Event Notification Server Documentation

Pliable Pixels

Jul 20, 2021

Contents

CHAPTER 1

Breaking Changes

1.1 Version 6.1.19 onwards

- `match_past_detections`, `past_det_max_diff_area` and `max_detection_size` (and associated label prefixes) need to be in the `general` section of `ml_sequence`. In the previous release they were stuffed inside each model sequence which lead to problems (imagine an event where snapshot and alarm had different objects and you were checking both. In pass 1, snapshot would match but alarm would not, so you'd see objects in alarm. In pass 2, alarm would match, but not snapshot and it would keep going on like this, effectively making match_past_detections useless. To avoid this, I am checking for match_past_detections *after* all matching is done)
- You can now choose to ignore certain labels when you match past detections using *ignore_past_detection_labels*
- `stream_sequence` now has a few new fields fields:
  - `delay_between_frames`. If specified, will wait for those many seconds before processing each frame.
  - `delay_between_snapshots`. If specified, will wait for those many seconds when processing snapshot frames. This allows you to do something like this: `frame_set: snapshot,snapshot, snapshot,alarm` with a `delay_between_snapshots:2`, which means it will keep analyzing snapshot 3 times, but with 2 seconds in between, which lets you grab multiple snapshot frames as it changes during an event. This is really only useful for this specific case.

1.2 Version 6.1.18 onwards

- I now support face detection using TPU (NOT recognition). See objectconfig.ini for an example
- You can now add descriptive names for each model sequence to better differentiate in logs
- Each model sequence now has an `enabled` flag (default is `yes`). If `no` that means the model won't be loaded. This is a good way to temporarily remove models while keeping config files intact

- We now also have a `same_model_strategy` value of `union` - when set to `union` it will combine detection from all models for that type

1.3 Version 6.1.17 onwards

- You can now localize `past_det_max_diff_area`, `max_detection_size` to specific objects by prefixing the object name. Example `car_max_detection_size` if present will override values for `max_detection_size` for objects that are cars. Same holds true for `car_past_det_max_diff_area` if `match_past_detections=yes`
- `mlapiconfig.ini` also supports the above values - you no longer have to keep putting these to `objectconfig.ini`

1.4 Version 6.1.12 onwards

- A lot of config changes, if you are using mlapi. Basically, I'm no longer fully supporting settings in objectconfig.ini transferring to mlapi. See *Exceptions when using mlapi*

1.5 Version 6.1.0 onwards

- You can now string togther multiple models in arbitrary fashion to suit your needs. There is a new entry in `objectconfig.ini` called `ml_sequence` that you can use to create your own sequence. Note that if `ml_sequence` is present, it will override any/all parameters in the `[object]`, `[face]` and `[alpr]` sections. Please read this section to understand how this works.
- The `hog` model has been removed. Note this refers to the `hog` person detection model, not the hog detection of a face. That still exists. With Yolo, TinyYolo, coral there was no need to support this very low performance model anymore.
- You can now also specify arbitrary frames for analysis. See See here for details (look at options attribute).
- To enable the new `*_sequence` attributes mentioned above, make `use_sequence=yes` in `objectconfig.ini`
- A new attribute, `disable_locks`. When set to `yes`, it will result in locks not being grabbed before inferencing. The entire idea of grabbing locks is so that you have control on how many simultaneous processes use your CPU/GPU/TPU resources, so I'd recommend you don't enable it. Set it to `yes` only if you are facing lock issues (such as timeouts)

1.6 Version 6.0.5 onwards

- You can now specify object detection patterns on a per polygon/zone basis. The format is `<polygonname>_zone_detection_pattern` This works for imported ZM zones too. Please read the comments in `objectconfig.ini`. Note that this attribute will not be automatically added to a migrated `objectconfig.ini` file as the actual attribute name will change depending on your zone name
- `zmeventnotification.ini` has a new attribute called `replace_push_messages` in the `[fcm]` section. When enabled (default is `no`), push messages will replace each other in the notification bar. This was the old Android behaviour prior to FCMv1. You can go back to this mode of operation for both iOS and Android if you enable this.

1.7 Version 6.0.1 onwards

- `zmeventnotification.ini` new attribute in `[fcm]` called `use_fcmv1` with a default of `yes`. It is recommended you keep this on as this switches from the legacy FCM protocol to the FCM v1 which allows for better features (which I will add over time).
- `objectconfig.ini` has a new attribute `fast_gif` in `[animation]`. If you are creating animations for push and generating GIFs, this creates a 2x speed GIF.

1.8 Version 6.0.0 onwards

- The ES has a new attribute in `[customize]` called `es_rules`. A sample file gets automatically installed when you run the install script in `/var/lib/zmeventnotification`
- Its use is optional. It is a JSON file with various rules for the ES that are not configuration related. Over the next few releases, this fill will replace the cryptic context of `tokens.txt` As of now, it can be used to specify custom times for notification. This list will grow over time.
- A new perl dependency (optional) has been added to *Installation of the Event Server (ES)* if you need flexible datetime parsing for ES rules.

On the Object Detection part:

- This is going to be a big bad breaking change release, but continues the path to unification between various components I've developed.
- To help with this 'big bad breaking change', I've provided an upgrade script. When you run `./install.sh` it will automatically run it at the end and put a `migrated-objectconfig.ini` in your current directory (from where you ran `./install.sh`) You can also run it manually by invoking `tools/config_upgrade.py -c /etc/zm/objectconfig.ini`
- All the ml code has now moved to pyzm and both local hook and mlapi use pyzm. This means when I update ml code, both systems get it right always
- This version also supports Google Coral Edge TPU
- Several `objectconfig.ini` attributes have been replaced and some removed towards this unification goal:
  - `models` is now `detection_sequence`
  - `yolo` is no longer used. Instead `object` is used. `object` could be multiple object detection techniques, yolo or otherwise.
  - `[ml]` is now `[remote]`
  - `[object]` is a new section, which contains two new attributes:
    * `object_framework` which can be `opencv` or `coral_edgetpu`
    * `object_processor` which can be `cpu`, `gpu` or `tpu`
  - `yolo_min_confidence` is now `object_min_confidence`
  - `config`, `weights`, `labels` are now `object_config`, `object_weights` and `object_labels` respectively.
  - None of the `tiny_` attributes exist anymore. Simply switch weights, labels and config files to switch between full and tiny
  - `yolo_type` doesn't exist anymore (as `tiny_` attributes are removed, so it doesn't make sense)
  - `alpr_pattern` is now `alpr_detection_pattern`

  - detect_pattern no longer exists. You now have a per detection type pattern, which allows you to specify patterns based on the detection type:
    * object_detection_pattern - for all objects
    * alpr_detection_pattern - for for license plates
    * face_detection_pattern - for all faces detected
    * [general] has various new attributes that allow you to limit concurrent processing:
      · cpu_max_processes specific how many simultaneous instances of model execution will be allowed at one time. When more than this number is reached, processes will wait till in-flight processes complete. cpu_max_lock_wait specifies how long each process will wait (default 2 mins) before throwing an error.
      · tpu_max_processes and tpu_max_lock_wait same as above but for TPU
      · gpu_max_processes and gpu_max_lock_wait same as above but for GPU

1.9 Version 5.15.7 onwards

- The <>/models/tinyyolo directory is now <>/models/tinyyolov3. install.sh will automatically move it, but remember to change your objectonfig.ini path if you are using tiny yolo.
- You now have an option to use the new Tiny Yolo V4 models which will be automatically downloaded unless you disabled it (You'll need OpenCV master as of Jul 11, 2020 as support for it was only merged 6 days ago)
- A new attribute, ject_area has been introduced in objectconfig.ini. This specifies the largest area a detected object should take inside the image. You can keep it as a % or px value. Remember the image is resized to 416x416. better to keep in %

1.10 Version 5.15.6 onwards

- I got lazy with 5.15.5. There were some errors that I fixed post 5.15 which I 'post-pushed' into 5.15.5. It is possible you installed 5.15.5 and don't have these fixes. In other words, if your 5.15.5 is broken, Please upgrade.
- In this release, I've also taken a necessary step towards model naming normalization. Basically, Yolo models are now YoloV3 and CSPN is now Yolov4. This is because this is the terminology Alexey has started using in his repo. This means you will have to change your objectconfig.ini and align it with the same objectconfig.ini provided in this repo. I've also normalized the names of the config, weights and name files for each model. The short of all of this is, look under the [yolo] section of the sample config and replace your current yolo paths. Note that I assume you use install.sh to install. If not, you'll have to manually rename the old model names to the new ones. (Note that YoloV4 requires OpenCV 4.4 or above)

1.11 Version 5.15.5 onwards

- zmeventnotification.ini has a new attribute, topic under [mqtt] which lets you set the topic name for the messages
- objectconfig.ini has a new attribute, only_triggered_zm_zones. When set to yes, this will remove objects that don't fall into zones that ZM detects motion in. Make sure you read the comments in objectconfig.ini above the attribute to understand its limitations

1.12 Version 5.14.4 onwards

- Added ability for users to PR contrib modules See *Guidelines for contrib*
- `zmeventnotification.ini` adds two new attributes that makes it simpler for users to keep object detection plugin hooks intact *and also* trigger their own scripts for housekeeping. See the ini script for documentation on `event_start_hook_notify_userscript` and `event_end_hook_notify_userscript`

1.13 Version 5.13.3 onwards

- New attribute `es_debug_level` in `zmeventnotification.ini` that controls debug level verbosity. Default is `2`
- New CSPNet support with ResNeXt (requires OpenCV 4.3 or above) - Note that this requires a **manual model download** as the model is in a google drive link and all automated download scripts are hacks that stop working after a while.
- You can now choose which models to download as part of `./install.sh`. See install-specific-models

1.14 Version 5.11 onwards

- If you are using platerecognition.com local SDK for ALPR, their SDK and cloud versions have slightly different API formats. There is a new attribute called `alpr_api_type` in `objectconfig.ini` that should be set to `local` to handle this.
- `skip_monitors` in `zmeventnotification.ini` is now called `hook_skip_monitors` to correctly reflect this only means hooks will be skipped for these monitors. A new attribute `skip_monitors` has been added that controls which monitors the ES will skip completely (That is, no analysis/notifications at all for these monitors)
- Added support for live animations as part of push messages. This requires an upgraded zmNinja app (`1.3.0.91` or above) as well as ZoneMinder master (1.35) as of Mar 17 2020. Without these two updates, live notifications will not work. Specifically: - This introduces a new section in `objectconfig.ini` called `[animation]`. Please read the config for more details. - You are also going to have to re-run `install.sh` to install new dependencies

1.15 Version 5.9.9 onwards

- You can now hyper charge your push notifications, including getting desktop notifications. See below
- I now support 3rd party push notification systems. A popular one is pushover that a lot of people seem to use for customizing the quality of push notifications, including critical notifications, quiet time et. al. This adds the following parameters: - A new section called `[push]` in `zmeventnotification.ini` that adds two new attributes: `use_api_push` and `api_push_script` - I've provided a sample push script that supports pushover. This gets automatically installed when you use `install.sh` into `/var/lib/zmeventnotification/bin/pushapi_pushover.py` - This also addes a new channel type called `api` to the pre-existing `fcm,web,mqtt` set. - You are of course, encouraged to write your own 3rd party plugins for push and PR back to the project. - Read more in this article

1.16 Version 5.7.7 onwards

- For those who are happy to use the legacy openALPR self compiled version for license plate detection that does not use DNNs, I support that. This adds new parameters to *objectconfig.ini*. See objectconfig.ini for new parameters under the "If you are using OpenALPR command line" section.

1.17 Version 5.7.4 onwards

- I know support the new OpenCV 4.1.2 GPU backend support for CUDA. This will only work if you are on OpenCV 4.1.2 and have compiled it correctly to use CUDA and are using the right architecture. - This adds a new attribute `use_opencv_dnn_cuda` in `objectconfig.ini` which by default is `no`. Please read the comments in `objectconfig.ini` about how to use this.
- The ES supports a control channel using which you can control its behavior remotely - This adds new attributes `use_escontrol_interface`, `escontrol_interface_file` and `escontrol_interface_password` to `zmeventnotification.ini`. Read more about it *Category: escontrol messages*.
- If you are using face recognition, you now have the option of automatically saving unknown faces to a specific folders. That way it's easy for you to review them later and retrain your known faces. - This introduces the following new attributes to `objectconfig.ini`: `save_unknown_faces`, `save_unknown_faces_leeway_pixels` and `unknown_images_path`. Their documentation is part of `objectconfig.ini`
- The detection script(s) now attach a JSON payload of the detected objects along with the text, separated by `--SPLIT--`. If you are hacking your own scripts, you need to handle this. The ES automatically handles it when sending notifications.

1.18 Version 5.2 onwards

- *use_hooks* is a new attribute that controls whether hooks will be used or not
- *send_event_end_notification* is a new attribute that controls whether end notifications are sent

1.19 Version 5.0 onwards

- `install.sh` no longer tries to install opencv on its own. You will have to install `opencv` and `opencv-contrib` on your own. See install instructions in *Machine Learning Hooks*.
- The `hook_script` attribute is deprecated. You now have `hook_on_event_start` and `hook_on_event_end` which lets you invoke different scripts when an event starts or ends. You also have the concepts of channels, that allows you to decide whether to send a notification even if hooks don't return anything. Read up about `notify_on_hook_success` and `notify_on_hook_fail` in `zmeventnotification.ini`
- Now that we support pre/post event hooks, the script names have changed too (`zm_detect_wrapper.sh` is `zm_event_start.sh` and we have a new script called `zm_event_end.sh` that is really just a dummy script. Change it to what you need to do at the end of an event, if you enable event end notifications)
- You can now offload the entire machine learning processes to a remote server. All you need to do is to use `ml_gateway` and related options in `objectconfig.ini`. The "ML gateway" is my mlapi project

- The ES now supports a `restart_interval` config item in `zmeventnotification.ini`. If not 0, this will restart the ES after those many seconds (example `7200` is 2 hours). This may be needed if you find the ES locking up after a few hours. I think 5.0 resolves this locking issue (see this issue) but if it doesn't use this, umm, hack for now.

1.20 Version 4.6 onwards

- If you are using hooks, make sure you run `sudo ./install.sh` again - it will create additional files in `/var/lib/zmeventnotification`
- The hook files `detect.py` and `detect_wrapper.sh` are now called `zm_detect.py` and `zm_detect_wrapper.sh`. Furthermore, these scripts no longer reside in `/usr/bin`. They will now reside in `/var/lib/zmeventnotification/bin`. I suppose I did not need to namespace and move, but I thought of the latter after I did the namespace changing.
- If you are using face recognition, 4.6.1 and above now allow multiple faces per person. Note that it is recommended you train them before you run detection. See the documentation for it in *Machine Learning Hooks*.

1.21 Version 4.4 onwards

- If you are using picture messaging, then the URL format has changed. Please REMOVE `&username=<user>&password=<passwd>` from the URL and put them into the `picture_portal_username` and `picture_portal_password` fields respectively

1.22 Version 4.1 onwards

- Hook versions will now always be `<ES version>.x`, so in this case `4.1.x`
- Hooks have now migrated to using a proper python ZM logger module so it better integrates with ZM logging
- To view detection logs, you now need to follow the standard ZM logging process. See *Logging* documentation for more details)
- You no longer have to manually install python requirements, the setup process should automatically install them
- If you are using MQTT and your `MQTT:Simple` library was installed a while ago, you may need to update it. A new `login` method was added to that library on Dec 2018 which is required (ref)

1.23 Version 3.9 onwards

- Hooks now add ALPR, so you need to run *sudo -H pip install -r requirements.txt* again
- See modified objectconfig.ini if you want to add ALPR. Currently works with platerecognizer.com, so you will need an API key. See hooks docs for more info

1.24 Version 3.7 onwards

- There were some significant changes to ZM (will be part of 1.34), which includes migration to Bcrypt for passwords. Changes were made to support Bcrypt, which means you will have to add additional libraries. See the installation guide.

