ファイルを置き換えるだけでwhatap pluginを有効にすることができます。

Telegraf releaseバージョン

- [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> (実行ファイル)
- <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> (実行ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
- 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)

- 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - 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> (実行可能ファイル)
 - <https://repo.whatap.io/telegraf/telegraf-release-1.12/windows/amd64/>

[telegraf-1.12.0~842282d_windows_amd64.zip](#)

コンパイル インストール

事前準備条件

- Go SDK構成
- go depインストール
- ソースファイル

`${GOPATH} /src/github.com/influxdata/telegraf/` ディレクトリの下に圧縮ファイルを解凍します。

① Telegraf releaseバージョンにwhatap pluginを含むソースファイルです。

- 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

インストール手順

① ビルドに関する詳細は [Telegraf github](#) ページを参照してください。

`${GOPATH} /src/github.com/influxdata/telegraf` 以下のMakefileでビルドを進めてください。

1. 依存関係ファイルのインストールを進めてください。依存関係ファイルのインストールで、go depが使用されるため、go depを最初にインストールする必要があります。

```
# make deps
```

2. 実行ファイルをビルドしてください。 `${GOPATH}/bin` ディレクトリに実行ファイル(telegraf)が生成されます。

```
# make go-install
```

3. インストールパッケージが必要な場合は、 `make package` コマンドを使用して作成できます。クロスコンパイルして、OSのパッケージファイルを作成してください。

```
# make package
```

設定

Telegrafのconfigファイル([telegraf.conf](#))に次の **outputs.whatap** 設定を有効にしてください。

⚠ Telegrafプラグインは、ユーザーがサービスに直接設定する必要があります。

whatap pluginを含むconfigファイルは、次のように生成できます。

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

whatapプロジェクト情報

[ホーム画面](#)でプロジェクトを選択し、プロジェクトの情報を設定してください。接続するプロジェクトのエージェントインストール手順で確認できます。

- プロジェクトアクセスキー：プロジェクトを識別するプロジェクトアクセスキーです。
- プロジェクトコード：プロジェクトを識別する数値型コードです。
- 収集サーバー情報(IP、ポート番号)：収集情報の配信先となるWhaTap収集サーバーのIPとポート番号です。

config設定(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 = "x2tgtnopk2t9-xxxxxxxx-aaaaaaaa"
#
# ## 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]]` コメント(#)を削除してください。
- `license` のコメント(#)を削除してください。プロジェクトのプロジェクトアクセスキー情報を入力します。文字型です。
- `pcode` のコメント(#)を削除してください。プロジェクトコードを入力します。数値型です。
- `servers` のコメント(#)を削除してください。収集サーバー情報(IP、ポート番号)を設定してください。プロトコルはtcpのみ対応しています。文字型で、"tcp://IP:ポート"を登録してください。配列型で登録してください。コンマ(,)で区切り、配列型で登録してください。

 Telegrafを再起動して変更した設定を適用してください。

サポート環境

WhaTapのTelegrafエージェントをインストールする前に動作環境を確認してください。

- **OS**

- Linux Red Hat 6.x x64以降
- Ubuntu 12.x x64以降
- FreeBSD 10.x x64以降

- **Telegraf**

- Telegraf 1.3以降

① socket_writeroutput pluginに対応するバージョンのみ利用可能です。

エージェントのインストール

WhaTapのTelegrafエージェントを使用するための基本的なインストール手順について説明します。

Telegrafから収集するメトリクスをWhaTap収集サーバーに送信することができます。収集されたデータはWhaTapのプロジェクト内で確認できます。このページでは、whatap-telegrafエージェントをインストールし、Telegrafから受け取ったデータを収集サーバーに送信する方法について説明します。

既にインストールされているTelegrafの `socket_writer output plugin` を有効化し、whatap-telegraf(Local TCP) エージェントにデータを渡します。

WhaTapサービスを使用するには、[会員登録](#)後にプロジェクトを作成し、対象サーバーにエージェントをインストールする必要があります。会員登録に関する詳細については [次の文書](#) を参照してください。

