N1-智慧家庭異質服務整合平台

安裝與開發指南說明書

工研院資通所

中華民國 105 年 12 月

一、技術項目簡介

隨著物聯網應用興起，硬體與軟體服務等異質服務的整合日趨重要，本平台提供智慧聯網產品與各項網路的串連服務，依不同網路服務的排列組合，創造出智慧生活體驗。

二、應用範圍說明

智慧家電、聯網產品、網路服務。
智慧家庭異質服務整合平台可以應用於聯網家電產品（如：插座、冰箱等）與網路服務（如：社群平台、天氣等）之情境應用，透過本技術平台，提供使用者不一樣的生活體驗。

三、安裝指南說明

Manual Installation

- Requirements Software and hardware requirements to run the Huginn installation
- Install Installation guide for Ubuntu/Debian
- Update Update an existing Huginn installation

Deploy updates via Capistrano

Requirements

Operating Systems

Supported Unix distributions by this guide

Ubuntu (16.04, 14.04 and 12.04)

Debian (Jessie and Wheezy)

Unsupported Unix distributions

- CentOS
- Red Hat Enterprise Linux
- OS X
- Arch Linux
- Fedora
- Gentoo
- FreeBSD

On the above unsupported distributions is still possible to install Huginn, and many people do. Follow the installation guide and

substitute the `apt` commands with the corresponding package manager commands of your distribution.

Non-Unix operating systems such as Windows

Huginn is developed for Unix operating systems. Huginn does **not** run on Windows and we have no plans of supporting it in the near future. Please consider using a virtual machine to run Huginn on Windows.

Ruby versions

Huginn requires Ruby (MRI) 2.2 or 2.3. You will have to use the standard MRI implementation of Ruby. We love JRuby and Rubinius but Huginn needs several Gems that have native extensions.

Hardware requirements

CPU

- *single core* setups will work but depending on the amount of Huginn Agents and users it will run a bit slower since the application server and background jobs can not run simultaneously
- *dual core* setups are the **recommended** system/vps and will work well for a decent amount of Agents
- 3+ cores can be needed when running multiple DelayedJob workers

Memory

You need at least 0.5GB of physical and 0.5GB of addressable memory (swap) to install and use Huginn with the default configuration! With less memory you need to manually adjust the `Gemfile` and Huginn can respond with internal server errors when accessing the web interface.

- 256MB RAM + 0.5GB of swap is the absolute minimum but we strongly **advise against** this amount of memory. See the Wiki page about running Huginn on systems with low memory
- 0.5GB RAM + 0.5GB swap will work relatively well with SSD drives, but can feel a bit slow due to swapping
- 1GB RAM + 1GB swap will work with two unicorn workers and the threaded background worker
- **2GB RAM** is the **recommended** memory size, it will support 2 unicorn workers and both the threaded and the old separate workers
- for each 300MB of additional RAM you can run one extra DelayedJob worker

Unicorn Workers

It's possible to increase the amount of unicorn workers and this will usually help for to reduce the response time of the applications and increase the ability to handle parallel requests.

For most instances we recommend using: CPU cores = unicorn workers.

If you have a 512MB machine we recommend to configure only one Unicorn worker and use the threaded background worker to prevent excessive swapping.

DelayedJob Workers

A DelayedJob worker is a separate process which runs your Huginn Agents. It fetches Websites, polls external services for updates, etc. Depending on the amount of Agents and the check frequency of those you might need to run more than one worker (like it is done in the threaded setup).

Estimating the amount of workers needed is easy. One worker can perform just one check at a time.

If you have 60 Agents checking websites every minute which take about 1 second to respond, one worker is fine.

If you need more Agents or are dealing with slow/unreliable websites/services, you should consider running additional workers.

Installation from source

Important Notes

This guide is long because it covers many cases and includes all commands you need.

This installation guide was created for and tested on **Debian/Ubuntu** operating systems. Please read doc/install/requirements.md for hardware and operating system requirements.

This is the official installation guide to set up a production server. To set up a **development installation** or for many other installation options please see the getting started section of the readme.

The following steps have been known to work. Please **use caution when you deviate** from this guide. Make sure you don't violate any assumptions Huginn makes about its environment. For example many

people run into permission problems because they change the location of directories or run services as the wrong user.
If you find a bug/error in this guide please **submit a pull request**.
If not stated otherwise all commands should be run as user with sudo permissions or as root.
When having problems during the installation please check the troubleshooting section.