1.25 version 3.3 onwards

- Please use `yes` or `no` instead of `1` and `0` in `zmeventnotification.ini` to maintain consistency with `objectconfig.ini`
- In `zmeventnotification.ini`, `store_frame_in_zm` is now `hook_pass_image_path`

1.26 version 3.2 onwards

- Changes in paths for everything. - event server config file now defaults to `/etc/zm`
- hook config now defaults to `/etc/zm`
- Push token file now defaults to `/var/lib/zmeventnotification/push`
- all object detection data files default to `/var/lib/zmeventnotification`
- **If you are migrating from a previous version:**
  - Make a copy of your `/etc/zmeventnotification.ini` and `/var/detect/objectconfig.ini` (if you are using hooks)
  - Run `sudo -H ./install.sh` again inside the repo, let it set up all the files
  - Compare your old config files to the news ones at `/etc/zm` and make necessary changes
  - Make sure everything works well
  - You can now delete the old `/var/detect` folder as well as `/etc/zmeventnotification.ini`
  - Run zmNinja again to make sure its token is registered in the new tokens file (in `/var/lib/zmeeventnotification/push/tokens.txt`)

CHAPTER 2

Installation of the Event Server (ES)

2.1 3rd party dockers

I don't maintain docker images, so please don't ask me any questions about docker environments. There are others who maintain docker images. I have no affiliation with any of them. Feel free to explore the various options below, but please don't ask me about them. I've also not tried any of these dockers.

- Alex's repo (in progress): Various ZM configurations, currently ZM and ZM+ES (no hooks)
- Vangorra's repo:A docker container with ZM+ES+hooks+MLAPI
- dlandon's repo: A ZM+ES+hooks container (may not be free/maintained anymore)
- The Moosman's repo: A docker container for MLAPI

If there are other repositories you are aware of that work well, let me know.

If you are using docker images, the next section does not apply.

2.2 Clone the repo

I'd recommend users download the latest stable release. Starting version 4.5 I have started tagging releases that are rested. The master branch will always be 'cutting-edge'.

2.2.1 To clone the latest stable release:

```
git clone https://github.com/pliablepixels/zmeventnotification.git
cd zmeventnotification
# repeat these two steps each time you want to update to the latest stable
git fetch --tags
git checkout $(git describe --tags $(git rev-list --tags --max-count=1))
```

2.2.2 To clone master:

```
git clone https://github.com/pliablepixels/zmeventnotification.git
# repeat these two steps each time you want to update
git checkout master # only needed if you are on some other branch later
git pull
```

2.3 Configure the ini files

- Edit `zmeventnotification.ini` to your liking. More details about various parts of the configuration are explained later in this readme
- If you are behind a firewall, make sure you enable port `9000`, TCP, bi-directional (unless you changed the port in the code)
- If you are _not_ using machine learning hooks, make sure you comment out the `hook_script` attribute in the `[hook]` section of the ini file or you will see errors and you will not receive push
- We now need to install a bunch of dependencies (as described below)

2.4 Install Dependencies

Note that I assume you have other development packages already installed like `make`, `gcc` etc as the plugins may require them. The following perl packages need to be added (these are for Ubuntu - if you are on a different OS, you'll have to figure out which packages are needed - I don't know what they might be)

(**General note** - some users may face issues installing dependencies via `perl -MCPAN -e "Module::Name"`. If so, its usually more reliable to get into the CPAN shell and install it from the shell as a 2 step process. You'd do that using `sudo perl -MCPAN -e shell` and then whilst inside the shell, `install Module::Name`)

- `Crypt::MySQL` (if you have updated to ZM 1.34, this is no longer needed)
- `Net::WebSocket::Server`
- `Config::IniFiles` (you may already have this installed)
- `Crypt::Eksblowfish::Bcrypt` (if you have updated to ZM 1.34, you will already have this)
- `Time::Piece` for parsing ES rules

Installing these dependencies is as simple as:

```
sudo perl -MCPAN -e "install Crypt::MySQL"
sudo perl -MCPAN -e "install Config::IniFiles"
sudo perl -MCPAN -e "install Crypt::Eksblowfish::Bcrypt"
```

If after installing them you still see errors about these libraries missing, please launch a CPAN shell - see General Note above.

If you face issues installing Crypt::MySQL try this instead: (Thanks to aaronl)

```
sudo apt-get install libcrypt-mysql-perl
```

If you face issues installing Crypt::Eksblowfish::Bcrypt, this this instead:

```
sudo apt-get install libcrypt-eksblowfish-perl
```

If there are issues installing Config::IniFiles and the errors are related to Module::Build missing, use following command to get this module in debian based systems and install Config::IniFiles again.

```
sudo apt-get install libmodule-build-perl
```

Next up install WebSockets

```
sudo apt-get install libyaml-perl
sudo apt-get install make
sudo perl -MCPAN -e "install Net::WebSocket::Server"
```

Then, you need JSON.pm installed. It's there on some systems and not on others In ubuntu, do this to install JSON:

```
sudo apt-get install libjson-perl
```

Get HTTPS library for LWP:

```
perl -MCPAN -e "install LWP::Protocol::https"
```

If you want to enable MQTT:

```
perl -MCPAN -e "install Net::MQTT::Simple"
```

If you are setting up MQTT:

- A minimum version of MQTT 3.1.1 is required
- If your `MQTT:Simple` library was installed a while ago, you may need to update it. A new `login` method was added to that library on Dec 2018 which is required (ref)

Note that starting 1.0, we also use `File::Spec`, `Getopt::Long` and `Config::IniFiles` as additional libraries. My ubuntu installation seemed to include all of this by default (even though `Config::IniFiles` is not part of base perl).

If you get errors about missing libraries, you'll need to install the missing ones like so:

```
perl -MCPAN -e "install XXXX" # where XXX is Config::IniFiles, for example
```

If you are also planning on using the machine learning hooks, you will need to make sure you have Python3 and pip3 installed and working properly. Refer to your OS package documentation on how to get Python3 and pip3.

2.5 Configure SSL certificate (Generate new, or use ZoneMinder certs if you are already using HTTPS)

NOTE: If you plan on using picture messaging in zmNinja, then you cannot use self signed certificates. You will need to generate a proper certificate. LetsEncrypt is free and perfect for this.

If you are using secure mode (default) you **also need to make sure you generate SSL certificates otherwise the script won't run** If you are using SSL for ZoneMinder, simply point this script to the certificates.

If you are not already using SSL for ZoneMinder and don't have certificates, generating them is as easy as:

(replace `/etc/zm/apache2/ssl/` with the directory you want the certificate and key files to be stored in)

```
sudo openssl req -x509 -nodes -days 4096 -newkey rsa:2048 -keyout /etc/zm/apache2/ssl/
↪zoneminder.key -out /etc/zm/apache2/ssl/zoneminder.crt
```

It's **very important** to ensure the `Common Name` selected while generating the certificate is the same as the hostname or IP of the server. For example if you plan to access the server as `myserver.ddns.net` Please make sure you use `myserver.ddns.net` as the common name. If you are planning to access it via IP, please make sure you use the same IP.

Once you do that please change the following options in the config file to point to your SSL certs/keys:

```
[ssl]
cert = /etc/zm/apache2/ssl/zoneminder.crt
key = /etc/zm/apache2/ssl/zoneminder.key
```

2.5.1 IOS Users

On some IOS devices and when using self signed certs, I noticed that zmNinja was not able to register with the event server when it was using WSS (SSL enabled) and self-signed certificates. To solve this, I had to email myself the zoneminder certificate (`zoneminder.crt`) file and install it in the phone. Why that is needed only for WSS and not for HTTPS is a mystery to me. The alternative is to run the eventserver in WS mode by disabling SSL.

2.6 Install the server (optionally along with hooks)

NOTE : By default `install.sh` moves the ES script to `/usr/bin`. If your ZM install is elsewhere, like `/usr/local/bin` please modify the `TARGET_BIN` variable in `install.sh` before executing it.

- You can now move the ES to the right place by simply doing `sudo ./install.sh` and following prompts. Other options are below:
- Execute `sudo ./install.sh --no-install-hook` to move the ES to the right place without installing machine learning hooks

2.7 Update the configuration files

When you install the ES, it comes with default configuration files. They key files are:

- `/etc/zm/zmeventnotification.ini` - various parameters that control the ES
- `/etc/zm/objectconfig.ini` - various parameters that control the machine learning hooks
- `/etc/zm/secrets.ini` - a common key/value mapping file where you store your personal configurations

You **always** have to modify `/etc/zm/secrets.ini` to your server settings. Please review the keys and update them with your settings. At the least, you will need to modify:

- `ZM_USER` - the username used to log into your ZM web console
- `ZM_PASSWORD` - the password for your ZM web console
- `ZM_PORTAL` - the URL for your ZM instance (typically `https://<domain>/zm`)
- `ZM_API_PORTAL` - the URL for your ZM API instance (typically `https://<portal>/api`)
- `ES_CERT_FILE` and `ES_KEY_FILE` - the certificates to use if you are using HTTPS

Next, You can/should run it manually at first to check if it works

2.8 Optional but Recommended: Making sure everything is running (in manual mode)

- Start the event server manually first using `sudo -u www-data /usr/bin/zmeventnotification.pl --debug` (Note that if you omit `--config` it will look for `/etc/zm/zmeventnotification.ini` and if that doesn't exist, it will use default values) and make sure you check syslogs to ensure its loaded up and all dependencies are found. If you see errors, fix them. Then exit and follow the steps below to start it along with Zoneminder. Note that the `-u www-data` runs this command with the user id that apache uses (in some systems this may be `apache` or similar). It is important to run it using the same user id as your webserver because that is the permission zoneminder will use when run as a daemon mode.
- Its is HIGHLY RECOMMENDED that you first start the event server manually from terminal, as described above and not directly dive into daemon mode (described below) and ensure you inspect syslog to validate all logs are correct and THEN make it a daemon in ZoneMinder. If you don't, it will be hard to know what is going wrong. See *this section* later that describes how to make sure its all working fine from command line.

2.9 Making sure the ES gets auto-started when ZM starts

- Go to your web interface, and go to `Options->Systems` and enable `OPT_USE_EVENTNOTIFICATION` and you are all set.
- If you plan on using the machine learning hooks, there is more work to do. Please refer to *Installation*.

The rest of this section is NOT NEEDED for 1.32.0 and above!

2.10 Set up logging correctly for troubleshooting

For quick debugging, you can just run the ES or hooks manually by adding `--debug` but for proper logging, follow steps *here*

CHAPTER 3

Key Principles - Event Notification Server and Hooks

3.1 Summary

This guide is meant to give you an idea of how the Event Notification Server (ES) works, how it invokes hooks and how notifications are finally sent out.

3.2 From Event Detection to Notification

3.2.1 1: How it starts

The ES is a perl process (typically `/usr/bin/zmeventnotification.pl`) that acts like just any other ZM daemon (there are many) that is started by ZoneMinder when it starts up. Specifically, the ES gets "auto-started" only if you have enabled `OPT_USE_EVENT_NOTIFICATION` in your `Zoneminder->System` menu options. Technically, ZM uses a 'control' process called `zmdc.pl` that starts a bunch of important daemons (see here for a list of daemons) and keeps a tab on them. If any of them die, they get restarted.

Configuration files

This may be a good place to talk about configuration files. The ES has many customizations that are controlled by `/etc/zm/zmeventnotification.ini`. If you are using hooks, they are controlled by `/etc/zm/objectconfig.ini`. Both these files use `/etc/zm/secrets.ini` to move personal information away from config files. Study both these ini files well. They are heavily commented for your benefit.

3.2.2 2: Detecting New Events

Once the ES is up and running, it uses shared memory to know when new events are reported by ZM. Basically, ZM writes data to shared memory (SHM) whenever a new event is detected. The ES regularly polls this memory to detect new events. This has 2 important side effects:

- The ES *must* run on the same server that ZM is running on. If you are using a multi-server system, you need an ES *per* server.
- If an event starts and ends before the ES checks SHM, this event may be missed. If you are seeing that happening, reduce `event_check_interval` in `zmeventnotification.ini`. By default this is set to 5 seconds, which means events that open and close in a span of 5 seconds have a potential of being missed, if they start immediately after the ES checks for new events.

3.2.3 3: Deciding what to do when a new event starts

When the ES detects a new event, it forks a sub-process to handle that event and continues its loop to listening for new events (by polling SHM). There is exactly one fork for each new event and that fork typically lives till the event is completely finished.

3.1: Hooks (Optional)

If you are *not* using hooks, that is `use_hooks=no` in `/etc/zm/zmeventnotification.ini` then directly skip to the next section.

The entire concept of hooks is to "influence" whether or not to actually send out a notification for a new event. If you are already using hooks, you are likely using the most popular hook that I wrote, which actually does object/person/face detection on the image(s) that constitute the event to make an intelligent decision on whether you really want to be notified of the event. If you recall, the initial reason why I wrote the ES was to send "push notifications" to zmNinja. You'd be inundated if you got a push for *every* new event ZM reports.

So when you have hooks enabled, the script that is invoked when a new event is detected by the ES is defined in `event_start_hook` inside `zmeventnotification.ini`. I am going to assume you did not change that hook script, because the default script does the fancy image recognition that lot of people love. That script, which is usually `/var/lib/zmeventnotification/bin/zm_event_start.sh` does the following:

- It invokes */var/lib/zmeventnotification/bin/zm_detect.py* that is the actual script that does the fancy detection and waits for a response. If this python file detects objects that meet the criteria in `/etc/zm/objectconfig.ini` it will return an exit code of `0` (success) with a text string describing the objects, else it will return an exit code of `1` (fail)
- It passes on the output and the return value of the script back to the ES
- At this stage, if hooks were used and it returned a success (`0`) and `use_hook_description=yes` in `zmeventnotification.ini` then the detection text gets written to the ZM DB for the event

The ES has no idea what the event start script does. All it cares about is the "return value". If it returns `0` that means the hook "succeeded" and if it returned any non `0` value, the script failed. This return code makes a difference on whether the final notification is sent out or not, as you will see later.

3.2: Will the ES send a notification?

So at this stage, we have a new event and we need to decide if the ES will send out a notification. The following factors matter:

- If you had hooks enabled, and the hook succeeded (i.e. return value of `0`), then the notification *may* be sent to the channels you specified in `event_start_notify_on_hook_success`.
- If the hook failed (i.e. return value of non zero, then the notification *may* be sent to the channels specified in `event_start_notify_on_hook_fail`)

Summary of rules:

- if hooks are used, needs to return 0 as exit status
- Then, if you use dynamic controls (`use_escontrol_interface=yes`), those commands will be checked
- Then, if you have a rule file (ES 6.0+), rules will have to allow it
- Then, channel must be in the notify_on_xxx attributes
- Then, if FCM, monitor must be in tokens.txt for that device
- Then, if FCM, delay must be > delay specified in tokens.txt

3.2.1: Wait what is a channel?

At a high level, there are 4 types of clients that are interested in receiving notifications:

- zmNinja: the mobile app that uses Firebase Cloud Messaging (FCM) to get push notifications. This is the "fcm" channel
- Any websocket client: This included zmNinja desktop and any other custom client you may have written to get notifications via web sockets. This is the "web" channel
- receivers that use MQTT. This is the "mqtt" channel.
- Any 3rd party push solution which you may be using to deliver push notifications. A popular one is "pushover" for which I provide a plugin. This is the "api" channel.

So, for example:

```
event_start_notify_on_hook_success = all
event_start_notify_on_hook_fail = api,web
```

This will mean when a new event occurs, everyone may get a notification if the hook succeeded but if the hook fails, only API and Web channels will be notified, not FCM. This means zmNinja mobile app will not be notified. Obviously, if you don't want to get deluged with constant notifications on your phone, don't put `fcm` as a channel in `event_Start_notify_on_hook_fail`.

3.2.2: The tokens.txt file

Why do I say above that you *may* get a notification?

You'd think if the channels conditions are met and the hook conditions are met, then those channels *will* get a notification. Not quite.

Note: `tokens.txt` is another "configuration" file that impacts the decision process for sending a notification out. This only applies to the "fcm" channel (i.e. mobile push notification) and is not documented very much. So read this section well.

There is another file, `/var/lib/zmeventnotification/push/tokens.txt` that dictates if events are finally sent or not. This pre-dates all the hook stuff and was created really so that zmNinja could receive notifications from the ES.

This file is actually created when zmNinja sets up push notification. Here is how it works:

- When zmNinja runs and you enable push notifications, it asks either Apple or Google for a unique token to receive notifications via their push servers.
- This token is then sent to the ES via websockets. The ES stores this token in the `tokens.txt` file and everytime it restarts, it reloads these tokens so it knows these clients expect notifications over FCM. **So if your zmNinja app cannot connect to the ES for the first time, the token will never be saved and the ES will never be able to send notifications to your zmNinja app**.