❗ Telegrafエージェントは、ユーザーが該当サービスに直接インストールおよび設定する必要があります。

プロジェクト作成

エージェントをインストールする前にプロジェクトを作成してください。

1. [WhaTapモニタリングサービス](#)にログインしてください。
2. 左側のサイドメニューで **プロジェクト全体** > **Project** ボタンをクリックします。
3. **商品選択** 画面でインストールする製品を選択します。
4. 以下の項目を入力または選択します。
 - **プロジェクト名**：プロジェクト名を入力します。
 - **データサーバーリージョン**：データサーバーの設置されているリージョンを選択します。リージョンとは、クラウドサービスを提供するデータセンターの集合体です。選択したリージョンにユーザーデータが保存されます。
 - **プロジェクトタイムゾーン**：アラートおよびレポート生成時の基準となる時刻を設定します。
 - **通知言語の設定**：アラート通知メッセージの言語を設定します。（韓国語、英語に対応）
 - **プロジェクトグループ**：複数のプロジェクトをグループとしてまとめて管理できます。所属するグループ

がある場合は選択してください。

- **プロジェクト説明**：プロジェクトに関する補足説明や詳細情報を入力します。

5. すべての設定が完了したら、**プロジェクトを作成する** ボタンをクリックします。

① 組織を選択してプロジェクトを追加する場合は、**組織下位のグループ**を必須に設定する必要があります。
グループに関する詳細は、[次のドキュメント](#)を参照してください。

プロジェクトアクセスキーの確認

プロジェクトアクセスキーは、WhaTapサービスを有効にするための固有IDです。

インストール手順セクションで**プロジェクトアクセスキーの発行**ボタンをクリックします。プロジェクトアクセスキーが自動的に発行されると、次の手順に進みます。

② プロジェクトを作成すると、自動的に**エージェントインストール**ページに移動します。**エージェントインストール**ページが表示されない場合は、左側のメニューから**全プロジェクト**を選択し、新しく作成したプロジェクトを選択します。

インストール手順とファイル構成

1. WhaTap リポジトリをインストールしてください。
2. whatap-telegraf Linuxパッケージ(yum, apt-get)をインストールしてください。
3. プロジェクトアクセスキー および WhaTapサーバー情報を設定してください。
4. Telegrafの `socket_writer output plugin` を設定し、データをリンクさせてください。

① Telegrafエージェントファイル構成

- `whatap_telegraf`



サービス実行可能ファイルです。Telegrafから渡された情報を収集サーバーに送信するプログラムです。

- `/etc/init.d/whatap-telegraf` (FreeBSD `/usr/local/etc/whatap_telegraf`)

サービススクリプトです。

- `whatap.conf`

設定ファイルです。収集サーバーのアドレスとサーバーのプロジェクトアクセスキーが入力するファイルです。

- `whatap-telegraf-#.log`

エージェントログファイルです。(`/usr/whatap/telegraf/logs`)

パッケージをインストール

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

1. WhaTap リポジトリをインストールしてください。

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

2. 次のコマンドを実行し、パッケージをインストールします。

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

3. 設定スクリプトを実行してサービスを開始します。

```
echo "license=[発行されたプロジェクトアクセスキー]" | sudo tee /usr/whatap/telegraf/whatap.conf
echo "whatap.server.host=[WhaTapサーバーアドレス]" | sudo tee -a /usr/whatap/telegraf/whatap.conf
sudo service whatap-telegraf restart
```

1. WhaTap リポジトリをインストールしてください。

```
$ 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. 次のコマンドを実行し、パッケージをインストールします。

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

3. 設定スクリプトを実行してサービスを開始します。

```
echo "license=[発行されたプロジェクトアクセスキー]" | sudo tee /usr/whatap/telegraf/whatap.conf
echo "whatap.server.host=[WhaTapサーバーアドレス]" | sudo tee -a /usr/whatap/telegraf/whatap.conf
sudo service whatap-telegraf restart
```

1. WhaTap リポジトリをインストールしてください。

```
$ 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. 次のコマンドを実行し、パッケージをインストールします。

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

3. 設定スクリプトを実行してサービスを開始します。

```
echo "license=[発行されたプロジェクトアクセスキー]" | sudo tee /usr/whatap/telegraf/whatap.conf
echo "whatap.server.host=[WhaTapサーバーアドレス]" | sudo tee -a /usr/whatap/telegraf/whatap.conf
sudo service whatap-telegraf restart
```

