ActivityWatch Documentation

Release 0.1.1

Erik Bjäreholt and contributors

Apr 10, 2017

User documentation

Note: ActivityWatch is currently under development and should not be considered stable software, yet.

CHAPTER 1

Introduction

ActivityWatch is a bundle of software that provides storage for life-logging data such as what you do on your computer.

What the system does is handle collection and retrieval of all kinds of logging data (relating to your life, your computer or any type of record of an event). aw-server provides a safe repository where you store your data, it is not a place for modification (providing data integrity), once a record is created, it is intended to be immutable.

What ActivityWatch is

- A set of watchers (i.e. afk-watcher, window-watchers) that record relevant information about what you do and what happens on your computer
- A way of storing data collected from a wide variety of sources in an immutable manner: Events added are persistent and the stored data cannot be changed unless it's first copied.
- Provides a common dataformat accomodating most needs in a flexible yet simple manner

What ActivityWatch isn't

- A tool for doing advanced data analysis
- A full-fledged data visualization tool

Reason for existence

There are plenty of companies offering services which do collection of Quantified Self data with goals ranging from increasing personal producivity to understanding the people that managers manage (organizational productivity). However, all known services suffer from a significant disadvantage, the users data is in the hands of the service providers which leads to the problem of trust. Every customer of these companies have their data in hands they are forced to trust if they want to use their service.

This is a significant problem, but the true reason that we decided to do something about it was that existing solutions were inadequate. They focused on short-term insight, a goal worthy in itself, but we also want long-term understanding. Making the software completely free and open source so anyone can {use, audit, improve, extend} it seemed like the only reasonable alternative.

Data philosophy

Raw data is always the most valuable data.

QS data doesn't take much space by todays standards, but when you are a service having thousand of customers, every megabyte per user counts.

For the users however, every megabyte of data is worth it. It is therefore of importance that we collect and store data in the highest reasonable resolution such that we later don't have to "fill the gaps" in incomplete or aggregated data with heuristics and trickery.

Many services doing collection and analysis of QS data today don't actually store the raw data but instead store only summaries or low-resolution data (such as summarizing all time within an interval, instead of storing the individual intervals). This is a problem today with existing services: they store summarized data instead of the raw data.

This is indicative of that they actually lack a long-term plan. They want to provide a certain type of analysis *today*, which is fine, but we expect to want to do some unknown analysis in the future, and for that we might need the raw data. And we suspect that we would rather choose how detailed our analysis should be then rather than saving a bit of space by reducing the data resolution and detail before storing it.

Simply put: it is of importance that we start collecting the raw data now, before it disappears into the aether.

Security

One of the reasons this project was started was due to the fact that we were missing security in how our Quantified Self data was stored. Data needs to be collected on many devices, and be stored at a central and secure location or distributed for redundancy.

Since we want to be able to provide a safe storage service for initial users who do not have the time to run a server of their own, we will provide a feature such that we will only have the users encrypted data, without information of the contents (with exception for some relatively unimportant metadata such as allocated storage space, sessions, clients, and number of entries).

NOTE: Security features discussed here are all considered work in progress and this software is not yet fit for exposure to the internet. Only allow connections from localhost!

CHAPTER 2

Getting started

Short introductory text should go here.

Content from aw-server/README.md should be moved here.

Installation

Nothing here yet.

Usage

Nothing here yet.

Best practice

Nothing here yet.

CHAPTER 3

Storing data

The server part of ActivityWatch, aw-server, by default comes with a few methods of storing data. As of 0.1.1 the default is the JSON store which simply stores each bucket in it's own JSON file.

Other methods include:

- MongoDB
- In-memory (non-persistent, useful in testing)

CHAPTER 4

User Interface

Tray icon

In order to ensure stability and user-awareness a tray icon is planned to get developed.

This may contain a way to monitor the status of the server as well as watchers.

Web Interface

In order to achieve user-friendlyness, ActivityWatch will in the future come with a web interface allowing for overview, monitoring and configuration.

This interface is likely to provide basic graphs, but more advanced and configurable visualization (such as the ones found in Zenobase and RescueTime) is not a priority and is unlikely to get implemented as a part of the core ActivityWatch project.

Before we can start work on an interface we need the core system functionality down, so this wont become a priority until then.

CHAPTER 5

Pausing logging

Feature status: Planned

The possibility to pause logging is a low-tech solution to filter sensitive data. It could most easily be implemented by instructing a watcher to simply stop logging, such functionality might be made universal to the client libraries which would in turn filter out all events from the pause command to the resume command.

CHAPTER 6

Filtering data

Feature: Planned

ActivityWatch was born out of a frustration with the privacy issues of existing life logging solutions. We feel that it's important that nothing gets logged that shouldn't be logged. This way the cost of data breach is bounded, and the barrier to sharing your own data for scientific purposes uses will hopefully become smaller.

This is expected to be almost impossible to perfect since what someone considers sensitive might not be for someone else (due to e.g. culture and law). But the basics are easy to get right: such as not logging private browser tabs by default.

For the ones who believe they can adequately protect their data, the option should will course be available to disable this filter.

Pausing logging is likely to get implemented before this, so if it's available and this feature isn't: Use it until we get this done.

CHAPTER 7

Development

Nothing here yet.

CHAPTER 8

Modules

Server

Known as aw-server, it handles storage and retrieval of all activities/entries in buckets. Usually there exists one bucket per client.

Watchers/Clients

Since aw-server doesn't do any data collection on it's own, we need watchers that observe the world and sent the data off to aw-server for storage.

Libraries

Since all watchers need a common set of client functionality, such as calling the APIs and handling when a server is unavailable, this behavior has been extracted to a set of client libraries.

Currently the primary client library is written in Python, and known simply as aw-client, but a client library written in JavaScript is on the way and will have the same level of support in the future.

CHAPTER 9

API

ActivityWatch uses a REST API that binds together aw-server and it's clients. Most applications should never need to access the API directly but should instead use the client libraries available for the language the application is written in. If no such library yet exists for a given language, this document is meant to provide enough specification to create one.

Warning: The API is currently under development, and is subject to change. It will be documented in better detail when first version has been frozen.

Security

Clients might in the future be able to have read-only or append-only access to buckets, providing additional security and preventing compromised clients from being able to cause a severe security breach. All clients should have a symmetric key used for encrypting data in transit, since we can't guarantee that hosts can provide valid SSL certificates.

Security is something we shouldn't dare to mess up, so the implementation is likely to be following a KISS approach awaiting further review and proposals of more sophisticated security schemes.

Buckets API

The most common API used by ActivityWatch clients is the API providing read and append access buckets. Buckets are data containers used to group data together which shares some metadata (such as client type, hostname or location).

The basic API endpoints are as follows:

Get bucket

```
GET /api/0/buckets/<bucket_id>
```

Create bucket

```
POST /api/0/buckets/<bucket_id>
```

Events API

The most common API used by ActivityWatch clients is the API providing read and append access buckets. Buckets are data containers used to group data together which shares some metadata (such as client type, hostname or location).

The basic API endpoints are as follows:

Get events

```
GET /api/0/buckets/<bucket_id>/events
```

Create event

```
POST /api/0/buckets/<bucket_id>/events
```

Heartbeat API

Warning: Experimental API, not yet ready for use.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search