However, there are other things the `tokens.txt` file saves. Let's take a look:

Here is a typical tokens.txt entry: (This used to be a cryptic colon separated file, now migrated to JSON starting ES 6.0.1)

```
{"tokens":{"<long token>":
            { "platform":"ios",
              "monlist":"1,2,5,6,7,8,9,10",
              "pushstate":"enabled",
              "intlist":"0,120,120,120,0,120,120,0",
              "appversion":"1.5.001",
              <etc>
            }
          }
}
```

- long token = unique token, we discussed this above
- monlist = list of monitors that will be processed for events for this connection. For example, in the first row, this device will ONLY get notifications for monitors 1,2,5
- intlist = interval in seconds before the next notification is sent. If we look at the first row, it says monitor 1 events will be sent as soon as they occur, however for monitor 2 and 5, notifications will only be sent if the previous notification for that monitor was *at least* 120 seconds before (2 mins). How is this set? You actually set it via zmNinja->Settings->Event Server Settings
- platform the device type (we need this to create a push notification message correctly)
- pushstate = Finally, this tells us if push is enabled or disabled for this device. There are two ways to disable - you can disable push notifications for zmNinja on your device, or you can simply uncheck "use event server" in zmNinja. This is for the latter case. If you uncheck "use event server", we need to be able to tell the ES that even though it has a token on file, it should not send notifications.
- appversion = version of zmNinja (so we know if FCMv1 is supported). For any zmNinja version prior to `1.6.000` this is set to `unknown`.

Important: It is important to note here that if zmNinja is not able to connect to the ES at least for the first time, you will never receive notifications. Check your `tokens.txt` file to make sure you have entries. If you don't that means zmNinja can't reach your ES.

You will also note that `tokens.txt` does not contain any other entries besides android and iOS. zmNinja desktop does not feature here, for example. That is because `tokens.txt` only exists to store FCM registrations. zmNinja desktop only receives notifications when it is running and via websockets, so that connection is established when the desktop app runs. FCM tokens on the other hand need to be remembered, because zmNinja may not be running in your phone and the ES still nees to send out notifications to all tokens (devices) that might have previously registered.

3.2.4: Wait, what on earth is a "Rules file"?

Starting ES 6.0, I've added a `es_rules.json` that gets installed in `/etc/zm/`. It is a json file, that over time will expand in functionality. As of today, it only supports the "mute" action. You can specify "mute" time ranges where the ES will not send out notifications.

Basically, I dislike the format of `tokens.txt`. It was done a long time ago and is cryptic. I should have made it easier to understand and edit. _Eventually_, I'll migrate everything to this JSON file except for token IDs.

Here is an example of the rules file:

```
{
  "notifications": {
      "monitors":{
          "999": {
              "rules": [{
                      "comment": "Be careful with dates, no leading spaces, etc",
                      "time_format":"%I:%M %p",
                      "from":"9:30 pm",
                      "to":"1 am",
                      "daysofweek": "Mon,Tue,Wed",
                      "cause_has":"^(?!.*(person)).*$",
                      "action": "mute"
                  },
                  {
                      "time_format": "%I:%M %p",
                      "from": "3 am",
                      "to": "6 am",
                      "action": "mute",
                      "cause_has": "truck"


                  }
              ]
          },
          "998": {
              "rules": [{
                  "time_format":"%I:%M %p",
                  "from":"5 pm",
                  "to":"7 am",
                  "action":"mute"

              }]
          }

      }
  }
```

}

It says for Monitor ID 999, don't send notifications between 9:30pm to 1am on Mon,Tue,Wed for any alarms that don't have "person" in it's cause assuming you are using object detection. It also says from 3am - 6am for all days of the week, don't send alarms if the alarm cause has "truck" in it.

For Monitor 998, don't send notifications from 5pm to 7am for all days of the week. Note that you need to install `Time::Pice` in Perl.

3.2.4 4: Deciding what to do when a new event ends

Everything above was when an event first starts. The ES also allows similar functions for when an event *ends*. It pretty much follows the flow defined in *3: Deciding what to do when a new event starts* with the following differences:

- The hook, if enabled is defined by `event_end_hook` inside `zmeventnotification.ini`
- The default end script which is usually `/var/lib/zmeventnotification/bin/zm_event_end.sh` doesn't do anything. All the image recognition happens at the event start. Feel free to modify it to do anything you want. As of now, its just a "pass through" that returns a success (`0`) exit code
- Sending notification rules are the same as the start section, except that `event_end_notify_on_hook_success` and `event_end_notify_on_hook_fail` are used for channel rules in `zmeventnotification.ini`
- When the event ends, the ES will check the ZM DB to see if the detection text it wrote during start still exists. It may have been overwritten if ZM detect more motion after the detection. As of today, ZM keeps its notes in memory and doesn't know some other entity has updated the notes and overwrites it.
- At this stage, the fork that was started when the event started exits

3.2.5 User triggers after event_start and event_end

Starting version `5.14` I also support two new triggers called `event_start_hook_notify_userscript` and `event_end_hook_notify_userscript`. If specified, they are invoked so that the user can perform any housekeeping jobs that are necessary. These triggers are useful if you want to use the default object detection scripts *as well* as doing your own things after it.

3.2.6 5: Actually sending the notification

So let's assume that all checks have passed above and we are now about to send the notification. What is actually sent?

- `zmeventnotification.pl` finally sends out the message. The exact protocol depends on the channel:
  - If it is FCM, the message is sent using FCM API
  - If it is MQTT, we use use `MQTT::Simple` (a perl package) to send the message
  - If it is Websockets, we use `Net::WebSocket`, another perl package to send the message
  - If it is a 3rd party push service, then we rely on `api_push_script` in *zmeventnotification.ini*' to send the message.

5.1 Notification Payload

Irrespective of the protocol, the notification message typically consists of:

- Alarm text
- if you are using `fcm` or `push_api`, you can also include an image of the alarm. That picture is typically a URL, specified in `picture_url` inside `zmeventnotification.ini`
- If you are sending over MQTT, there is additional data, including a JSON structure that provides the detection text in an easily parseable structure (`detection` field)
- There are some other fields included as well

5.1.1 Image inside the notification payload

We mentioned above that the image is contained in the `picture_url` attribute. Let's dive into that a bit. The format of the picture url is: `https://pliablepixels.duckdns.org:8889/zm/index.php?view=image&eid=EVENTID&fid=<FID>&width=600`

There are interesting things you can do with the `<FID>` part.

- `fid=BESTMATCH` - this will replace the frameID with whichever frame objects were detected
- `fid=objdetect`

Whatever value is finally used for `<FID>` is what we call the "anchor" frame.

Note: Animations are a new concept and requires ZM 1.35+. Animations can be created around the time of alarm and sent to you as a live notification, so you see moving frames in your push message. You can create animations as MP4 or GIF files (or both). MP4 is more space efficient and animates approximately +-5 seconds around the anchor frame. GIF animation takes more space and animates approximately +-2 seconds around the anchor frame.

- `fid=objdetect`
  - in ZM 1.34 and below this will extract the frame that has objects with borders around them (static image)
  - in ZM 1.35+ if you have opted to create a GIF animation, this will return the GIF animation of the event or the frame with borders around the objects (static image)
- `fid=objdetect_gif`
  - only ZM 1.35+. Returns the GIF animation for the alarmed event if it exists
- `fid=objdetect_mp4`
  - only ZM 1.35+. Returns the MP4 animation for the alarmed event if it exists

3.3 Controlling the Event Server

There is both a static and dynamic way to control the ES.

- You can change parameters in `zmeventnotification.ini`. This will however require you to restart the ES (`sudo zmdc.pl restart zmeventnotification.pl`). You can also change hook related parameters in `objectconfig.ini` and they will automatically take effect for the next detection (because the hook scripts restart with each invocation), if you are using local detections.
- So obviously, there was a need to allow for programmatic change to the ES and dynamically.

That is what the "ES control interface" does. It is a websocket based interface that requires authentication. Once you authenticate, you can change any ES parameter that is in the config. Read more about it: *Category: escontrol messages*.

Just remember:

- admin override via this channel takes precedence over config file
- admin overrides are stored in a different file `/var/lib/zmeventnotification/misc/escontrol_interface.dat` and are encoded. So if you are confused why your config changes to the ini file are not working, and you have enabled this control interface, check for that dat file and remove it to start from scratch.

3.4 How Machine Learning works

There is a dedicated document that describes how hooks work at *Machine Learning Hooks*. Refer to that for details. This section will describe high level principles.

As described earlier, the entry point to all the machine learning goodness starts with `/var/lib/zmeventnotitication/bin/zm_detect.py`. This file reads `/etc/zm/objectconfig.ini` and based on the many settings there goes about doing various forms of detection. There are some important things to remember:

- When the hooks are invoked, ZM has *just started* recording the event. Which means there are only limited frames to analyze. Infact, at times, if you see the detection scripts are not able to download frames, then it is possible they haven't yet been written to disk by ZM. This is a good situation to use the `wait` attribute in `objectconfig.ini` and wait for a few seconds before it tries to get frames.

Gotcha

If you ever wonder why detection did not work when the ES invoked it, but worked just fine when you ran the detection manually, this may be why: during detection the snapshot was different from the final value.

- The detection scripts DO NOT analyze all frames recorded so far. That would take too long (well, not if you have a powerful GPU). It only analyzes two frames at most, depending on your `frame_id` value in `objectconfig.ini`. Those two frames are `snapshot` and `alarm`, assuming you set `frame_id=bestmatch`
- `snapshot` is the frame that has the highest score. It is very possible this frame changes *after* the detection is done, because it is entirely possible that another frame with a higher score is recorded by ZM as the event proceeds.
- There are various steps to detection:
  1. Match all the rules in `objectconfig.ini` (example type(s) of detection for that monitor, etc.)
  2. Do the actual detection
  3. Make sure the detections meet the rules in `objectconfig.ini` (example, it intersects the polygon boundaries, category of detections, etc.)
  4. Of these step 2. can either be done locally or remotely, depending on how you set up `ml_gateway`. Everything else is done locally. See *this FAQ entry* for more details.

CHAPTER 4

Machine Learning Hooks

Note: Before you install machine learnings hooks, please make sure you have installed the Event Notification Server (*Installation of the Event Server (ES)*) and have it working properly

Important: Please don't ask me basic questions like "pip3 command not found - what do I do?" or "cv2 not found, how can I install it?" Hooks require some terminal knowledge and familiarity with troubleshooting. I don't plan to provide support for these hooks. They are for reference only

4.1 Key Features

- Detection: objects, faces
- Recognition: faces
- Platforms:
  - CPU (object, face detection, face recognition),
  - GPU (object, face detection, face recognition),
  - EdgeTPU (object, face detection)
- Machine learning can be installed locally with ZM, or remotely via mlapi

4.2 Limitations

- Only tested with ZM 1.32+. May or may not work with older versions
- Needs Python3 (Python2 is not supported)

4.3 What

Kung-fu machine learning goodness.

This is an example of how you can use the `hook` feature of the notification server to invoke a custom script on the event before it generates an alarm. I currently support object detection and face recognition.

Please don't ask me questions on how to use them. Please read the extensive documentation and ini file configs

4.4 Installation

4.4.1 Automatic install

- You need to have `pip3` installed. On ubuntu, it is `sudo apt install python3-pip`, or see this
- Clone the event server and go to the `hook` directory

```
git clone https://github.com/pliablepixels/zmeventnotification # if you don't already␣
↪have it downloaded

cd zmeventnotification
```

- (OPTIONAL) Edit `hook/zm_event_start.sh` and change:
  - `CONFIG_FILE` to point to the right config file, if you changed paths

```
sudo -H ./install.sh # and follow the prompts
```

I use a library called Shapely for polygon intersection checks. Shapely requires a library called GeOS. If you see errors related to `Failed `CDLL(libgeos_c.so)`` you may manually need to install libgeos like so:

```
sudo apt-get install libgeos-dev
```

Google TPU: If you are planning on using Google EdgeTPU support, you'll have to invoke the script using:

```
sudo -H INSTALL_CORAL_EDGETPU=yes ./install.sh # and follow the prompts
```

EdgeTPU models/underlying libraries are not downloaded automatically.

For EdgeTPU, the expectation is that you have followed all the instructions at the coral site first. Specifically, you need to make sure you have:

1. Installed the right libedgetpu library (max or std)
2. Installed the right tensorflow-lite library
3. Installed pycoral APIs as per the instructions.

If you don't, things will break. Further, you also need to make sure your web user has access to the coral device.

On my ubuntu system, I needed to do:

```
sudo usermod -a -G plugdev www-data
```

Very important: You need to reboot after this, otherwise, you may notice that the TPU code crashes when you run the ES in daemon mode (may work fine in manual mode)

.._install-specific-models:

Starting version 5.13.3, you can *optionally* choose to only install specific models by passing them as variables to the install script. The variables are labelled as `INSTALL_<model>` with possible values of `yes` (default) or `no`. `<model>` is the specific model.

So for example:

```
sudo INSTALL_YOLOV3=no INSTALL_YOLOV4=yes ./install.sh
```

Will only install the `YOLOv4 (full)` model but will skip the `YOLOV3` model.

Another example:

```
sudo -H INSTALL_CORAL_EDGETPU=yes ./install.sh --install-es --no-install-config --
↪install-hook
```

Will install the ES and hooks, but no configs and will add the coral libraries.

Note:: If you plan on using object detection, starting v5.0.0 of the ES, the setup script no longer installs opencv for you. This is because you may want to install your own version with GPU accelaration or other options. There are two options to install OpenCV:

- You install a pip package. Very easy, but you don't get GPU support
- You compile from source. Takes longer, but you get all the right modules as well as GPU support. Instructions are simple, if you follow them well.

Important: However you choose to install openCV, you need a minimum version of *4.1.1*. Using a version below that will very likely not work.

Installing OpenCV: Using the pip package (Easy, but not recommended if you plan to use OpenCV ML - example Yolo)

```
# Note this does NOT enable GPU support
# It also seems to miss modules like bgsem etc

sudo -H pip3 install opencv-contrib-python

# NOTE: Do NOT install both opencv-contrib-python and opencv packages via pip. The␣
↪contrib package includes opencv+extras
```

Installing OpenCV: from source (Recommended if you plan to use OpenCV ML - example Yolo)

General installation instructions are available at the official openCV site. However, see below, if you are looking for GPU support:

If you want to install a version with GPU support, I'd recommend you install OpenCV 4.2.x because it supports a CUDA backend for deep learning. Adrian's blog has a good howto on compiling OpenCV 4.2.x from scratch.

I would strongly recommend you build from source, if you are able to. Pre built packages are not official from OpenCV and often seem to break/seg fault on different configurations.

Make sure OpenCV works

Important: After you install opencv, make sure it works. Start python3 and inside the interpreter, do a `import cv2`. If it seg faults, you have a problem with the package you installed. Like I said, I've never had issues after building from source.

Note that if you get an error saying `cv2 not found` that means you did not install it in a place python3 can find it (you might have installed it for python2 by mistake)

Note 3: if you want to add "face recognition" you also need to do the following: (Note that if you are using Google Coral, you can do face detection [not recognition] by using the coral libraries as well. You can skip this section if you want to use TPU based face detection)

```
sudo apt-get install libopenblas-dev liblapack-dev libblas-dev  # not mandatory, but␣
↪gives a good speed boost!
sudo -H pip3 install face_recognition # mandatory
```

Takes a while and installs a gob of stuff, which is why I did not add it automatically, especially if you don't need face recognition.

Note, if you installed `face_recognition` earlier without blas, do this:

```
sudo -H pip3 uninstall dlib
sudo -H pip3 uninstall face-recognition
sudo apt-get install libopenblas-dev liblapack-dev libblas-dev # this is the␣
↪important part
sudo -H pip3 install dlib --verbose --no-cache-dir # make sure it finds openblas
sudo -H pip3 install face_recognition
```

4.4.2 Option 2: Manual install

If automatic install fails for you, or you like to be in control, take a look at what `install.sh` does. I used to maintain explict instructions on manual install, but its painful to keep this section in sync with `install.sh`

4.5 Post install steps

- Make sure you edit your installed `objectconfig.ini` to the right settings. You MUST change the `[general]` section for your own portal.
- Make sure the `CONFIG_FILE` variable in `zm_event_start.sh` is correct

4.6 Test operation

```
sudo -u www-data /var/lib/zmeventnotification/bin/zm_event_start.sh <eid> <mid> #␣
↪replace www-data with apache if needed
```

Replace with your own `eid` (Example 123456). This will be an event id for an event that you want to test. Typically, open up the ZM console, look for an event you want to run analysis on and select the ID of the event. That is the `eid`. The `mid` is the monitor ID for the event. This is optional. If you specify it, any monitor specific settings (such as zones, hook customizations, etc. in `objectconfig/mlapiconfig.ini` will be used).

The above command will try and run detection.