1. WhaTap リポジトリをインストールしてください。

```
$ 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. 設定スクリプトを実行してサービスを開始します。

```
echo "license=[発行されたプロジェクトアクセスキー]" | tee /usr/whatap/telegraf/whatap.conf
echo "whatap.server.host=[WhaTapサーバーアドレス]" | tee -a /usr/whatap/telegraf/whatap.conf
sudo service whatap_telegraf restart
```

Telegraf連携

telegraf.confに `socket_writer output plugin` 設定を行い、収集された情報をwhatap-telegrafに渡します。

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]]` のコメントを解除します。
- `address`項目にtcp接続を設定します。
- データタイプは、`json`形式として設定します。

Telegrafエージェントのインストールを完了しました。[次の文書](#)でインストール時のトラブルを解決する方法を確認してください。

インストール問題の解決

whatap-telegrafサービス(Service)が実行されていない場合、または、エラーが発生した場合は、サービスを再起動します。

❗ Telegrafプラグインおよびエージェントは、ユーザーが独自のサービスに直接インストールして設定する必要があります。

- Red Hat/CentOS

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

- Debian/Ubuntu

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

- FreeBSD

```
$ service whatap_telegraf restart
```

設定

WhaTapエージェントは、エージェントごとに必要な設定を`whatap.conf`ファイルに書き込みます。エージェントは、環境変数を使用して設定ファイルを見つけて読み込みます。`whatap.conf`ファイルを使用して設定できるオプションは次のとおりです。

ⓘ Telegrafエージェントは、ユーザーが直接設定する必要があります。

エージェント名の識別

WhaTapは、モニタリング情報の収集対象のインフラストラクチャーサーバーの識別のために、基本的にサーバーから収集した情報を使用します。

- ⓘ エージェント名は、プロジェクトごとに一意である必要があります。
- エージェントID／名前を変更しても、古いデータには関連付けられません。

- **object_name** String

既定値 `{type} - {hostname} - {ip2} - {ip3} - {docker}`

アプリケーションを識別するためのエージェント名(ONAME) 構成方式です。ONAMEに基づいてOIDが生成されます。

名称	説明
<code>{type}</code>	app_nameに設定した値を使用します。既定値は、Telegrafです。
<code>{ip#}</code>	IPを分割するときは、#番目の桁を使用します。
<code>{hostname}</code>	サーバーのホスト名を使用します。
<code>{docker}</code>	DockerコンテナIDを使用します。

- **app_name** String

アプリケーションを識別するためのエージェント名(ONAME)の構成要素です。 `object_name` の `{type}` に対応する値です。

① whatap-telegrafのサービスを再起動した後に適用されます。

エージェントの通信設定

- **license** String

エージェントのインストール時にサーバーから受け取ったプロジェクトアクセスキーを設定します。 プロジェクトアクセスキーには、エージェントが属するプロジェクトとセキュリティ通信のための暗号化キーが含まれています。

- **whatap.server.host** ip_address

既定値 127.0.0.1,127.0.0.1

エージェントが収集したデータを送信するサーバーを指定します。収集サーバーを冗長化して2つ以上のIPがある場合、コンマ(,)で区分して指定できます。指定されたIPは、収集サーバーのproxyデーモンがリスニング状態でサービスする必要があります。

- **whatap.server.port** tcp_port

既定値 6600

収集サーバーのポート番号を指定します。ポートは1つしか指定できないため、 `whatap_server_host` で指定した収集サーバーは同じポート番号を使用する必要があります。

- **tcp_so_timeout** Millisecond

既定値 60000

収集サーバーと通信するTCPセッションのSocket Timeout値を指定します。

- **tcp_connection_timeout** Millisecond

既定値 5000

収集サーバーと通信するTCPセッションのConnection Timeout値を指定します。

- **net_send_max_bytes** Byte

既定値 5242880

収集サーバーにデータを送信するときに、一度に送信される最大サイズを指定します。

- **net_send_buffer_size** Byte

既定値 1024

データ送信を行う必要がある最大バイトのサイズです。

Telegraf

- **telegraf_tcp_port** tcp_port

既定値 6600

Telegraf socker_writer output pluginが接続するTCPのポート番号です。

① whatap-telegrafのサービスを再起動した後に適用されます。

- **telegraf_delta_fields** String

既定値 1024

累積値を持つフィールドの増加値を計算し、別のフィールドを追加します。name.fields形式として指定し、複数
はコンマ(,)で区切ります。

例、nginx.accepts、nginx.requestsnginxのaccepts、requests値の増加値をacceptsDelta、requestsDeltaフィールド名として追加送信します。

ログ

- **log_keep_days** Day

既定値 7

ログファイルの保存期間を設定します。

管理

Telegrafエージェントを管理するために、[whatap.conf](#) ファイル使用して設定できる追加オプションについて説明します。

アップデート

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

1. パッケージ情報の更新のためにキャッシュ情報を削除してください。