Overview

The Huginn installation consists of setting up the following components:

- Packages / Dependencies
- Ruby
- System Users
- Database
- Huginn
- Nginx

1. Packages / Dependencies

`sudo` is not installed on Debian by default. Make sure your system is up-to-date and install it.

```
# run as root!
apt-get update -y
apt-get upgrade -y
apt-get install sudo -y
```

Note: During this installation some files will need to be edited manually. If you are familiar with vim set it as default editor with the commands below. If you are not familiar with vim please skip this and keep using the default editor.

```
# Install vim and set as default editor
sudo apt-get install -y vim
sudo update-alternatives --set editor
/usr/bin/vim.basic
```

Import node.js repository (can be skipped on Ubuntu and Debian Jessie):

```
curl -sL https://deb.nodesource.com/setup_0.12 | sudo
bash -
```
Install the required packages (needed to compile Ruby and native extensions to Ruby gems):
```
sudo apt-get install -y runit build-essential git
zlib1g-dev libyaml-dev libssl-dev libgdbm-dev
libreadline-dev libncurses5-dev libffi-dev curl
openssh-server checkinstall libxml2-dev libxslt-dev
libcurl4-openssl-dev libicu-dev logrotate python-
docutils pkg-config cmake nodejs graphviz
```
2. Ruby

The use of Ruby version managers such as RVM, rbenv or chruby with Huginn in production frequently leads to hard-to-diagnose problems. Version managers are not supported and we strongly advise everyone to follow the instructions below to use a system Ruby.

Remove the old Ruby versions if present:
```
sudo apt-get remove -y ruby1.8 ruby1.9
```
Download Ruby and compile it:
```
mkdir /tmp/ruby && cd /tmp/ruby
curl -L --progress http://cache.ruby-
lang.org/pub/ruby/2.3/ruby-2.3.1.tar.bz2 | tar xj
cd ruby-2.3.1
./configure --disable-install-rdoc
make -j`nproc`
sudo make install
```
Install the bundler and foreman gems:
```
sudo gem install rake bundler foreman --no-ri --no-
rdoc
```
3. System Users

Create a user for Huginn:
```
sudo adduser --disabled-login --gecos 'Huginn' huginn
```
4. Database

Install the database packages

```
sudo apt-get install -y mysql-server mysql-client
libmysqlclient-dev
# Pick a MySQL root password (can be anything), type
it and press enter,
# retype the MySQL root password and press enter
```

Check the installed MySQL version (remeber if its >= 5.5.3 for the `.env` configuration done later):

```
mysql --version
```

Secure your installation

```
sudo mysql_secure_installation
```

Login to MySQL

```
mysql -u root -p
# Type the MySQL root password
```

Create a user for Huginn do not type the `mysql>`, this is part of the prompt. Change `$password` in the command below to a real password you pick

```
mysql> CREATE USER 'huginn'@'localhost' IDENTIFIED BY
'$password';
```

Ensure you can use the InnoDB engine which is necessary to support long indexes

```
mysql> SET default_storage_engine=INNODB;
# If this fails, check your MySQL config files (e.g.
`/etc/mysql/*.cnf`, `/etc/mysql/conf.d/*`)
# for the setting "innodb = off"
```

Grant the Huginn user necessary permissions on the database

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE,
DROP, INDEX, ALTER, LOCK TABLES ON
`huginn_production`.* TO 'huginn'@'localhost';
```

Quit the database session

```
mysql> \q
```

Try connecting to the new database with the new user

```
sudo -u huginn -H mysql -u huginn -p -D
huginn_production
```

```
# Type the password you replaced $password with
earlier
```
You should now see `ERROR 1049 (42000): Unknown database 'huginn_production'` which is fine because we will create the database later.
You are done installing the database and can go back to the rest of the installation.

5. Huginn

Clone the Source

```
# We'll install Huginn into the home directory of the
user "huginn"
cd /home/huginn
# Clone Huginn repository
sudo -u huginn -H git clone
https://github.com/cantino/huginn.git -b master
huginn
# Go to Huginn installation folder
cd /home/huginn/huginn
# Copy the example Huginn config
sudo -u huginn -H cp .env.example .env
# Create the log/, tmp/pids/ and tmp/sockets/
directories
sudo -u huginn mkdir -p log tmp/pids tmp/sockets
# Make sure Huginn can write to the log/ and tmp/
directories
sudo chown -R huginn log/ tmp/
sudo chmod -R u+rwX,go-w log/ tmp/
# Make sure permissions are set correctly
sudo chmod -R u+rwX,go-w log/
sudo chmod -R u+rwX tmp/
sudo -u huginn -H chmod o-rwx .env
# Copy the example Unicorn config
sudo -u huginn -H cp config/unicorn.rb.example
config/unicorn.rb
```

Configure it