If it doesn't work, go back and figure out where you have a problem

4.7 Upgrading

To upgrade at a later stage, see *How do I safely upgrade zmeventnotification to new versions?*.

4.8 Sidebar: Local vs. Remote Machine Learning

Starting v5.0, you can now choose to run the machine learning code on a separate server. This can free up your local ZM server resources if you have memory/CPU constraints. See *this FAQ entry*.

4.9 Which models should I use?

- Starting 5.16, Google Coral Edge TPU is supported. See install instructions above.
- Starting 5.15.6, you have the option of using YoloV3 or YoloV4. V3 is the original one while V4 is an optimized version by Alexey. See here. V4 is faster, and is supposed to be more accurate but YMMV. Note that you need a version GREATER than 4.3 of OpenCV to use YoloV4
- If you are constrained in memory, use tinyyolo
- Each model can further be customized for accuracy vs speed by modifying parameters in their respective `.cfg` files. Start here and then browse the issues list.
- For face recognition, use `face_model=cnn` for more accuracy and `face_model=hog` for better speed

4.10 Understanding detection configuration

Starting v6.1.0, you can chain arbitrary detection types (object, face, alpr) and multiple models within them. In older versions, you were only allowed one model type per detection type. Obviously, this has required structural changes to `objectconfig.ini`

This section will describe the key constructs around two important structures:

- ml_sequence (specifies sequence of ML detection steps)
- stream_sequence (specifies frame detection preferences)

4.10.1 6.1.0+ vs previous versions

When you update to 6.1.0, you may be confused with objectconfig. Specifically, which attributes should you use and which ones are ignored? It's pretty simple, actually.

Note: use_sequence=no is no longer supported. Please make sure it is set to yes, and follow instructions on how to set up ml_sequence and stream_sequence correctly

---

- When `use_sequence` is set to `yes` (default is yes), `ml_options` and `stream_sequence` structures override anything in the `[object]`, `[face]` and `[alpr]` sections Specifically, the following values are ignored in objectconfig.ini in favor of values inside the sequence structure:
  - frame_id, resize, delete_after_analyze, the full [object], [alpr], [face] sections
  - any overrides related to object/face/alpr inside the [monitor] sections

- However, that being said, if you take a look at `objectconfig.ini`, the sample file implements parameter substitution inside the structures, effectively importing the values right back in. Just know that what you specify in these sequence structures overrides the above attributes. If you want to reuse them, you need to put them in as parameter substitutions like the same ini file has done
- If you are using the new `use_sequence=yes` please don't use old keywords as variables. They will likely fail.

Example, this will **NOT WORK**:

```
use_sequence=yes

[monitor-3]
detect_sequence='object,face,alpr'

[monitor-4]
detect_sequence='object'

[ml]
ml_sequence= {
   <...>
   general: {
      'model_sequence': '{{detection_sequence}}'
   },
   <...>

}
```

What you need to do is use a different variable name (as `detect_sequence` is a reserved keyword which is used if `use_sequence=no`)

But this **WILL WORK**:

```
use_sequence=yes

[monitor-3]
my_sequence=object,face,alpr

[monitor-4]
my_sequence=object

[ml]
ml_sequence= {
   <...>
   general: {
      'model_sequence': '{{my_sequence}}'
   },
   <...>

}
```

- When `use_sequence` is set to `no`, zm_detect internally maps your old parameters to the new structures

Internally, both options are mapped to `ml_sequence`, but the difference is in the parameters that are processed. Specifically, before ES 6.1.0 came out, we had specific objectconfig fields that were used for various ML parameters that were processed. These were primarily single, well known variable names because we only had one model type running per type of detection.

More details: What happens when you go with use_sequence=no?

NOTE: Please use `use_sequence=yes`. In the past, I've allowed a mode where legacy settings were converted to `stream_sequence` auto-magically when I changed formats. This caused a lot of issues and was a nightmare to maintain. So please migrate to the proper format. If you've been using `use_sequence=no` it will likely break things, but please read the help and convert properly. `use_sequence=no` is no longer maintained.

In the old way, the following 'global' variables (which could be overriden on a per monitor basis) defined how ML would work:

- `xxx_max_processes` and `xxx_max_lock_wait` that defined semaphore locks for each model (to control parallel memory consumption)
- All the `object_xxx` variables that define the model file, name file, and a host of other parameters that are specific to object detection
- All the `face_xxx` variables, `known_images_path`, *unknown_images_path'*, `save_unknown_*` attributes that define the model file, name file, and a host of other parameters that are specific to face detection
- All the `alpr_xxx` variables that define the model file, name file, and a host of other parameters that are specific to alpr detection

When you make `use_sequence=no` in your config, I have a function called `convert_config_to_ml_sequence()` (see here) that basically picks up those variable, maps it to an `ml_sequence` structure with exactly one model per sequence (like it was before). It picks up the sequence of models from `detection_sequence` which was the old way.

Further, in this mode, a `sream_sequence` structure is internally created that picks up values from the old attributes, `detection_mode`, `frame_id`, `bestmatch_order`, `resize`

Therefore, the concept here was, if you choose not to use the new detection sequence, you _should_ be able to continue using your old variables and the code will internally map.

More details: What happens when you go with use_sequence=yes?

When you go with `yes`, `zm_detect.py` does NOT try to map any of the old variables. Instead, it directly loads whatever is defined inside `ml_sequence` and `stream_sequence`. However, you will notice that the default `ml_sequence` and `stream_sequence` are pre-filled with template variables.

For example:

```
ml_sequence= {
        <snip>
             'object': {
                      'general':{
                              'pattern':'{{object_detection_pattern}}',
                              'same_model_sequence_strategy': 'first' # also 'most',
↪'most_unique's
                      },
                      'sequence': [{
                              #First run on TPU with higher confidence
                              'object_weights':'{{tpu_object_weights}}',
                              'object_labels': '{{tpu_object_labels}}',
                              'object_min_confidence': {{tpu_min_confidence}},
         <snip>

                      },
                      {
                              # YoloV4 on GPU if TPU fails (because sequence strategy␣
↪is 'first')
```

(continues on next page)

(continued from previous page)

```
                                'object_config':'{{yolo4_object_config}}',
                                'object_weights':'{{yolo4_object_weights}}',
                                'object_labels': '{{yolo4_object_labels}}',
```

Note the variables inside `{{}}`. They will be replaced when the structure is formed. And you'll note some are old style variables (example `object_detection_pattern`) along with may new ones. So here is the thing: You can use any variable names you want in the new style. Obviously, we can't use `object_weights` as the only variable if we plan to chain different models. They'll have different values.

So remember, the config presented here is a **SAMPLE**. You are **expected to change them**.

So in the new way, if you want to change `ml_sequence` or `stream_sequence` on a per monitor basis, you have 2 choices: - Put variables inside the main `*_sequence` options and simply redefine those variables on a per monitor basis - Or, redo the entire structure on a per monitor basis. I like Option 1.

4.10.2 Understanding ml_sequence

The `ml_sequence` structure lies in the `[ml]` section of `objectconfig.ini` (or `mlapiconfig.ini` if using mlapi). At a high level, this is how it is structured (not all attributes have been described):

```
ml_sequence = {
   'general': {
      'model_sequence':'<comma separated detection_type>'
   },
   '<detection_type>': {
      'general': {
         'pattern': '<pattern>',
         'same_model_sequence_strategy':'<strategy>'
      },
      'sequence:[{
         <series of configurations>
      },
      {
         <series of configurations>
      }]
   }
}
```

Explanation:

- The `general` section at the top level specify characterstics that apply to all elements inside the structure.
  - `model_sequence` dictates the detection types (comma separated). Example `object,face,alpr` will first run object detection, then face, then alpr
- Now for each detection type in `model_sequence`, you can specify the type of models you want to leading along with other related paramters.

  Note: If you are using mlapi, there are certain parameters that get overriden by `objectconfig.ini` See *Exceptions when using mlapi*

Leveraging same_model_sequence_strategy and frame_strategy effectively

When we allow model chaining, the question we need to answer is 'How deep do we want to go to get what we want?' That is what these attributes offer.

`same_model_sequence_strategy` is part `ml_sequence` with the following possible values:

- **`first` - When detecting objects, if there are multiple fallbacks, break out the moment we get a match** using any object detection library (Default)
- `most` - run through all libraries, select one that has most object matches
- `most_unique` - run through all libraries, select one that has most unique object matches

`frame_strategy` is part of `stream_sequence` with the following possible values:

- 'most_models': Match the frame that has matched most models (does not include same model alternatives) (Default)
- 'first': Stop at first match
- 'most': Match the frame that has the highest number of detected objects
- 'most_unique' Match the frame that has the highest number of unique detected objects

A proper example:

Take a look at this article for a walkthrough.

All options:

`ml_sequence` supports various other attributes. Please see the pyzm API documentation that describes all options. The `options` parameter is what you are looking for.

4.10.3 Understanding stream_sequence

The `stream_sequence` structure lies in the `[ml]` section of `objectconfig.ini`. At a high level, this is how it is structured (not all attributes have been described):

```
stream_sequence = {
     'frame_set': '<series of frame ids>',
     'frame_strategy': 'most_models',
     'contig_frames_before_error': 5,
     'max_attempts': 3,
     'sleep_between_attempts': 4,
               'resize':800

 }
```

Explanation:

- `frame_set` defines the set of frames it should use for analysis (comma separated)
- `frame_strategy` defines what it should do when a match has been found
- `contig_frames_before_error`: How many contiguous errors should occur before giving up on the series of frames
- `max_attempts`: How many times to try each frame (before counting it as an error in the `contig_frames_before_error` count)
- `sleep_between_attempts`: When an error is encountered, how many seconds to wait for retrying
- `resize`: what size to resize frames too (useful if you want to speed things up and/or are running out of memory)

A proper example:

Take a look at this article for a walkthrough.

All options:

`stream_sequence` supports various other attributes. Please see the pyzm API documentation that describes all options. The `options` parameter is what you are looking for.

4.10.4 How ml_sequence and stream_sequence work together

Like this:

```
for each frame in stream sequence:
   perform stream_sequence actions on each frame
   for each model_sequence in ml_options:
   if detected, use frame_strategy (in stream_sequence) to decide if we should try␣
↪other model sequences
      perform general actions:
         for each model_configuration in ml_options.sequence:
            detect()
            if detected, use same_model_sequence_strategy to decide if we should try␣
↪other model configurations
```

4.10.5 Exceptions when using mlapi

If you are using the remote mlapi server, then most of these settings migrate to `mlapiconfig.ini` Specifically, when `zm_detect.py` sees `ml_gateway` in its `[remote]` section, it passes on the detection work to mlapi.

NOTE: Please use `use_sequence=yes`. In the past, I've allowed a mode where legacy settings were converted to `stream_sequence` auto-magically when I changed formats. This caused a lot of issues and was a nightmare to maintain. So please migrate to the proper format. If you've been using `use_sequence=no` it will likely break things, but please read the help and convert properly. `use_sequence=no` is no longer maintained.

Here are a list of parameters that still need to be in `objectconfig.ini` when using mlapi. (A simple rule to remember is zm_detect.py uses `objectconfig.ini` while mlapi uses `mlapiconfig.ini`)

- `ml_gateway` (obviously, as the *ES* calls *zm_detect*, and zm_detect calls mlapi if this parameter is present in *objectconfig.ini*)
- `ml_fallback_local` (if *mlapi* fails, or is not running, *zm_detect* will switch to local inferencing, so this needs to be in *objectconfig.ini*)
- `show_percent` (*zm_detect* is the one that actually creates the text you see in your object detection (*detected:[s] person:90%*))
- `write_image_to_zm` (zm_detect is the one that actually puts *objdetect.jpg* in the ZM events folder - *mlapi* can't because it can be remote)
- `write_debug_image` (*zm_detect* is the one that creates a debug image to inspect)
- `poly_thickness` (because *zm_detect* is the one that actually writes the image/debug image as described above, makes sense that *poly_thickness* needs to be here)
- `image_path` (related to above - to know where to write the image)
- `create_animation` (*zm_detect* has the code to put the animation of mp4/gif together)
- `animation_types` (same as above)
- `show_models` (if you want to show model names along with text)

These need to be present in both mlapiconfig.ini and objectconfig.ini

- `secrets`

- base_data_path
- api_portal
- portal
- user
- password

So when using mlapi, migrate configurations that you typically specify in `objectconfig.ini` to `mlapiconfig.ini`. This includes:

- Monitor specific sections
- ml_sequence and stream_sequence
- In general, if you see detection with mlapi missing something that worked when using objectconfig.ini, make sure you have not missed anything specific in mlapiconfig.ini with respect to related parameters

Also note that if you are using ml_fallback, repeat the settings in both configs.

Here is a part of my config, for example:

```
import_zm_zones=yes
## Monitor specific settings
[monitor-3]
# doorbell
model_sequence=object,face
object_detection_pattern=(person|monitor_doorbell)
valid_face_area=184,235 1475,307 1523,1940 146,1940

[monitor-7]
# Driveway
model_sequence=object,alpr
object_detection_pattern=(person|car|motorbike|bus|truck|boat)

[monitor-2]
# Front lawn
model_sequence=object
object_detection_pattern=(person)

[monitor-4]
#deck
object_detection_pattern=(person|monitor_deck)
stream_sequence = {
      'frame_strategy': 'most_models',
      'frame_set': 'alarm',
      'contig_frames_before_error': 5,
      'max_attempts': 3,
      'sleep_between_attempts': 4,
      'resize':800

   }
```

4.11 About specific detection types

4.11.1 License plate recognition

Three ALPR options are provided:

- Plate Recognizer . It uses a deep learning model that does a far better job than OpenALPR (based on my tests). The class is abstracted, obviously, so in future I may add local models. For now, you will have to get a license key from them (they have a free tier that allows 2500 lookups per month)
- OpenALPR . While OpenALPR's detection is not as good as Plate Recognizer, when it does detect, it provides a lot more information (like car make/model/year etc.)
- OpenALPR command line. This is a basic version of OpenALPR that can be self compiled and executed locally. It is far inferior to the cloud services and does NOT use any form of deep learning. However, it is free, and if you have a camera that has a good view of plates, it will work.

`alpr_service` defined the service to be used.

4.11.2 Face Dection & Recognition

When it comes to faces, there are two aspects (that many often confuse):

- Detecting a Face
- Recognizing a Face

Face Detection

If you only want "face detection", you can use either dlib/face_recognition or Google's TPU. Both are supported. Take a look at `objectconfig.ini` for how to set them up.

Face Detection + Face Recognition

Face Recognition uses this library. Before you try and use face recognition, please make sure you did a `sudo -H pip3 install face_recognition` The reason this is not automatically done during setup is that it installs a lot of dependencies that takes time (including dlib) and not everyone wants it.

Face recognition limitations

Don't expect magic with overhead cameras. This library requires a reasonable face orientation (works for front facing, or somewhat side facing poses) and does not work for full profiles or completely overhead faces. Take a look at the accuracy wiki of this library to know more about its limitations. Also note that I found *cnn* mode is much more accurage than *hog* mode. However, *cnn* comes with a speed and memory tradeoff.

Using the right face recognition modes

- Face recognition uses dlib. Note that in `objectconfig.ini` you have two options of face detection/recognition. Dlib has two modes of operation (controlled by `face_model`). Face recognition works in two steps: - A: Detect a face - B: Recognize a face

`face_model` affects step A. If you use `cnn` as a value, it will use a DNN to detect a face. If you use `hog` as a value, it will use a much faster method to detect a face. `cnn` is *much* more accurate in finding faces than `hog` but much slower. In my experience, `hog` works ok for front faces while `cnn` detects profiles/etc as well.

Step B kicks in only after step A succeeds (i.e. a face has been detected). The algorithm used there is common irrespective of whether you found a face via `hog` or `cnn`.

Configuring face recognition directories

- Make sure you have images of people you want to recognize in `/var/lib/zmeventnotification/known_faces`
- You can have multiple faces per person
- Typical configuration:

```
known_faces/
  +----------bruce_lee/
              +------1.jpg
              +------2.jpg
  +----------david_gilmour/
           +------1.jpg
           +------img2.jpg
           +------3.jpg
  +----------ramanujan/
           +------face1.jpg
           +------face2.jpg
```

In this example, you have 3 names, each with different images.

