SEAD Backend User documentation

Table of Contents

Installation

Installation of the Go Landing Zone and the Python API can be automated by using the Puppet modules included in the repository. The puppet modules are written for and assume to be executed on an Ubuntu 14.04 x64 Linux server. First you must install the prerequisites for running Puppet. From the terminal, execute:

sudo apt-get install puppet git

Copy the Puppet files onto the server, and execute Puppet:

sudo puppet apply --modulepath=puppet/modules/ puppet/manifests/site.pp

After Puppet has executed the modules correctly, the server should be listening on ports 8080 and 9000. Verify with netstat:

netstat -tln | egrep ':(8080|9000)'

Python API

The Python API is used to query the raw data contained in the database. It listens for HTTP GET requests on port 8080. When a request comes in to the URI “/” it returns a usage message, in Python style. Data can be requested of the API for devices by serial number by using the URI “/serial”. For example, to qeury the API running locally for data from serial 7, you would execute a GET request to 127.0.0.1:8080/7. The official server is db.sead.systems. Filters can be applied using the following GET parameters:

- **start_time:** The earliest data point to include in the query. Specified as a UTC Unix timestamp.
- **end_time:** The earliest data point to include in the query. Specified as a UTC Unix timestamp.
- **type:** The type of data to retrieve. Current options: I for current, T for temperature, W for power, V for voltage.
- **subset:** Select a subset of the data within the given filters such that the number of elements returned are evenly spaced within the range of data returned in the filter. This option may not return exactly specified number of data points. The number of data points returned should be close to the specified number. If the same query without a subset clause would return fewer than the specified number of rows, the subset option will have no effect.

Example API calls with parameters:

- http://db.sead.systems:8080/000002?type=T
  Returns all temperature data for the SEAD plug with serial number 000002.
- http://db.sead.systems:8080/000001?start_time=1416202050&end_time=1416202054
  Returns all types of data for the SEAD plug with serial number 000001 between the UTC Unix timestamps 1416202050 and 1416202054.
- http://db.sead.systems:8080/000001?type=W&subset=100
  Returns all power data for the SEAD plug with serial number 000001 downsampled to approximately 100 data points.

The service can be started using the init script installed via Puppet:

sudo service seadapi start

The Python API is the primary interface to be used by the front end designers to query the database for SEAD plug data, and facilitates the necessary action for our first user story: *As a GUI developer, I want to query the database for signals within a certain range*.

Our final goals for the API are currently being blocked by other teams. Querying based on device type is dependent on the analysis sub-team producing tested and working modules. Querying SEAD panel data is dependent on the SEAD panel team (not affiliated with CMPS 115) finishing the data transmission component of the SEAD panel.

Once we have the API prepared to accept information from the SEAD panel and the classification scripts, we will be able to generate a signature to classify the device based on the raw data received. This will allow us to proceed with our second user story: *As a GUI developer, I want to query the database for data by appliance type.*

Go Landing Zone

The Go Landing Zone is a high speed, concurrent TCP server which listens for connections from SEAD plugs on port 9000. The plug requests it configuration from the server before beginning to send the data it collects into its buffer. This stream of packets is decoded, and the data is bulk inserted into the Postgres database to be later queried by the API.

The landing zone is not intended to directly interacted with by a user. The data collected by the landing zone is accessible via the API. The correct way to test if the landing zone is functioning properly is to connect a SEAD plug and check if new data becomes available through the API.

Once we have the groundwork necessary to organize the protocol coordination between the SEAD panel and the landing zone, we will be able receive data from the SEAD panel and perform a classification of the raw data to determine the appliance type. This will complete our final user story: *As a SEAD panel developer, I want to send raw appliance data to the server and a classification result to be rendered in the GUI.*