```
$ yum clean all
```

2. whatap-telegrafパッケージをアップデートしてください。

```
$ yum update whatap-telegraf
```

3. 特定のバージョンをインストールするには、インストール時にバージョン情報を明示してください。

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

1. パッケージ情報の更新のためにキャッシュ情報を削除してください。

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

2. whatap-telegrafパッケージをアップデートしてください。

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

3. 特定のバージョンをインストールするには、インストール時にバージョン情報を明示してください。

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

1. パッケージを再インストールしてください。

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. 特定のバージョンをインストールするために、`{version}` にバージョン情報を入力してインストールしてください。

SH

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

一時停止

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

SH

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

SH

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

SH

```
$ service whatap_telegraf stop
```

削除

- パッケージ(yum, apt-get)を削除してください。
- [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
```
```

- 必要に応じて、`_usr/whatap/telegraf_`の下にあるログ ファイルおよびその他のファイルを削除してください。

使用例

Telegrafの詳細については、influxdataの[Telegraf documentation](#)を参照してください。

❗ Telegrafプラグインおよびエージェントは、ユーザーが独自のサービスに直接インストールして設定する必要があります。

SNMP

SNMP input pluginを使用して、SNMP性能データをリアルタイムで収集および、チャート化します。収集されたデータは、`telegraf_snmp` および、`telegraf_interface`` のカテゴリで検索できます。

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@xxx"
```

```

## 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

SNMP Trap input pluginを使用して、SNMP Trapイベントをリアルタイムで収集し、メール、SMSメッセージ、メッセージャーなどで送信できます。収集されたデータは `telegraf_snmp_trap` カテゴリで検索することができます。

Linux Shell