- It is recommended that you now train the images by doing:

```
sudo -u www-data /var/lib/zmeventnotification/bin/zm_train_faces.py
```

If you find yourself running out of memory while training, use the size argument like so:

```
sudo -u www-data /var/lib/zmeventnotification/bin/zm_train_faces.py --size 800
```

- Note that you do not necessarily have to train it first but I highly recommend it. When detection runs, it will look for the trained file and if missing, will auto-create it. However, detection may also load yolo and if you have limited GPU resources, you may run out of memory when training.
- When face recognition is triggered, it will load each of these files and if there are faces in them, will load them and compare them to the alarmed image

known faces images

- Make sure the face is recognizable
- crop it to around 800 pixels width (doesn't seem to need bigger images, but experiment. Larger the image, the larger the memory requirements)
- crop around the face - not a tight crop, but no need to add a full body. A typical "passport" photo crop, maybe with a bit more of shoulder is ideal.

4.12 Troubleshooting

- In general, I expect you to debug properly. Please don't ask me basic questions without investigating logs yourself
- Always run `zm_event_start.sh` or `zm_detect.py` in manual mode first to make sure it works
- Make sure you've set up debug logging as described in *Logging*
- One of the big reasons why object detection fails is because the hook is not able to download the image to check. This may be because your ZM version is old or other errors. Some common issues:
  - Make sure your `objectconfig.ini` section for `[general]` are correct (portal, user,admin)
  - For object detection to work, the hooks expect to download images of events using `https://yourportal/zm/?view=image&eid=<eid>&fid=snapshot` and possibly `https://yourportal/zm/?view=image&eid=<eid>&fid=alarm`
  - Open up a browser, log into ZM. Open a new tab and type in `https://yourportal/zm/?view=image&eid=<eid>&fid=snapshot` in your browser. Replace `eid` with an actual event id. Do you see an image? If not, you'll have to fix/update ZM. Please don't ask me how. Please post in the ZM forums
  - Open up a browser, log into ZM. Open a new tab and type in `https://yourportal/zm/?view=image&eid=<eid>&fid=alarm` in your browser. Replace `eid` with an actual event id. Do you see an image? If not, you'll have to fix/update ZM. Please don't ask me how. Please post in the ZM forums

4.12.1 Debugging and reporting problems

Reporting Problems:

1. Make sure you have debug logging enabled as described in *Logging*
2. Don't just post the final error message. Please post _full_ debug logs.

If you have problems with hooks, there are two areas of failure:

- The ES is unable to unable to invoke hooks properly (missing files/etc)
  - This will be reported in ES logs. See *this section*
- Hooks don't work
  - This is covered in this section
- The wrapper script (typically `/var/lib/zmeventnotification/bin/zm_event_start.sh` is not able to run `zm_detect.py`)
  - This won't be covered in either logs (I need to add logging for this...)

To understand what is going wrong with hooks, I like to do things the following way:

- Stop the ES if it is running (`sudo zmdc.pl stop zmeventnotification.pl`) so that we don't mix up what we are debugging with any new events that the ES may generate
- Next, I take a look at `/var/log/zm/zmeventnotification.log` for the event that invoked a hook. Let's take this as an example:

```
01/06/2021 07:20:31.936130 zmeventnotification[28118].DBG [main:977] [|---->␣
↪FORK:DeckCamera (6), eid:182253 Invoking hook on event start:'/var/lib/
↪zmeventnotification/bin/zm_event_start.sh' 182253 6 "DeckCamera" " stairs" "/var/
↪cache/zoneminder/events/6/2021-01-06/182253"]
```

(continues on next page)

(continued from previous page)

Let's assume the above is what I want to debug, so then I run zm_detect manually like so:

```
sudo -u www-data /var/lib/zmeventnotification/bin/zm_detect.py --config /etc/zm/
↪objectconfig.ini --debug --eventid 182253  --monitorid 6 --eventpath=/tmp
```

Note that instead of `/var/cache/zoneminder/events/6/2021-01-06/182253` as the event path, I just use `/tmp` as it is easier for me. Feel free to use the actual event path (that is where objdetect.jpg/json are stored if an object is found).

This will print debug logs on the terminal.

4.13 Performance comparison

- CPU: Intel(R) Xeon(R) CPU E5-1660 v3 @ 3.00GHz 8 cores, with 32GB RAM
- GPU: GeForce 1050Ti
- TPU: Google Coral USB stick, running on USB 3.0 in 'standard' mode
- Environment: I am running using mlapi, so you will see load time only once across multiple runs
- Image size: 800px

The TPU is running in standard mode, not max. Also note that these figures use pycoral, which is a python wrapper around the TPU C++ libraries. You should also look at Google's Coral coral benchmark site for better numbers. Note that their performance figures are specific to their C++ code. Python will have additional overheads (as noted on their site) Finally, if you are facing data transfter/delegate loading issues considering buying a good quality USB 3.1/10Gbps rated cable. I faced intermittent issues with delegate load issues (not always) which seems to have gone away after I ditched Google's cable with a good quality one (I bought this one)

```
pp@homeserver:/var/lib/zmeventnotification/mlapi$ tail -F /var/log/zm/zm_mlapi.log |␣
↪grep "perf:"


First Run (Model load included):

01/25/21 14:45:19 zm_mlapi[953841] DBG1 detect_sequence.py:398 [perf: Starting for␣
↪frame:snapshot]
01/25/21 14:45:22 zm_mlapi[953841] DBG1 coral_edgetpu.py:93 [perf: processor:tpu TPU␣
↪initialization (loading /var/lib/zmeventnotification/models/coral_edgetpu/ssdlite_
↪mobiledet_coco_qat_postprocess_edgetpu.tflite from disk) took: 3086.99 ms]
01/25/21 14:45:22 zm_mlapi[953841] DBG1 coral_edgetpu.py:179 [perf: processor:tpu␣
↪Coral TPU detection took: 39.30 ms]
01/25/21 14:45:22 zm_mlapi[953841] DBG1 yolo.py:88 [perf: processor:gpu Yolo␣
↪initialization (loading /var/lib/zmeventnotification/models/yolov4/yolov4.weights␣
↪model from disk) took: 182.36 ms]
01/25/21 14:45:23 zm_mlapi[953841] DBG1 yolo.py:169 [perf: processor:gpu Yolo␣
↪detection took: 1249.93 ms]
01/25/21 14:45:23 zm_mlapi[953841] DBG2 yolo.py:204 [perf: processor:gpu Yolo NMS␣
↪filtering took: 0.68 ms]
01/25/21 14:45:26 zm_mlapi[953841] DBG1 face.py:40 [perf: processor:gpu Face␣
↪Recognition library load time took: 0.00 ms ]
01/25/21 14:45:30 zm_mlapi[953841] DBG1 face.py:201 [perf: processor:gpu Finding␣
↪faces took 4418.08 ms]
```

(continues on next page)

(continued from previous page)

```
01/25/21 14:45:30 zm_mlapi[953841] DBG1 face.py:213 [perf: processor:gpu Computing␣
↪face recognition distances took 80.49 ms]
01/25/21 14:45:30 zm_mlapi[953841] DBG1 face.py:245 [perf: processor:gpu Matching␣
↪recognized faces to known faces took 3.13 ms]
01/25/21 14:45:30 zm_mlapi[953841] DBG1 detect_sequence.py:398 [perf: Starting for␣
↪frame:alarm]

Second Run:

01/25/21 14:45:35 zm_mlapi[953841] DBG1 detect_sequence.py:398 [perf: Starting for␣
↪frame:snapshot]
01/25/21 14:45:35 zm_mlapi[953841] DBG1 coral_edgetpu.py:179 [perf: processor:tpu␣
↪Coral TPU detection took: 24.66 ms]
01/25/21 14:45:35 zm_mlapi[953841] DBG1 yolo.py:169 [perf: processor:gpu Yolo␣
↪detection took: 58.06 ms]
01/25/21 14:45:35 zm_mlapi[953841] DBG2 yolo.py:204 [perf: processor:gpu Yolo NMS␣
↪filtering took: 1.35 ms]
01/25/21 14:45:35 zm_mlapi[953841] DBG1 face.py:201 [perf: processor:gpu Finding␣
↪faces took 290.92 ms]
01/25/21 14:45:35 zm_mlapi[953841] DBG1 face.py:213 [perf: processor:gpu Computing␣
↪face recognition distances took 14.51 ms]
01/25/21 14:45:35 zm_mlapi[953841] DBG1 face.py:245 [perf: processor:gpu Matching␣
↪recognized faces to known faces took 2.23 ms]
```

4.13.1 Manually testing if detection is working well

You can manually invoke the detection module to check if it works ok:

```
sudo -u www-data /var/lib/zmeventnotification/bin/zm_detect.py --config /etc/zm/
↪objectconfig.ini  --eventid <eid> --monitorid <mid> --debug
```

The `--monitorid <mid>` is optional and is the monitor ID. If you do specify it, it will pick up the right mask to apply (if it is in your config)

STEP 1: Make sure the scripts(s) work

- Run the python script manually to see if it works (refer to sections above on how to run them manually)
- `./zm_event_start.sh <eid> <mid>` –> make sure it downloads a proper image for that eid. Make sure it correctly invokes detect.py If not, fix it. (`<mid>` is optional and is used to apply a crop mask if specified)
- Make sure the `image_path` you've chosen in the config file is WRITABLE by www-data (or apache) before you move to step 2

STEP 2: run zmeventnotification in MANUAL mode

- `sudo zmdc.pl stop zmeventnotification.pl`
- change console_logs to yes in `zmeventnotification.ini`
- `sudo -u www-data ./zmeventnotification.pl --config ./zmeventnotification.ini`
- Force an alarm, look at logs

STEP 3: integrate with the actual daemon - You should know how to do this already

4.14 Questions

See *Machine Learning Hooks FAQ*

CHAPTER 5

Configuration Guide

There are two parts to the configuration of this system:

- The Event Notification Server configuration - typically `/etc/zm/zmeventnotification.ini`
- The Machine Learning Hooks configuration - typically `/etc/zm/objdetect.ini` and/or `/var/lib/zmeventnofication/mlapi/mlapiconfig.ini`

The ES comes with a sample ES config file which you should customize as fit. The sample config file is well annotated, so you really should read the comments to get an erstanding of what each parameter does. Similary, the ES also comes with a sample objdetect.ini file which you should read as well if you are using hooks. If you are using mlapi, read its config.

5.1 Secret Tokens

All the config files have a notion of secrets. Basically, it is a mechanism to separate out your personal passwords and tokens into a different file from your config file. This allows you to easily share your config files without inadvertently sharing your secrets.

Basically, this is how it works:

You add an attribute called `secrets` to the `[general]` section of either/both config files. This points to some filename you have created with tokens. Then you can just use the token name in the config file.

For example, let's suppose we add this to `/etc/zm/objectconfig.ini`:

```
[general]
# This is an optional file
# If specified, you can specify tokens with secret values in that file
# and onlt refer to the tokens in your main config file
secrets=/etc/zm/secrets.ini

portal=!ZM_PORTAL
user=!ZM_USER
password=!ZM_PASSWORD
```

And `/etc/zm/secrets.ini` contains:

```
# your secrets file
[secrets]
ZMES_PICTURE_URL=https://mysecretportal/zm/index.php?view=image&eid=EVENTID&
↪fid=objdetect&width=600
#ZMES_PICTURE_URL=https://mysecretportal/zm/index.php?view=image&eid=EVENTID&
↪fid=snapshot&width=600
ZM_USER=myusername
ZM_PASSWORD=mypassword
ZM_PORTAL=https://mysecretportal/zm
```

Then, while parsing the config file every time a key value is found that starts with `!` that means it's a secret token and the corresponding value from the secrets file will be substituted.

The same concept applies to `/etc/zm/zmeventnotification.ini` and `/var/lib/zmeventnofication/mlapi/mlapiconfig.ini`

Obviously this means you can no longer have a password beginning with an exclamation mark directly in the config. It will be treated as a secret token. To work around this, create a password token in your secrets file and put the real password there.

CHAPTER 6

Event Notification Server FAQ

6.1 Machine Learning! Mmm..Machine Learning!

Easy. You will first have to read this document to correctly install this server along with zoneminder. Once it works well, you can explore how to enable Machine Learning based object detection that can be used along with ZoneMinder alarms. If you already have this server figured out, you can skip directly to the machine learning part (*Machine Learning Hooks*)

If you have questions on the machine learning part, see *Machine Learning Hooks FAQ*

6.2 What is it?

A WSS (Secure Web Sockets) and/or MQTT based event notification server that broadcasts new events to any authenticated listeners. (As of 0.6, it also includes a non secure websocket option, if that's how you want to run it)

6.3 Why do we need it?

- The only way ZoneMinder sends out event notifications via event filters - this is too slow
- People developing extensions to work with ZoneMinder for Home Automation needs will benefit from a clean interface
- Receivers don't poll. They keep a web socket open and when there are events, they get a notification
- Supports WebSockets, MQTT and Apple/Android push notification transports
- Offers an authentication layer
- Allows you to integrate custom hooks that can decide if an alarm needs to be sent out or not (an example of how this can be used for person detection is provided)

6.4 Is this officially developed by ZM developers?

No. I developed it for zmNinja, but you can use it with your own consumer.

6.5 How can I use this with Node-Red or Home Assistant?

As of version 1.1, the event server also supports MQTT (Contributed by @vajonam). zmeventnotification server can be configured to broadcast on a topic called `/zoneminder/<monitor-id>` which can then be consumed by Home Assistant or Node-Red.

To enable this, set `enable = 1` under the `[mqtt]` section and specify the `server` to broadcast to.

You will also need to install the following module for this work

```
perl -MCPAN -e "install Net::MQTT::Simple"
```

The MQTT::Simple module is known to work only with Mosquitto as of 10 Jun 2019. It does not work correctly with the RabbitMQ MQTT plugin. The easiest workaround if you have an unsupported MQTT system is to install Mosquitto on the Zoneminder system itself and bridge that to RabbitMQ. You can bind Mosquitto to 127.0.0.1 and disable authentication to keep it simple. The eventserver.pl is then configured to send events to the local Mosquitto. This is an example known working bridge set up (on Ubuntu, for example, this is put into /etc/mosquitto/conf.d/local.conf):

```
bind_address 127.0.0.1
allow_anonymous true
connection bridge-zm2things
address 10.10.1.20:1883
bridge_protocol_version mqttv311
remote_clientid bridge-zm2things
remote_username zm
remote_password my_mqtt_zm_password
try_private false
topic # out 0
```

Set the address, remote_username and remote_password for Mosquitto to use on the RabbitMQ. Note that this is a one way bridge, so there is only a topic # out 0. try_private false is needed to avoid a similar error to using MQTT::Simple.

6.6 Disabling security

While I don't recommend either, several users seem to be interested in the following

- To run the eventserver on Websockets and not Secure Websockets, use `enable = 0` in the `[ssl]` section of the configuration file.
- To disable ZM Auth checking (be careful, anyone can get all your data INCLUDING passwords for ZoneMinder monitors if you open it up to the Internet) use `enable = 0` in the `[auth]` section of the configuration file.

6.7 How do I safely upgrade zmeventnotification to new versions?

6.7.1 STEP 1: get the latest code

Download the latest version & change dir to it:

```
git clone https://github.com/pliablepixels/zmeventnotification.git
cd zmeventnotification/
```

6.7.2 STEP 2: stop the current ES

```
sudo zmdc.pl stop zmeventnotification.pl
```

6.7.3 STEP 3: Make a backup of your config files

Before you execute the next step you may want to create a backup of your existing `zmeventnotification.ini` and `objectconfig.ini` config files. The script will prompt you to overwrite. If you say 'Y' then your old configs will be overwritten. Note that old configs are backed up using suffixes like `~1`, `~2` etc. but it is always good to backup on your own.

6.7.4 STEP 4: Execute the install script

NOTE : By default `install.sh` moves the ES script to `/usr/bin`. If your ZM install is elsewhere, like `/usr/local/bin` please modify the `TARGET_BIN` variable in `install.sh` before executing it.

```
sudo -H ./install.sh
```

Follow prompts. Note that just copying the ES perl file to `/usr/bin` is not sufficient. You also have to install the updated machine learning hook files if you are using them. That is why `install.sh` is better. If you are updating, make sure not to overwrite your config files (but please read breaking changes to see if any config files have changed). Note that the install script makes a backup of your old config files using `~n` suffixes where `n` is the backup number. However, never hurts to make your own backup first.