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

OpenTelemetry 概要

❗ OpenTelemetryのモニタリングはベータ版で、予期せぬ挙動やエラーが発生する可能性があります。重要なデータがある環境や運用環境では使用を推奨しません。フィードバックや問題点が発生したら、サポートチーム(support@whatap.io)までお問い合わせください。

この文書は、WhaTapモニタリングサービスのアプリケーション商品を利用しながらOpenTelemetryを通じてデータを追加収集しようとするユーザーのための文書です。

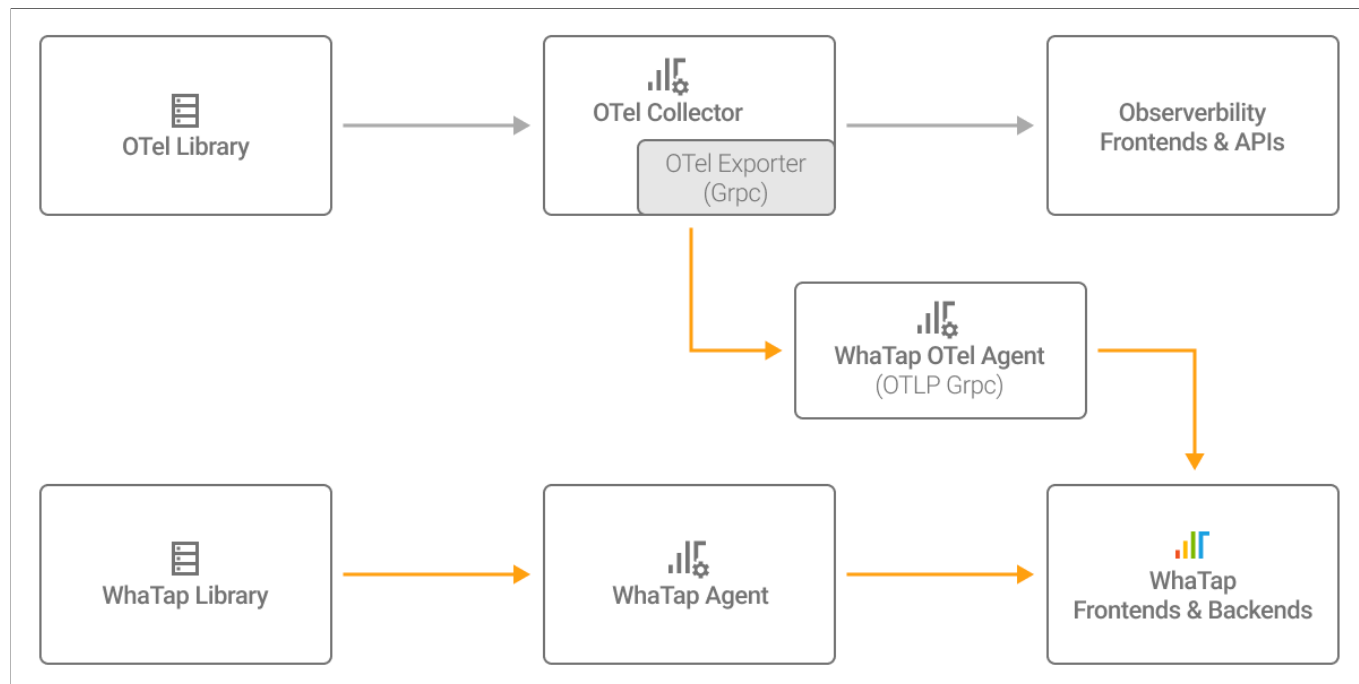
OpenTelemetryを通じてWhaTapモニタリングを使用する

OpenTelemetryは、遠隔測定データ(メトリクス、ログ、及び追跡)を計測、生成、収集、エクスポートするためのオープンソースオブザーバビリティ(Observability)フレームワークです。OpenTelemetryを通じて収集した測定データをWhaTapモニタリングプラットフォームに転送することで、アプリケーションやサービスの性能と動作を分析し、問題を解決できます。

もし、運用中のアプリケーションまたはサービスがOpenTelemetryライブラリに収集される場合は、追跡及び指標、ログデータをWhaTapモニタリングプラットフォームに持ってくる方法を選択できます。WhaTapモニタリングプラットフォームのダッシュボード及び分析メニューを活用して、可視性の高いオブザーバビリティモニタリングを提供します。

OpenTelemetryエージェント

OpenTelemetryエージェントは、OpenTelemetry CollectorのOTLP Exporterを通じて収集されたデータをWhaTapモニタリングプラットフォームに加工・伝達するエージェントです。



エージェントのインストールと実行のためのプロセスについては、以下を参考にしてください。

1. OpenTelemetry Collectorがインストールされたインスタンスに[WhaTapエージェント](#)をインストールして実行します。
2. 実行されたwhatap-otelサービスは、OTLP ExporterのためのGrpcサーバーを実行します。
 - リスニングポートは、基本6600ポートです。
 - OpenTelemetry Collectorの基本設定と同様に、gzip圧縮設定は `True` となります。

サポート環境

OpenTelemetryエージェントをインストールできるOSは、次のとおりです。

- CentOS/Red Hat 6.xバージョン(64bit) 以降
- Debian/Ubuntu 12.04バージョン(64bit) 以降
- Amazon linux (64bit)
- Alpine linux (64bit)

エージェントのインストール

⚠ OpenTelemetryのモニタリングはベータ版で、予期せぬ挙動やエラーが発生する可能性があります。重要なデータがある環境や運用環境では使用を推奨しません。フィードバックや問題点が発生したら、サポートチーム(support@whatap.io)までお問い合わせください。

OpenTelemetryエージェントのインストール方法を案内します。

エージェントダウンロード

次のコマンドを実行してOpenTelemetry Collectorがインストールされているインスタンスにエージェントをダウンロードします。

- [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 /
```

エージェントのデフォルト設定

エージェント設定には、WhaTapプロジェクトのアクセスキーと収集サーバーのIPアドレスが必要です。 [WhaTapモニタリングサービス](#)でアプリケーション(Application)商品のプロジェクトに移動します。 **管理 > エージェントインストール** メニューでアクセス キーと収集サーバーのIPアドレスを確認して準備します。

whatap.confファイルを設定する

エージェント設定([whatap.conf](#))ファイルへのアクセスキーと収集サーバーのIPアドレスを設定します。

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

環境変数で設定する

環境変数(Environment)でアクセスキーと 収集サーバーのIP アドレスを設定できます。

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

ⓘ 環境変数と`whatap.conf`ファイルをすべて設定した場合、`whatap.conf`ファイルの設定値が優先されます。

Dockerインストール

Opentelemetry Collectorがインストールされていれば、Docker containerでwhatap-otelを実行してCollectorのデータをGRPC 6600ポートで受け取ることができます。

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

docker-compose.yaml

```
version: '3.7'  
  
services:  
  whatap_otel:  
    image: whatap/whatap-otel:latest  
  
    environment:  
      WHATAP_ACCESSKEY: x605111111111-z111111111-x111111111m  
      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インストール

Opentelemetry Collectorとwhatap-otelを一つのPODの中に一緒に生成するdeploymentの例です。

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-z111111111-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]

```

Opentelemetry Collectorの設定

whatap-otelでデータを伝達するようにOTLP Exporterを設定します。

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   # otlp exporterにwhatap-otelを対象として設定します。
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

```

- エンドポイント(`endpoint`)設定： `127.0.0.1:6600`

ローカルにインストールしたエージェントが使用するデフォルトのポートは 6600 です。

- `tls.secure` 設定： `insecure: true`
- gzip圧縮設定をデフォルトで使用します。(gzip使用)

❗ OpenTelemetry Collectorインストールの詳細については、次のリンクを参考にしてください。

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

その他のプラットフォーム

Tyk Gateway

[Tyk Gateway](#)のOpenTelemetry機能を活性化した上でOpenTelemetry Collectorを通じてトランザクション情報を確認することができます。

1. OpenTelemetry CollectorをTyk Gatewayに接続します。

tyk.conf

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

  ...

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

1. 接続されたOpenTelemetry Collectorにwhatap-otelエージェントが接続されたら、トレース情報を収集します。

Kong Gateway

Kong Gatewayの内蔵OpenTelemetryプラグインを設定すると、OpenTelemetry Collectorを通じてトランザクション情報を確認できます。

OpenTelemetryプラグインで、OpenTelemetry Collectorのエンドポイントを指定します。CollectorにWhatap-otelエージェントが接続されると、トレース情報が収集されます。

1. kong.conf

trace設定を有効化します。

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

2. OpenTelemetryプラグインの設定

OpenTelemetry CollectorのエンドポイントURLを登録します。

CollectorのOTLPレシーバーのデフォルト設定は以下のとおりです。

- 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>

- 基本設定の例

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

- kong.yaml の設定例

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

- Admin APIによるプラグイン設定の例

```
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"
    }
  }
}
```

```
}  
}  
,
```

- 対応バージョン：Kong Gateway 3.3.x 以上

ⓘ 設定の詳細については、以下のリンクを参照してください。

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

モニタリングの開始

- すべての設定を完了した後、アプリケーションサーバーを再起動すると、エージェントが情報を収集します。**ダッシュボード > アプリケーションダッシュボード**メニューに移動して、収集されたデータをリアルタイムでモニタリングしてください。
- エージェント設定の詳細については、[次の文書](#)を確認してください。

エージェントの設定

⚠ OpenTelemetryのモニタリングはベータ版で、予期せぬ挙動やエラーが発生する可能性があります。重要なデータがある環境や運用環境では使用を推奨しません。フィードバックや問題点が発生したら、サポートチーム(support@whatap.io)までお問い合わせください。

OpenTelemetryから様々な指標を収集し、処理するようにエージェントを設定できます。

エージェント

- `otel_grpc_server_enabled` **Boolean**

既定値 `true`

OpenTelemetry protocol(OTLP)を通じて転送されるデータを受信するためのGrpcサーバーを活性化します。

- `otel_grpc_server_port` **String**

既定値 `6600`

GrpcサーバーのListenポートです。

ⓘ ポート番号を変更した後は、whatap-otelサービスを再起動する必要があります。

- `otel_grpc_server_gzip_enabled` **Boolean**

既定値 `true`

Grpcサーバから読み込むデータの暗号化により `gzip` を活性化します。

- `debug_otel_grpc_server_enabled` **Boolean**

既定値 `false`

OpenTelemetry Traceデータ処理に関するデバッグログの出力を活性化します。

エージェントログ

- `log_keep_days` Day `Int`

既定値 7

ログファイルの保存期間を設定します。

- `log_stdout_enabled` `Boolean`

既定値 `false`

ファイルログと同じようにStdoutで出力するオプションを追加します。

Trace

OpenTelemetry Traceデータ処理に関するエージェント設定オプションです。WhaTapは、OpenTelemetry Traceの構成要素であるSpanの `attributes` オブジェクトから値を取得し、モニタリング情報として活用します。

Spanの `attributes` オブジェクトがWhaTapのエージェント設定とどのように関連付けられているか、次のコードを参考にしてください。

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