Note that you can also automate updates like so:

```
sudo -H ./install.sh --install-hook --install-es --no-install-config --no-interactive
```

The above will install/update the hooks, install/update the ES server but will not overwrite your existing config files. **NOTE** that newer versions of the ES/detection scripts may introduce new parameters in `zmeventnotification.ini` and `objectconfig.ini`. You may need to paste them in manually, so always read *Breaking Changes*

6.7.5 STEP 5: Start the new updated server

```
sudo zmdc.pl start zmeventnotification.pl
```

Make sure you look at the logs to make sure its started properly

6.8 Configuring the notification server

6.8.1 Understanding zmeventnotification configuration

Starting v1.0, @synthead reworked the configuration (brilliantly) as follows:

- If you just run `zmeventnotification.pl` it will try and load `/etc/zm/zmeventnotification.ini`. If it doesn't find it, it will use internal defaults
- If you want to override this with another configuration file, use `zmeventnotification.pl --config /path/to/your/config/filename.ini`.
- Its always a good idea to validate you config settings. For example:

```
sudo /usr/bin/zmeventnotification.pl --check-config


Configuration (read /etc/zm/zmeventnotification.ini):

Secrets file.......................... /etc/zm/secrets.ini
Restart interval (secs)............... 172800

Port ................................. 9000
Address .............................. [::]
Event check interval ................. 5
Monitor reload interval .............. 300

Auth enabled ......................... yes
Auth timeout ......................... 20

Use FCM .............................. yes
FCM API key .......................... (defined)
Token file ........................... /var/lib/zmeventnotification/push/tokens.txt

Use MQTT ..............................no
MQTT Server ...........................127.0.0.1
MQTT Username .........................(undefined)
MQTT Password .........................(undefined)

SSL enabled .......................... yes
SSL cert file ........................ /etc/myserver/fullchain.pem
SSL key file ......................... /etc/myserver/privkey.pem

Verbose .............................. no
Read alarm cause ..................... yes
Tag alarm event id ................... yes
Use custom notification sound ........ no

Hook Script on Event Start ........... '/var/lib/zmeventnotification/bin/zm_event_
↪start.sh'
Hook Script on Event End.............. '/var/lib/zmeventnotification/bin/zm_event_end.
↪sh'

Notify on Event Start (hook success).. all
Notify on Event Start (hook fail)..... web
Notify on Event End (hook success)... fcm,web
Notify on Event End (hook fail)...... web

Notify End only if Start success......yes

Use Hook Description........... yes
Keep frame match type.......... yes
Skipped monitors.............. (undefined)
Store Frame in ZM..............yes
```

(continues on next page)

(continued from previous page)

```
Picture URL ................... https://myserver/zm/index.php?view=image&eid=EVENTID&
↪fid=objdetect&width=600
Include picture................ yes
Picture username .............. zmes
Picture password .............. (defined)
```

6.8.2 What is the hook section ?

The `hook` section allows you to invoke a custom script when an alarm is triggered by ZM.

`hook_script` points to the script that is invoked when an alarm occurs

If the script returns success (exit value of 0) then the notification server will send out an alarm notification. If not, it will not send a notification to its listeners. This is useful to implement any custom logic you may want to perform that decides whether this event is worth sending a notification for.

Related to `hook` we also have a `hook_description` attribute. When set to 1, the text returned by the hook script will overwrite the alarm text that is notified.

We also have a `hook_skip_monitors` attribute. This is a comma separated list of monitors. When alarms occur in those monitors, hooks will not be called and the ES will directly send out notifications (if enabled in ES). This is useful when you don't want to invoke hooks for certain monitors as they may be expensive (especially if you are doing object detection)

Finally, `keep_frame_match_type` is really used when you enable "bestmatch". It prefixes an `[a]` or `[s]` to tell you if object detection succeeded in the alarmed or snapshot frame.

Here is an example: (Note: just an example, please don't ask me for support for person detection)

- You will find a sample `zm_event_start.sh` script in the `hook` directory. This script is invoked by the notification server when an event starts.
- This script in turn invokes a python OpenCV based script that grabs an image with maximum score from the current event so far and runs a fast person detection routine.
- It returns the value "person detected" if a person is found and none if not
- The wrapper script then checks for this value and exits with either 0 (send alarm) or 1 (don't send alarm)
- the notification server then sends out a ": person detected" notification to the clients listening

Those who want to know more: - Read the detailed notes here - Read this for an explanation of how this works

6.9 Troubleshooting common situations

6.9.1 The ES randomly hangs

Short answer: Upgrade websocket library to 0.004000. You can do this by:

```
sudo -H cpan
# and inside the cpan shell
upgrade Net::WebSocket::Server
# exit after that
```

Explanation:

This is the reason why older versions of the websocket library would hang: If you have exposed the ES port (typically 9000) to the internet, there are chances your ES may lock up. The reason seems to be that that there are internet port scanners which establish a TCP connection that stays connected for a long time and does not upgrade to websockets. This causes the library which I use for the ES to handle websockets to lock up. The original issue can be viewed here.

If you want to disable censys, you can follow their instructions on their website to opt-out In Linux/ubuntu, I use ufw (make sure it is enabled) as a front-end to iptables and the following commmands do it:

```
sudo ufw deny from 74.120.14.0/24 comment "Deny censys"
sudo ufw deny from 162.142.125.0/24 comment "Deny censys"
sudo ufw deny from 167.248.133.0/24 comment "Deny censys"
sudo ufw deny from 192.35.168.0/23 comment "Deny censys"
```

Note that atleast on Ubuntu, ufw keeps getting disabled when I reboot, even after I enable it via systemctl. To fix that I had to add `After=netfilter-persistent.service` inside the `[Unit]` block of `/lib/systemd/system/ufw.service`

6.9.2 I can't connect to the ES

There could be multiple reasons:

- If you are connecting from WAN make sure you have set up port forwarding as needed
- Try changing the `address` attribute in `[network]` section of `zmeventnotification.ini`. If you don't have your IP specified, it will use `[::]`. Try `0.0.0.0` instead.

6.9.3 I just added a new monitor and the ES is not sending notifications for it

This generally happens if you add a monitor _after_ you configure the ES. What you need to do is go to zmNinja's `Menu->Settings->Event Server` option and enable the monitor you just added and press save.

6.9.4 The ES is missing events. I see them being triggered in ZM

There could be multiple issues:

- Let's start with the most obvious one. The ES and ZM need to be running on the same server
- Alarms are only triggered on Mocord, Modect and Nocord monitors
- If you changed monitor modes or added new monitors after the ES started running, restart the ES so it loads the latest information
- The ES polls ZM every 5 seconds for new alarms (the duration is controlled by `event_check_interval` in `zmeventnotification.ini`). This means that if your alarm is very brief, that is, it starts and ends before the ES polls for new events then it will be missed. Note that the ES will catch alarms both in `ALARM` and `ALERT` state. `ALARM` is when ZM is actually detecting motion in the event. `ALERT` is when ZM stops detecting motion but is still waiting around till it writes all your `post event frames` that you have configured on your ZM Monitor buffer settings. So here is an example: Let's say I have a "Garage" monitor that I've configured a post event buffer of 100 (frames) and I've set my camera FPS to 10. That means it will take ZM 10 seconds to close an event after my alarm occurs (it will be in `ALERT` stage all that time). In this case, no matter show short my actual alarm, the ES will always catch it.

6.9.5 LetsEncrypt certificates cannot be found when running as a web user

When the notification server is run in web user mode (example `sudo -u www-data`), the event notification server complains that it cannot find the certificate. The error is something like this:

```
zmeventnotification[10090].ERR [main:547] [Failed starting server: SSL_cert_file /etc/
↪letsencrypt/live/mysite.net-0001/fullchain.pem does not exist at /usr/share/perl5/
↪vendor_perl/IO/Socket/SSL.pm line 402.]
```

The problem is read permissions, starting at the root level. Typically doing `chown -R www-data:www-data /etc/letsencrypt` solves this issue

6.9.6 Picture notifications don't show images

Before you read this, make sure push notifications in general are working (without images). To get images working, the following conditions must be met:

- You must use HTTPS
- There is a 1MB limit to image size
- You can't use self signed certs
- The IP/hostname needs to be accessible by zmNinja on the mobile device you are receiving pushes on
- You need ZM 1.32.3 or above
- A good way to isolate if its a URL problem or something else is replace the `picture_url` in `/etc/zm/secrets.ini` with a knows HTTPS url like this Note that when you use a test image, comment out `picture_portal_username` and `picture_portal_password` so they are not auto appended. Remember to restart the ES.
- Once you have a public URL working as above, look at your ES DEBUG logs (not INF). When a push is beint sent out, you will notice a message like so:

  ```
  [|----> FORK:Driveway (7), eid:9666 fcmv1: Final JSON using FCMV1 being sent is: {
  ↪"title":"Driveway Alarm (9666)","image_url":"https://myserver:port/zm/index.php?
  ↪view=image&eid=9666&fid=objdetect&width=600&username=admin&password=xxx}, <etc>
  ```

  Take the URL inside `image_url` and replace the password with the actual password and paste it in your mobile device. If it works (without requiring to manually login), only then will push images show

Before you report issues, please make sure you have been diligent in testing - Try with a public URL as indicated above. This is important. - In your issue, post debug logs of zmeventnotification so I can see what message it is sending

6.9.7 Secure mode just doesn't work (WSS) - WS works

Try to put in your event server IP in the `address` token in `[network]` section of `zmeventnotification.ini`

6.9.8 I'm not receiving push notifications in zmNinja

This almost always happens when zmNinja is not able to reach the server. Before you contact me, please perform the following steps and send me the output:

1. Stop the event server. `sudo zmdc.pl stop zmeventnotification.pl`

2. Do a `ps -aef | grep zmevent` and make sure no stale processes are running
3. Edit your `/etc/zm/zmeventnotification.ini` and make sure `console_logs = yes` to get console debug logs
4. Run the server manually by doing `sudo -u www-data /usr/bin/zmeventnotification.pl --debug` (replace with `www-data` with `apache` depending on your OS)
5. You should now see logs on the commandline like so that shows the server is running:

```
018-12-20,08:31:32 About to start listening to socket
12/20/2018 08:31:32.606198 zmeventnotification[12460].INF [main:582] [About to start␣
↪listening to socket]
2018-12-20,08:31:32 Secure WS(WSS) is enabled...
12/20/2018 08:31:32.656834 zmeventnotification[12460].INF [main:582] [Secure WS(WSS)␣
↪is enabled...]
2018-12-20,08:31:32 Web Socket Event Server listening on port 9000
12/20/2018 08:31:32.696406 zmeventnotification[12460].INF [main:582] [Web Socket␣
↪Event Server listening on port 9000]
```

6. Now start zmNinja. You should see event server logs like this:

```
2018-12-20,08:32:43 Raw incoming message: {"event":"push","data":{"type":"token",
↪"platform":"ios","token":"cVuLzCBsEn4:APA91bHYuO3hVJqTIMsm0IRNQEYAUa<deleted>
↪GYBwNdwRfKyZV0","monlist":"1,2,4,5,6,7,11","intlist":"45,60,0,0,0,45,45","state":
↪"enabled"}}
```

If you don't see these logs on the event server, zmNinja is not able to connect to the event server. This may be because of several reasons:

- Your event server IP/DNS is not reachable from your phone
- If you are using SSL, your certificates are invalid (try disabling SSL first - both on the event server and on zmNinja)
- Your zmNinja configuration is wrong (the most common error I see is the server has SSL disabled, but zmNinja is configured to use `wss://` instead of `ws://`)

7. Assuming the above worked, go to zmNinja logs in the app. Somewhere in the logs, you should see a line similar to:

```
Dec 20, 2018 05:50:41 AM DEBUG Real-time event: {"type":"","version":"2.4","status":
↪"Success","reason":"","event":"auth"}
```

This indicates that the event server successfully authenticated the app. If you see step 6 work but not step 7, you might have provided incorrect credentials (and in that case, you'll see an error message)

8. Finally, after all of the above succeeds, do a `cat /var/lib/zmeventnotification/push/tokens.txt` to make sure the device token that zmNinja sent is stored (desktop apps don't have a device token). If you are using zmNinja on a mobile app, and you don't see an entry in `tokens.txt` you have a problem. Debug.
9. *Always* send me logs of both zmNinja and zmeventnotification - I need them to understand what is going on. Don't send me one line. You may think you are sending what is relevant, but you are not. One line logs are mostly useless.
10. Some other notes:

- If you are not using machine learning hooks, make sure you comment out the `hook_script` line in `[hook]`. If you have not setup the scripts correctly, if will fail and not send a push.

- If you don't see an entry in `tokens.txt` (typically in `/var/lib/zmeventnotification/push`) then your phone is not registered to get push. Kill zmNinja, start the app, make sure the event server receives the registration and check `tokens.txt`
- Sometimes, Google's FCM server goes down, or Apple's APNS server goes down for a while. Things automagically work in 24 hrs.
- Kill the app. Then empty the contents of `tokens.txt` in the event server (don't delete it). Then restart the event server. Start the app again. If you don't see a new registration token, you have a connection problem
- I'd strongly recommend you run the event server in "manual mode" and stop daemon mode while debugging.

6.9.9 I'm getting multiple notifications for the same event

Some possibilities:

- Most often, its because you have multiple copies of the eventserver running and you don't know it. Maybe you were manually testing it, and forgot to quit it and terminated the window. Do `sudo zmdc.pl stop zmeventnotification.pl` and then `ps -aef | grep zme`, kill everything, and start again. Monitor the logs to see how many times a message is sent out.
- There are situations where you device token has changed and `/var/lib/zmeventnotification/push/tokens.txt` has both the old and new token and both work. In this case, your device will get multiple notifications. Stop the ES, delete `tokens.txt` and let zmNinja re-register
- At times Google's FCM servers send out multiple notifications. Why? I don't know. But it sorts itself out very quickly, and if you think this must be the reason, I'll wager that you are actually in the 99.9% lot and haven't checked properly.

6.9.10 How do I reduce the time of delay from an alarm occuring in ZM to the notification being sent out?

- First, turn on debug logs. You'll know where the delays are occuring and then you can deep dive.
- Read the priniciples document: *From Event Detection to Notification*. Really, it will help you understand how this works
- There are some key areas you can optimize for:
  - The ES _polls_ ZoneMinder mapped memory for events. By default it is 5 seconds. To change it, change `event_check_interval` in `zm_eventnotification.ini`
  - One an alarm is detected, depending on whether you configured hooks or not, it will invoke object detection. Based on your server/desktop configuration, this can take just a few milliseconds to several seconds. If you are using machine learning hooks, consider using mlapi as it preloads models into memory only once. Loading a model can take a few seconds, while detection, if you are on a GPU or TPU takes milliseconds. If you don't use hooks, turn it off in config.
  - Again, if you are using hooks, there is a `wait` attribute in `objectconfig.ini` that waits for a few seconds before downloading the image from ZM. This was done because sometimes the ES may be asking to download an image that ZM hasn't written to disk yet (remember, ES is triggered when an alarm starts). This is really no longer needed, if you are using it. Starting ES 6.1.0, you can instead just use the much more powerful 'stream_sequence construct to specify retries.

This is really what comes to mind. If you are seeing unusual delays, please create a github issue and post debug logs (again, NOT info logs please)

6.9.11 Push notifications are delayed by several minutes when the phone turns off (Android)

There seems to be multiple potential reasons:

- Starting Android 6, a doze mode and battery optimization mode has been introduced which agressively tries to put the phone into low power mode. This results in the apps disconnecting from FCM servers for around 10-15 mins at a stretch, which may explain why you get delayed notifications. To avoid this, remove zmNinja from any battery optimization and doze mode effects. There are instructions here on how to do that (scroll to the middle of the page and see the table that describes what to do depending on your phone manufacturer).
- The ES delivers messages using `high` priority in Android. However, it looks like google _may_ *deprioritize <https://stackoverflow.com/questions/53646761/firebase-push-notification-delay-after-triggering-few-high-priority-notification>__* them if we send too many high priority messages. So try setting your `fcm_android_priority` to `normal` if you see this issue. However, if you set it to normal, messages may be delayed if you are in doze mode.
- Finally, experiment with setting `fcm_android_ttl` to `0` along with `fcm_android_priority` to `high`
- If nothing else works, set *use_fcmv1* to *no* in *zmeventnotification.ini* to go back to legacy protocol
- Finally, it is entirely possible there is some magic-foo of combination of attributes in FCMv1 which is not documented that may do the right thing. If you figure it out, please let me know.
- If you are wondering what this all means for iOS - it is unaffected. iOS uses a priority 10 by default (high) that delivers the notification instantly.