```

- ❗ • Whatapのトレース分析に関する詳細は、[次の文書](#)を参考にしてください。
- OpenTelemetryのTraceデータ処理の詳細については、[次のリンク](#)を参考にしてください。
- OpenTelemetryで提供する属性(`attributes`)詳細については、次のリンクを参考にしてください。
 - [Database](#)
 - [Error](#)
 - [Host](#)
 - [HTTP](#)
 - [Network](#)
 - [URL](#)
 - [User agent](#)

- **otel_trace_server_enabled** Boolean

既定値 true

OpenTelemetry Traceデータ処理を活性化します。

- **otel_trace_queue_size** Int

既定値 5000

OpenTelemetry Traceデータ処理のためのキューサイズを設定します。

- **otel_span_collector_max** Int

既定値 100000

OpenTelemetry TraceデータのSpanデータバッファの最大サイズ(個数)を指定します。

- **otel_attr_dbcon_keys** String

既定値 sql.system,db.connection_string,db.system

DB Connection情報として活用する属性名を入力します。その情報を活用してWhaTapでは**DB Connection**ステップで表示します。

- **otel_attr_sql_keys** String

既定値 db.statement,sql.query

SQLクエリ情報として活用する属性名を入力します。その情報を活用してWhaTapでは**SQL**ステップで表示します。

- **otel_attr_httpc_keys** String

既定値 url.full,http.url,http.target

HTTP call情報として活用する属性名を入力します。その情報を活用してWhaTapでは**HTTP call**ステップで表示します。

- **otel_attr_tx_url_keys** String

既定値 http.url,http.route,http.target,tyk.api.path,http.url

トランザクションのサービスURL情報として活用する属性名を入力します。

- **otel_attr_tx_host_keys** String

既定値 http.host

トランザクションのホスト情報として活用する属性名を入力します。

- **otel_attr_tx_method_keys** String

既定値 `http.method,http.request.method`

HTTP Method情報として活用する属性名を入力します。

- **otel_attr_tx_status_keys** String

既定値 `http.status_code`

トランザクションのHTTPステータスコード情報として活用する属性名を入力します。

- **otel_attr_tx_useragent_keys** String

既定値 `http.user_agent`

トランザクションのUser-Agent情報として活用する属性名を入力します。

- **otel_attr_tx_clientip_keys** String

既定値 `net.peer.ip,net.socket.peer.addr`

トランザクションのクライアントIP情報として活用する属性名を入力します。

- **otel_attr_tx_referer_keys** String

トランザクションのHTTP Referer情報として活用する属性名を入力します。

- **otel_attr_error_keys** String

既定値 `error.type`

トランザクションのエラー情報として活用する属性名を入力します。

- **otel_events_exception_keys** String

既定値 `exception.type`

Eventデータの属性(Attribute)名を指定して、エラータイプの情報として活用します。

- **otel_events_error_message_keys** String

既定値 `exception.message`

Eventデータの属性(Attribute)名を指定して、エラーメッセージの情報として活用します。

- **otel_events_error_stack_keys** String

既定値 `exception.stacktrace`

Eventデータの属性(Attribute)名を指定して、エラースタックの情報として活用します。

- **otel_tx_path_setby_name_enabled** Boolean

既定値 false

トランザクション名をSpanの名前で指定します。

- **otel_mtrace_stepid_setby_trace_spanid_enabled** Boolean

既定値 false

Trace SpanのSpanIDを外部接続のParentSpanIDとして認識するように設定します。HTTP call span情報がないときに強制的に設定します。

- **otel_virtual_httpc_enabled** Boolean

既定値 false

分散追跡のためにHTTP call span情報がないときに強制的に設定します。HTTP callアドレスは、自分のトランザクション名(URL)に `otel_virtual_httpc_name` をつけて設定します。

- **otel_virtual_httpc_name** String

既定値 external http call

HTTP call span情報がないときに強制的に設定する名前を指定します。

- **transaction_status_error_enable** Boolean

既定値 false

status codeが 400 以上の場合、エラーとして表記するように設定します。

- **tx_status_meter_enabled** Boolean

既定値 true

全体URLのStatus Codeごとに件数を10秒間隔で収集します。

- **stat_txstatus_enabled** Boolean

既定値 true

URL別のStatus Codeの件数を5分間隔で収集します。

Gateway

OpenTelemetry Gatewayデータ処理に関するエージェント設定オプションです。

- **otel_gateway_enabled** Boolean

既定値 `false`

ゲートウェイ統計情報の収集可否を設定します。

- 全URLのトランザクション、ゲートウェイ、API呼び出しそれぞれの経過時間10秒間隔で収集します。
- URL、status毎にトランザクション、ゲートウェイ、APIの呼び出しそれぞれの経過時間を5分間隔で収集します。

- **otel_attr_tx_gw_time_keys** String

既定値 `duration.kong`

ゲートウェイ経過時間を持つ属性(attribute)名(key)を指定します。

- **otel_attr_tx_api_time_key** String

既定値 `duration.proxy`

指定された属性(attribute)の値でAPIの呼び出し経過時間を設定します。API呼び出し経過時間を持つ属性(attribute)名(key)を指定します。

- **otel_replace_tx_resp_time_enabled** Boolean

既定値 `duration.proxy`

トランザクションは、開始時間と終了時間で応答時間を計算します。指定した属性のキー値でトランザクションの応答時間を変更する機能を活性化します。

- 応答時間を特定属性(attribute)のキー値は `otel_attr_replace_tx_resp_time_keys` に設定します。

- **otel_attr_replace_tx_resp_time_keys** String

既定値 `duration.kong`

応答時間に設定する属性のキー名を指定します。

Metric

OpenTelemetry Metricデータ処理に関するエージェント設定オプションです。

❗ OpenTelemetryのMetricデータ処理の詳細については、[次のリンク](#)を参考にしてください。

- **otel_metric_server_enabled** Boolean

既定値 false

OpenTelemetry Metricデータ処理を活性化します。

- **otel_metric_queue_size** Int

既定値 5000

Opentelemetry Metricデータ処理のためのキューサイズを設定します。

Log

OpenTelemetry Logデータ処理に関するエージェント設定オプションです。

❗ OpenTelemetryのLogデータ処理の詳細については、[次のリンク](#)を参考にしてください。

- **otel_log_server_enabled** Boolean

既定値 false

Opentelemetry Logデータ処理を活性化します。

- **otel_log_queue_size** Int

既定値 5000

Opentelemetry Logデータ処理のためのキューサイズを設定します。

リリースノート

最新バージョン v0.2.7

データを読み込んでいます。少々お待ちください。

エージェントをダウンロード

最新バージョン 0.2.7

| OS | ファイル形式 | アーキテクチャ | ダウンロード |
|---------------------------------|----------|---------|------------------------|
| Red Hat / CentOS / Amazon Linux | RPM | x86 x64 | ダウンロード |
| | | aarch64 | ダウンロード |
| Ubuntu 12 以降 / Debian | DEB | amd64 | ダウンロード |
| | | arm64 | ダウンロード |
| Alpine Linux | tar gzip | x86 x64 | ダウンロード |
| | | aarch64 | ダウンロード |

ⓘ OpenTelemetryで収集したデータをWhaTapモニタリングサービスで確認する方法については、次の文書を参照してください: [次の文書](#)