6.9.12 The server runs fine when manually executed, but fails when run in daemon mode (started by zmdc.pl)

(This only covers daemon mode of the ES server. If you are facing issues related to hooks running in daemon model, please see *Machine Learning Hooks FAQ*)

- Make sure the file where you store tokens (`/var/lib/zmeventnotification/push/tokens.txt or whatever you have used`) is not RW Root only. It needs to be RW `www-data` for Ubuntu/Debian or `apache` for Fedora/CentOS. You also need to make sure the directory is accessible. Something like `chown -R www-data:www-data /var/lib/zmeventnotification/push`
- Make sure your certificates are readable by `www-data` for Ubuntu/Debian, or `apache` for Fedora/CentOS (thanks to @jagee).
- Make sure the *path* to the certificates are readable by `www-data` for Ubuntu/Debian, or `apache` for Fedora/CentOS

6.9.13 When you run zmeventnotifiation.pl manually, you get an error saying 'port already in use' or 'cannot bind to port' or something like that

The chances are very high that you have another copy of `zmeventnotification.pl` running. You might have run it in daemon mode. Try `sudo zmdc.pl stop zmeventnotification.pl`. Also do `ps -aef | grep zmeventnotification` to check if another copy is not running and if you do find one running, you'll have to kill it before you can start it from command line again.

6.9.14 Running hooks manually detects the objects I want but fails to detect via ES (daemon mode)

There may be multiple reasons, but a common one is of timing. When the ES invokes the hook, it is invoked almost immediately upon event detection. In some cases, ZoneMinder still has not had time to create an alarmed frame, or the right snapshot frame. So what happens is that when the ES invokes the hook, it runs detection on a different image from the one you run later when invoked manually. Try adding a `wait = 5` to `objectconfig.ini` to that monitor section and see if it helps

6.9.15 Great Krypton! I just upgraded ZoneMinder and I'm not getting push anymore!

Make sure your eventserver is running: `sudo zmdc.pl status zmeventnotification.pl`

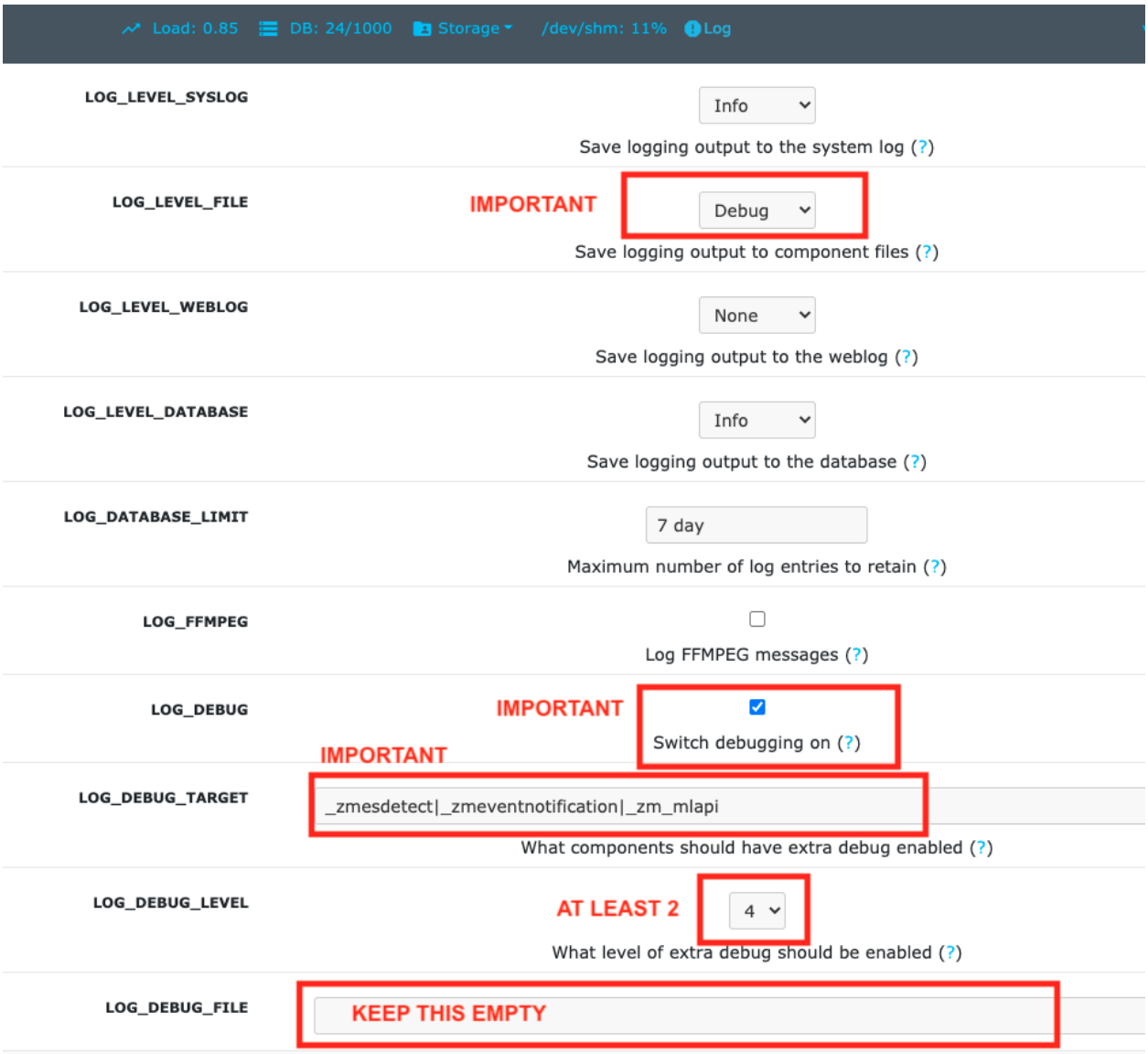
6.10 How do I disable secure (WSS) mode?

As it turns out many folks run ZM inside the LAN only and don't want to deal with certificates. Fair enough. For that situation, edit zmeventnotification.pl and use `enable = 0` in the `[ssl]` section of the configuration file. **Remember to ensure that your EventServer URL in zmNinja does NOT use wss too - change it to ws**. Also remember to restart the ES after this change.

6.10.1 Logging

Setting up logging in both ES and detection (if you use them) is critical to be able to diagnose issues. Here is what I do:

This gives you a summary of what you need to do:



- In `ZM->Options->Logs`:
  - LOG_DEBUG is on
  - LOG_LEVEL_FILE = debug
  - LOG_LEVEL_SYSLOG = Info
  - LOG_LEVEL_DATABASE = Info
  - LOG_DEBUG_TARGET = `_zmesdetect|_zmeventnotification`. This enables DEBUG logs for both detection and event server scripts. Iff you have other targets, just separate them with `|` - example, `_zmc|_zmesdetect`. If you only want to track detection logs and not ES logs, just do `_zmesdetect`. You can also enable debug logs for just one monitor's hooks like so: `_zmesdetect_m5|_zmeventnotification`. This will enable debug logs only when hooks are run for monitor 5. Just remember this: "detection logs" only deal with detecting objects. The ES logs will tell you whether the detection text was received properly, whether it was written to ZM DB properly and whether it was sent out in a notification.

The above config. will store debug logs in my `/var/log/zm` directory, while Info level logs will be recorded

in syslog and DB.

You will likely need to restart ZM after this.

So now, to view hooks/detect logs, all I do is:

```
tail -F  /var/log/zm/zmesdetect*.log
```

To view ES + hooks/detect logs:

```
tail -F  /var/log/zm/zmesdetect*.log /var/log/zm/zmeventnotification.log
```

Note that the detection code registers itself as `zmesdetect` with ZM. When it is invoked with a specific monitor ID (usually the case), then the component is named `zmesdetect_mX.log` where `X` is the monitor ID. In other words, that now gives you one log per monitor (just like `/var/log/zm/zmc_mX.log`) which makes it easy to debug/isolate. Also note we are doing `tail -F` not `tail -f`. `-F` tracks files as they get logrotated as well.

6.11 Debugging and reporting problems

STOP. Before you shoot me an email, **please** make sure you have read the *common problems* and have followed *every step* of the *install guide* and in sequence. I can't emphasize how important it is to be diligent.

STOP (redux): Please don't send me emails without relevant logs (unless of course, it is to do with situations where, say, zmNinja doesn't load and you can't extract logs). Read *Logging*.

Here is how to debug and report:

If your problem involves zmNinja:

- Enable Debug logs in zmNinja (Setting->Developer Options->Enable Debug Log), if zmNinja is part of the problem
- Clear zmNinja logs and then replicate the issue. Send me zmNinja logs right after that. That way it is easier for me to zero in to what the problem may be. If you send me a whole bunch of logs, unrelated to your issue, I'll likely not know what is going on.

If your problem involves the ES and/or the hooks:

- Enable ZM debug logs for both ES (and hooks if you use them) as described in *Logging*. Note that ES debug logs are different from hooks debug logs. You need to enable both if you use them.

When you send ES/detection logs:

```
03/19/20 06:45:03 zmesdetect_m2[21409] INF zm_detect.py:160 [---------| hook version:␣
↪5.10.1, ES version: 5.10 |------------]
```

This shows my ES version is `5.10` and hooks version is `5.10.1`, which is good. If you saw `5.9.4` and `5.10`, for example, we have a problem. Upgrade again and please upgrade both hooks and ES.

- Make sure you see `DBG` logs (Debug). If you only see `INF` logs, you haven't followed the instructions above to enable debug logs. Read *Logging* again.
- Don't just send me a slice of what you think is relevant. Please don't think you know what to send me. Let me decide that. From your side, send me the full logs. By full logs, I mean:
  - If you think your detection is *not* working for an event, say eid=77985, send me *all* the ES logs starting from `PARENT: New event 77985 reported for Monitor:<etc>` to `PARENT: Job: Deleting active_event eid:77985, mid:<etc>`. That is, everthing from start to end of that event. Also send me *all* the detection logs. Let's say the monitor in question was Monitor Id:2. Then the

detection logs will be in `/var/log/zm/zmesdetect_m2.log`. Send me *all* the logs from the start to the finish for that event.

- If you have issues starting the ES, send me *all* logs starting from when the ES starts after you do a `sudo zmdc.pl restart zmeventnotification.pl`

To get DEBUG logs:

Option 1 (Easy):

- Stop ES if it is running as a daemon (`sudo zmdc.pl stop zmeventnotification.pl`)
- Start a terminal and start zmeventnotification manually from command line like so `sudo -u www-data /usr/bin/zmeventnotification.pl --debug`
- This will print debug logs on the terminal
- Note that this will NOT print debug logs for hooks, so if you feel the problem is with hooks, see *this section*

Option 2 (More comprehensive): Enable ZM logs

- Follow steps *here*
- It is _very_ important to make sure you follow _all_ the steps or you won't see logs
- When you set up debug logging via ZM, you will see debug logs from both ES and hooks in different files like so:

```
pp@homeserver:~/fiddle/zmeventnotification$ tail -F /var/log/zm/zmeventnotification.
↪log /var/log/zm/zmesdetect_m*.log
==> /var/log/zm/zmeventnotification.log <==
10/06/2019 06:48:29.200008 zmeventnotification[13694].INF [main:557] [Invoking hook:'/
↪var/lib/zmeventnotification/bin/zm_event_start.sh' 33989 2 "DoorBell" " front" "/
↪var/cache/zoneminder/events/2/2019-10-06/33989"]
10/06/2019 06:48:34.013490 zmeventnotification[29913].INF [main:557] [New event 33990␣
↪reported for Monitor:10 (Name:FrontLawn)  front steps]
10/06/2019 06:48:34.020958 zmeventnotification[13728].INF [main:557] [Forking␣
↪process:13728 to handle 1 alarms]
10/06/2019 06:48:34.021347 zmeventnotification[13728].INF [main:557] [processAlarms:␣
↪EID:33990 Monitor:FrontLawn (id):10 cause: front steps]
10/06/2019 06:48:34.237147 zmeventnotification[13728].INF [main:557] [Adding event␣
↪path:/var/cache/zoneminder/events/10/2019-10-06/33990 to hook for image storage]
10/06/2019 06:48:34.237418 zmeventnotification[13728].INF [main:557] [Invoking hook:'/
↪var/lib/zmeventnotification/bin/zm_event_start.sh' 33990 10 "FrontLawn" " front␣
↪steps" "/var/cache/zoneminder/events/10/2019-10-06/33990"]
10/06/2019 06:48:46.529693 zmeventnotification[13728].INF [main:557] [For Monitor:10␣
↪event:33990, hook script returned with text: exit:1]
10/06/2019 06:48:46.529896 zmeventnotification[13728].INF [main:557] [Ending␣
↪process:13728 to handle alarms]
10/06/2019 06:48:47.640414 zmeventnotification[13694].INF [main:557] [For Monitor:2␣
↪event:33989, hook script returned with text: exit:1]
10/06/2019 06:48:47.640668 zmeventnotification[13694].INF [main:557] [Ending␣
↪process:13694 to handle alarms]

==> /var/log/zm/zmesdetect_m10.log <==
10/06/19 06:48:42 zmesdetect_m10[13732] DBG zm_detect.py:344 [No match found in /var/
↪lib/zmeventnotification/images/33990-alarm.jpg using model:yolo]
10/06/19 06:48:42 zmesdetect_m10[13732] DBG zm_detect.py:189 [Using model: yolo with /
↪var/lib/zmeventnotification/images/33990-snapshot.jpg]
10/06/19 06:48:46 zmesdetect_m10[13732] DBG zm_detect.py:194 [|--> model:yolo␣
↪detection took: 3.541227s]
```

- If you are debugging problems with receiving push notifications on zmNinja mobile, then replicate the following scenario:
  - Run the event server in manual mode as described above
  - Kill zmNinja
  - Start zmNinja
  - At this point, in the `zmeventnotification` logs you should registration messages (refer to logs example above). If you don't you've either not configured zmNinja to use the eventserver, or it can't reach the eventserver (very common problem)
  - Next up, make sure you are not running zmNinja in the foreground (move it to background or kill it). When zmNinja is in the foreground, it uses websockets to get notifications
  - Force an alarm like I described above. If you don't see logs in `zmeventnotification` saying "Sending notification over FCM" then the eventserver, for some reason, does not have your app token. Inspect `tokens.txt` (typically in `/etc/zm/`) to make sure an entry for your phone exists
  - If you see that message, but your mobile phone is not receiving a push notification:
  - Make sure you haven't disable push notifications on your phone (lots of people do this by mistake and wonder why)
  - Make sure you haven't muted notifications (again, lots of people...)
  - Sometimes, the push servers of Apple and Google stop forwarding messages for a day or two. I have no idea why. Give it a day or two?
  - Open up zmNinja, go right to logs and send it to me
  - If you have issues, please send me a copy of your zmeventnotification logs generated above from Terminal-Log, as well as zmNinja debug logs

6.12 Brickbats

Why not just supply the username and password in the URL as a resource? It's over TLS

Yup its encrypted but it may show up in the history of a browser you tried it from (if you are using a browser) Plus it may get passed along from the source when accessing another URL via the Referral header

So it's encrypted, but passing password is a bad idea. Why not some token?

- Well, now that ZM supports login tokens (starting 1.33), I'll get to supporting it, eventually.

Why WSS and not WS?

Not secure. Easy to snoop. Updated: As of 0.6, I've also added a non secure version - use `enable = 0` in the `[ssl]` section of the configuration file. As it turns out many folks don't expose ZM to the WAN and for that, I guess WS instead of WSS is ok.

Why ZM auth in addition to WSS?

WSS offers encryption. We also want to make sure connections are authorized. Reusing ZM authentication credentials is the easiest. You can change it to some other credential match (modify `validateZM` function)

CHAPTER 7

Machine Learning Hooks FAQ

7.1 My hooks run just fine in manual mode, but don't in daemon mode

The errors are almost always related to the fact that when run in daemon mode, python cannot find certain libraries (example `cv2`). This usually happens if you don't install these libraries globally (i.e. for all users).

To zero-in on what is going on:

- Make sure you have set up logging as per *Logging*. This should ensure these sort of errors are caught in the logs.
- Try and run the script manually the way the daemon calls it. You will see the invocation in `zmeventnotification.log`. Example:

```
FORK:DoorBell (2), eid:175153 Invoking hook on event start:'/var/lib/
↪zmeventnotification/bin/zm_event_start.sh' 175153 2 "DoorBell" " front"
↪"/var/cache/zoneminder/events/2/2020-12-13/175153"]
```

  So invoke manually like so:

```
sudo -u www-data '/var/lib/zmeventnotification/bin/zm_event_start.sh'␣
↪175153 2 "DoorBell" " front" "/var/cache/zoneminder/events/2/2020-12-13/
↪175153"
```

  The *-u www-data* is important (replace with whatever your webserver user name is)

One user reported that they never saw logs. I get the feeling its because logs were not setup correctly, but there are some other insights worth looking into. See here

7.2 I get a segment fault/core dump while trying to use opencv in detection

See *Make sure OpenCV works*.

7.3 I am trying to use YoloV4 and I see errors in OpenCV

- If you plan to use YoloV4 (full or Tiny) the minimum version requirement OpenCV 4.4. So if you suddently see an error like: `Unsupported activation: mish in function 'ReadDarknetFromCfgStream'` popping up with YoloV4, that is a sign that you need to get a later version of OpenCV.

7.4 Necessary Reading - Sample Config Files

The sample configuration files, zmeventnotification.ini and objectconfig.ini come with extensive commentary about each attribute and what they do. Please go through them to get a better understanding. Note that most of the configuration attributes in *zmeventnotification.ini* are not related to machine learning, except for the *[hook]* section.

7.5 How do the hooks actually invoke object detection?

- When the Event Notification Server detects an event, it invokes the script specified in `event_start_hook` in your `zmeventnotification.ini`. This is typically `/var/lib/zmeventnotification/bin/zm_event_start.sh`
- `zm_event_start.sh` in turn invokes `zm_detect.py` that does the actual machine learning. Upon exit, it either returns a `1` that means object found, or a `0` which means nothing found. Based on how you have configured your settings, this information is then stored in ZM and/or pushed to your mobile device as a notification.

7.6 How To Debug Issues

- Refer to *Logging*

7.7 It looks like when ES invokes the hooks, it misses objects, but when I run it manually, it detects it just fine

This is a very common situation and prior to ZM 1.34 there was also a bug. Here is what is likely happening:

- If you have configured `BESTMATCH` then the hooks will search for both your "alarmed" frame and the "snapshot" frame for objects. If you have configured `snapshot`, `alarm` or a specfic `fid=xx` only that frame will be searched
- An 'alarm' frame is the first frame that caused the motion trigger
- A 'snapshot' frame is the frame with the *highest* score in the event

The way ZM works is that the 'snapshot' frame may keep changing till the full event is over. This is because as event frames are analyzed, if their 'score' is higher than the current snapshot score, the frame is replaced.

Next up, the 'alarm' frame is much more static, but prior to version 1.34, ZM took finite time (typically a few seconds) to actually write the alarmed frame to disk. In 1.34 changes were made to write them as soon as possible, but it may still take some finite time. If the alarm frame is not written by the time the ES requests it, ZM will return the first frame.

What is likely happening in your case is that when the ES invokes the hooks, your snapshot frame is the current frame with the highest score, and your alarmed frame may or may not be written to disk yet. So the hooks run on what is available.

However, when you run it manually later, your snapshot image has likely changed. It is possible as well that your alarmed frame exists now, whereas it did not exist before.

7.7.1 How do I make sure this is what is happening?

- Enable `write_debug_image` in `objectconfig.ini`. This will create a debug image inside the event path where your event recording is. Take a look at the debug images it creates. Is it the same as the images you see at a later date? If not, you know this is exactly what is happening
- When you run the detection script manually, see if its printing an `[a]` or an `[s]` before the detected text. The latter means `snapshot` and if that is so, the chances are very high this is exactly what the issue is. Incase it prints `[a]` it also means the same thing, but the occurrence of this is less than snapshot.

7.7.2 How do I solve this issue?

- If you are running ZM 1.32 or below, upgrade to 1.34 (1.33 master as of Oct 2019). This *should* fix the issue of delayed alarm frame writes
- If you are running ZM 1.32 or below, turning off JPEG store will help. When JPEG store is enabled, snapshots are written later. This bug was fixed in 1.34 (see this issue).
- Add a `wait: 5` to that monitor in `objectconfig.ini` (again, please read the ini file to understand). This delays hook execution by 5 seconds. The hope here is that the time specified is sufficient for the alarmed frame and the right snapshot to be written to disk
- Fix your zone triggers. This is really the right way. If you use object detection, re-look at how your zone triggers to be able to capture the object of interest as soon as possible. If you do that, chances are high that by the time the script runs, the image containing the object will be written to disk.

7.8 I'm having issues with accuracy of Face Recognition

- Use `cnn` mode in face recognition. Much slower, but far more accurage than `hog`
- Look at debug logs.
  - If it says "no faces loaded" that means your known images don't have recognizable faces
  - If it says "no faces found" that means your alarmed image doesn't have a face that is recognizable
  - Read comments about `num_jitters`, `model`, `upsample_times` in `objectconfig.ini`
- Experiment. Read the accuracy wiki link.

7.9 I am using a Coral TPU and while it works fine, at times it fails loading

If you have configured the TPU properly, and on occassion you see an error like:

```
Error running model: Failed to load delegate from libedgetpu.so.1
```

then it is likely that you either need to replace your USB cable or need to reset your USB device. In my case, after I set it up correctly, it would often show the error above during runs. I realized that replacing the USB cable that Google provided solved it for a majority of cases. See this comment for my experience on the cable. After buying the cable, I still saw it on occassion, but not frequently at all. In those cases, resetting USB works fine and you don't have to reboot. See this comment.

7.10 Local vs. Remote server for Machine Learning

As of version 5.0.0, you can now configure an API gateway for remote machine learning by installing my mlapi server on a remote server. Once setup, simply point your `ml_gateway` inside `objectconfig.ini` to the IP/port of your gateway and make sure `ml_user` and `ml_password` are the user/password you set up on the API gateway. That's all.

The advantage of this is that you don't need to install any ML libraries within zoneminder if you are running mlapi on a different server. Further, mlapi loads the model only once so it is much faster. In older versions this was kludgy because you still had to install ML libraries locally in ZM, but no longer. Infact, I've completely switched to mlapi now for my own use. Note that when you use remote detection, you will still need opencv in the host machine (opencv is used for other functions)

CHAPTER 8

For Developers writing their own consumers

8.1 How do I talk to it?

- `{"JSON":"everywhere"}`
- Your client sends messages (authentication) over JSON
- The server sends auth success/failure over JSON back at you
- New events are reported as JSON objects as well
- By default the notification server runs on port 9000 (unless you change it)
- You need to open a secure web socket connection to that port from your client/consumer
- You then need to provide your authentication credentials (ZoneMinder username/password) within 20 seconds of opening the connection
- If you provide an incorrect authentication or no authentication, the server will close your connection
- As of today, there are 2 categories of messages:
- 'normal' messages that are exchanged between a client (like zmNinja) and the ES. These messages are the following types: - auth (from client to server) - control (from client to server) - push (only applicable for zmNinja) - alarm notifications (from server to client)
- 'escontrol' messages. This allows the client to change the behaviour of the ES dynamically. The changes are stored persistently in `/var/lib/zmeventnotification/misc/admin_interface.txt`.

8.1.1 Category: Normal messages

Authentication messages

To connect with the server you need to send the following JSON object (replace username/password) Note this payload is NOT encrypted. If you are not using SSL, it will be sent in clear.

Authentication messages can be sent multiple times. It is necessary that you send the first one within 20 seconds of opening a connection or the server will terminate your connection.

Client –> Server:

```
{"event":"auth","data":{"user":"<username>","password":"<password>"}}
```

Server –> Client: The server will send back one of the following responses

Authentication successful:

```
{"event":"auth", "type":"", "version":"0.2","status":"Success","reason":""}
```

Note that it also sends its version number for convenience

Incorrect credentials:

```
{"event":"auth", "type":"", "status":"Fail","reason":"BADAUTH"}
```

No authentication received in time limit:

```
{"event":"auth","type":"", "status":"Fail","reason":"NOAUTH"}
```

Control messages

Control messages manage the nature of notifications received/sent. As of today, Clients send control messages to the Server. In future this may be bi-directional

Control message to restrict monitor IDs for events as well as interval durations for reporting

A client can send a control message to restrict which monitor IDs it is interested in. When received, the server will only send it alarms for those specific monitor IDs. You can also specify the reporting interval for events.

Client–>Server:

```
{"event":"control","data":{"type":"filter","monlist":"1,2,4,5,6", "intlist":"0,0,3600,
↪60,0"}}
```

In this example, a client has requested to be notified of events only from monitor IDs 1,2,4,5 and 6 Furthermore it wants to be notified for each alarm for monitors 1,2,6. For monitor 4, it wants to be notified only if the time difference between the previous and current event is 1 hour or more (3600 seconds) while for monitor 5, it wants the time difference between the previous and current event to be 1 minute (60 seconds)

There is no response for this request, unless the payload did not have either monlist or intlist.

No monitorlist received:

```
{"event":"control","type":"filter", "status":"Fail","reason":"NOMONITORLIST"}
```

No interval received:

```
{"event":"control","type":"filter", "status":"Fail","reason":"NOINTERVALLIST"}
```

Note that if you don't want to specify intervals, send it a interval list comprising of comma separated 0's, one for each monitor in monitor list.

Control message to get Event Server version

A client can send a control message to request Event Server version

Client–>Server:

```
{"event":"control","data":{"type":"version"}}
```

Server–>Client:

```
{"event":"control", "type:":"version", "version":"0.2","status":"Success","reason":""}
```

Alarm notifications

Alarms are events sent from the Server to the Client

Server–>Client: Sample payload of 2 events being reported:

```
{"event":"alarm", "type":"", "status":"Success", "events":[{"EventId":"5060","Name":
↪"Garage","MonitorId":"1"},{"EventId":"5061","MonitorId":"5","Name":"Unfinished"}]}
```

Push Notifications (for both iOS and Android)

To make Push Notifications work, please make sure you read the section on enabling Push for the event server.

Concepts of Push and why it is only for zmNinja

Both Apple and Google ensure that a "trusted" application server can send push notifications to a specific app running in a device. If they did not require this, anyone could spam apps with messages. So in other words, a "Push" will be routed from a specific server to a specific app. Starting Jan 2018, I am hosting my trusted push server on Google's Firebase cloud. This eliminates the need for me to run my own server.

Registering Push token with the server

Client–>Server:

Registering an iOS device:

```
{"event":"push","data":{"type":"token","platform":"ios","token":"<device tokenid here>
↪", "state":"enabled"}}
```

Here is an example of registering an Android device:

```
{"event":"push","data":{"type":"token","platform":"android","token":"<device tokenid␣
↪here>", "state":"enabled"}}
```

For devices capable of receiving push notifications, but want to stop receiving push notifications over APNS/GCM and have it delivered over websockets instead, set the state to disabled

For example: Here is an example of registering an Android device, which disables push notifications over GCM:

```
{"event":"push","data":{"type":"token","platform":"android","token":"<device tokenid␣
↪here>", "state":"disabled"}}
```

What happens here is if there is a new event to report, the Event Server will send it over websockets. This means if the app is running (foreground or background in Android, foreground in iOS) it will receive this notification over the open websocket. Note that in iOS this means you won't receive notifications when the app is not running in the foreground. We went over why, remember?

Server–>Client: If its successful, there is no response. However, if Push is disabled it will send back

```
{"event":"push", "type":"", "status":"Fail", "reason": "PUSHDISABLED"}
```

Badge reset

Only applies to iOS. Android push notifications don't have a concept of badge notifications, as it turns out.

In push notifications, the server owns the responsibility for badge count (unlike local notifications). So a client can request the server to reset its badge count so the next push notification starts from the value provided.

Client–>Server:

```
{"event":"push", "data":{"type":"badge", "badge":"0"}}
```

In this example, the client requests the server to reset the badge count to 0. Note that you can use any other number. The next time the server sends a push via APNS, it will use this value. 0 makes the badge go away.

8.1.2 Category: escontrol messages

You can now control the ES dynamically using websockets. As of now, you can do the following:

- mute all notifications
- unmute all notifications
- restart the ES
- reset all customizations made in the ES control admin_interface

Note that any changes you make are persistently stored in the file specified in `escontrol_interface_file` attribute, which by default is `/var/lib/zmeventnotification/misc/escontrol_interface.dat`. This makes sure all settings are persistent across reboots.

escontrol authentication

Just like normal messages, you need to authenticate yourself. The password is specified by what you choose in `escontrol_interface_password` attribute inside `zmeventnotification.ini`.

To authenticate: **Client–>Server:**

```
{"event":"auth","category":"escontrol","data":{"password":"whatever" }}
```

Server–>Client:

```
{"type":"","reason":"","event":"auth","version":"5.7","status":"Success"}
```

escontrol commands

Get current control channel settings:

Client–>Server:

```
{"event": "escontrol", "data": {"command": "get"}}
```

Server–>Client:

```
{"request":{"data":{"command":"get"},"event":"escontrol"},"response":"{\
↪"notifications\":{\"9\":1,\"8\":1,\"10\":1,\"2\":1,\"5\":1,\"6\":1}}","event":
↪"escontrol","type":"","status":"Success"}
```

Thes only show Client–>Server messages. Responses are not shown.

Mute all notifications:

```
{"event":"escontrol", "data":{"command":"mute"}}
```

Unmute all notifications:

```
{"event":"escontrol", "data":{"command":"unmute"}}
```

Mute only notifications for monitor IDs 2,4,6 (other IDs retain old values):

```
{"event":"escontrol", "data":{"command":"mute", "monitors":[2,4,6]}}
```

Unmute only notifications for monitors 8,12,14 (other IDs retain old values):

```
{"event":"escontrol", "data":{"command":"unmute","monitors":[8,12,14]}}
```

Restart the ES:

```
{"event":"escontrol", "data":{"command":"restart"}}
```

Reset/Clear all settings specified via this channel:

{“event”:”escontrol”, “data”:{“command”:”reset”}}

Change any abitrary config value inside `zmeventnotification.ini`:

```
{"event":"escontrol", "data":{"command":"edit", "key":"use_hooks", "val":"no"}}
```

In the above example, we have disabled hooks dynamically (`use_hooks` is the attribute inside `zmeventnotification.ini` that controls if hooks will be used)

8.1.3 Testing from command line

If you are writing your own consumer/client it helps to test the event server commands from command line. The event server uses Secure/WebSockers so you can't just HTTP to it using tools like `curl`. You'll need to use a websocket client. While there are examples on the net on how to use `curl` for websockets, I've found it much simpler to use wscat like so:

```
wscat -c wss://myzmeventnotification.domain:9000 -n
connected (press CTRL+C to quit)
> {"event":"auth","data":{"user":"admin","password":"xxxx"}}
< {"reason":"","status":"Success","type":"","event":"auth","version":"0.93"}
```

In the example above, I used `wscat` to connect to my event server and then sent it a JSON login message which it accepted and acknowledged.

CHAPTER 9

Guidelines for contrib

The `contrib/` directory is for user provided scripts. These scripts could be for push, hooks or other things.

Scripts in `contribs` are NOT supported by me. Please don't ask me questions on why some of them don't work like you want them to. Reach out directly to the author. Which leads me to:

9.1 Contribution notes for developers

If you are contributing a script, and want me to merge the PR into mainline, you need to follow these guidelines:

- Place your script inside the `contrib/` folder. `install.sh` will automatically take care of moving it.
- Please put in proper comments on purpose/use.
- Make sure it is a general purpose script that is useful for others - if your script is fine tuned for a specific use-case that is not widely applicable, please don't create a PR. Keep it for your personal use.
- Please put in your github ID when doing PRs to contrib. It is expected that you support for script if there are questions. If you are not willing to do that, please don't create a PR (and if you do, I'm sorry, I can't accept it). What will then happen is if people post an issue about your script, they will either tag you, or I will. You are also free to point a link to a github issue tracker in your own ID if you want
- Use the right ES trigger to invoke your script. If your script does some housekeeping after object detection is done, `event_xxx_hook_notify_userscript` is probably the right trigger. If you are contributing a new push notification mechanism, `api_push_script` is the probably right trigger.
- Depending on which trigger you are using, take a look at the example scripts to see what arguments you get

Github Repository

Key Principles - Event Notification Server and Hooks Key principles of how things work. Read this to understand how everything ties together.

Breaking Changes Breaking changes. Always read this if you are upgrading (for example, lots changed with 3.x and 3.2)

Installation of the Event Server (ES) How to install the Event Notification Server

Machine Learning Hooks How to configure the machine learning hooks *after* you install the Event Server

Configuration Guide How to use config files

Event Notification Server FAQ Event Notification Server FAQ covering common scenarios/issues

Machine Learning Hooks FAQ Machine Learning Hooks FAQ covering common scenarios/issues

For Developers writing their own consumers If you want to use the Event Notification Server to make your own app/client

Guidelines for contrib If you want to contribute hook scripts to the ES

zmNinja Documentation Documentation for zmNinja