```
# Update Huginn config file and follow the
instructions
sudo -u huginn -H editor .env
```

If you are using a local MySQL server the database configuration should look like this (use the password of the huginn MySQL user you created earlier):

```
DATABASE_ADAPTER=mysql2
DATABASE_RECONNECT=true
DATABASE_NAME=huginn_production
DATABASE_POOL=20
DATABASE_USERNAME=huginn
DATABASE_PASSWORD='$password'
#DATABASE_HOST=your-domain-here.com
#DATABASE_PORT=3306
#DATABASE_SOCKET=/tmp/mysql.sock
DATABASE_ENCODING=utf8
# MySQL only: If you are running a MySQL server
>=5.5.3, you should
# set DATABASE_ENCODING to utf8mb4 instead of utf8 so
that the
# database can hold 4-byte UTF-8 characters like
emoji.
#DATABASE_ENCODING=utf8mb4
```

Important: Uncomment the RAILS_ENV setting to run Huginn in the production rails environment

```
RAILS_ENV=production
```

Change the Unicorn config if needed, the requirements.md has a section explaining the suggested amount of unicorn workers:

```
# Increase the amount of workers if you expect to
have a high load instance.
# 2 are enough for most use cases, if the server has
less then 2GB of RAM
# decrease the worker amount to 1
```

```
sudo -u huginn -H editor config/unicorn.rb
```

Important Note: Make sure to edit both `.env` and `unicorn.rb` to match your setup.

Note: If you want to use HTTPS, which is what we recommend, see Using HTTPS for the additional steps.

Note: For configuration changes after finishing the initial installation you have to re-export (see Install Init Script) the init script every time you change `.env`, `unicorn.rb` or your `Procfile`!

Install Gems

Note: As of bundler 1.5.2, you can invoke `bundle install -jN` (where `N` the number of your processor cores) and enjoy parallel gem installation with measurable difference in completion time (~60% faster). Check the number of your cores with `nproc`. For more information check this post. First make sure you have bundler >= 1.5.2 (run `bundle -v`) as it addresses some issues that were fixed in 1.5.2.

```
sudo -u huginn -H bundle install --deployment --
without development test
```

Initialize Database

```
# Create the database
sudo -u huginn -H bundle exec rake db:create
RAILS_ENV=production
# Migrate to the latest version
sudo -u huginn -H bundle exec rake db:migrate
RAILS_ENV=production
# Create admin user and example agents using the
default admin/password login
sudo -u huginn -H bundle exec rake db:seed
RAILS_ENV=production SEED_USERNAME=admin
SEED_PASSWORD=password
```

When done you see `See the Huginn Wiki for more Agent examples! https://github.com/cantino/huginn/wiki`

Note: This will create an initial user, you can change the username and password by supplying it in environmental variables `SEED_USERNAME` and `SEED_PASSWORD` as seen above. If you don't change the password

(and it is set to the default one) please wait with exposing Huginn to the public internet until the installation is done and you've logged into the server and changed your password.

Compile Assets

```
sudo -u huginn -H bundle exec rake assets:precompile RAILS_ENV=production
```

Install Init Script

Huginn uses foreman to generate the init scripts based on a `Procfile`
Edit the `Procfile` and choose one of the suggested versions for production

```
sudo -u huginn -H editor Procfile
```

Comment out (disable) these two lines

```
web: bundle exec rails server -p ${PORT-3000} -b ${IP-0.0.0.0}
jobs: bundle exec rails runner bin/threaded.rb
```

Enable (remove the comment) from these lines or those

```
# web: bundle exec unicorn -c config/unicorn.rb
# jobs: bundle exec rails runner bin/threaded.rb
```

Export the init scripts:

```
sudo bundle exec rake production:export
```

Note: You have to re-export the init script every time you change the configuration in `.env` or your `Procfile`!

Setup Logrotate

```
sudo cp deployment/logrotate/huginn /etc/logrotate.d/huginn
```

Ensure Your Huginn Instance Is Running

```
sudo bundle exec rake production:status
```

6. Nginx

Note: Nginx is the officially supported web server for Huginn. If you cannot or do not want to use Nginx as your web server, the wiki has a page on how to configure apache.

Installation

```
sudo apt-get install -y nginx
```

Site Configuration

Copy the example site config:

```
sudo cp deployment/nginx/huginn /etc/nginx/sites-available/huginn
sudo ln -s /etc/nginx/sites-available/huginn /etc/nginx/sites-enabled/huginn
```

Make sure to edit the config file to match your setup, if you are running multiple nginx sites remove the `default_server` argument from the `listen` directives:

```
# Change YOUR_SERVER_FQDN to the fully-qualified
# domain name of your host serving Huginn.
sudo editor /etc/nginx/sites-available/huginn
```

Remove the default nginx site, **if huginn is the only enabled nginx site**:

```
sudo rm /etc/nginx/sites-enabled/default
```

Note: If you want to use HTTPS, which is what we recommend, replace the `huginn` Nginx config with `huginn-ssl`. See Using HTTPS for HTTPS configuration details.

Test Configuration

Validate your `huginn` or `huginn-ssl` Nginx config file with the following command:

```
sudo nginx -t
```

You should receive `syntax is okay` and `test is successful` messages. If you receive errors check your `huginn` or `huginn-ssl` Nginx config file for typos, etc. as indicated in the error message given.

Restart

```
sudo service nginx restart
```

Done!

Initial Login

Visit YOUR_SERVER in your web browser for your first Huginn login.

The setup has created a default admin account for you. You can use it to log in:

```
admin (or your SEED_USERNAME)
password (or your SEED_PASSWORD)
```

Enjoy! ✨ ⭐ 🌴

You can use `cd /home/huginn/huginn && sudo bundle exec rake production:start` and `cd /home/huginn/huginn && sudo bundle exec rake production:stop` to start and stop Huginn.

Be sure to read the section about how to update your Huginn installation as well! You can also use Capistrano to keep your installation up to date.

Note: We also recommend applying standard security practices to your server, including installing a firewall (ufw is good on Ubuntu and also available for Debian).

Advanced Setup Tips

Using HTTPS

To use Huginn with HTTPS:

- In `.env`:
  - Set the `FORCE_SSL` option to `true`.
- Use the `huginn-ssl` Nginx example config instead of the `huginn` config:
  - `sudo cp deployment/nginx/huginn-ssl /etc/nginx/sites-available/huginn`
  - Update `YOUR_SERVER_FQDN`.
  - Update `ssl_certificate` and `ssl_certificate_key`.
  - Review the configuration file and consider applying other security and performance enhancing features.

Restart Nginx, export the init script and restart Huginn:

```
cd /home/huginn/huginn
sudo service nginx restart
sudo bundle exec rake production:export
```

Using a self-signed certificate is discouraged, but if you must use it follow the normal directions. Then generate the certificate:

```
sudo mkdir -p /etc/nginx/ssl/
cd /etc/nginx/ssl/
sudo openssl req -newkey rsa:2048 -x509 -nodes -days 3560 -out huginn.crt -keyout huginn.key
sudo chmod o-r huginn.key
```

Troubleshooting

If something went wrong during the installation please make sure you followed the instructions and did not miss a step.

When your Huginn instance still is not working first run the self check:

```
cd /home/huginn/huginn
sudo bundle exec rake production:check
```

We are sorry when you are still having issues, now please check the various log files for error messages:

Nginx error log `/var/log/nginx/huginn_error.log`

This file should be empty, it is the first place to look because `nginx` is the first application handling the request your are sending to Huginn.

Common problems:

- `connect() to unix:/home/huginn/huginn/tmp/sockets/unicorn.sock et failed`: The Unicorn application server is not running, ensure you uncommented one of the example configuration below the `PRODUCTION` label in your Profile and the unicorn config file (`/home/huginn/huginn/config/unicorn.rb`) exists.
- `138 open() "/home/huginn/huginn/public/..." failed (13: Permission denied)`: The `/home/huginn/huginn/public` directory needs to be readable by the nginx user (which is per default `www-data`)

Unicorn log `/home/huginn/huginn/log/unicorn.log`

Should only contain HTTP request log entries like: `10.0.2.2 - - [18/Aug/2015:21:15:12 +0000] "GET / HTTP/1.0" 200 - 0.0110`

If you see ruby exception backtraces or other error messages the problem could be one of the following:

- The configuration file `/home/huginn/huginn/config/unicorn.rb` does not exist
- Gem dependencies where not installed

Rails Application log `/home/huginn/huginn/log/production.log`

This file is pretty verbose, you want to look at it if you are getting the `We're sorry, but something went wrong.` error message when using Huginn. This is an example backtrace that can help you or other huginn developers locate the issue:

```
NoMethodError (undefined method `name' for
nil:NilClass):
  app/controllers/jobs_controller.rb:6:in `index'
config/initializers/silence_worker_status_logger.rb:5
:in `call_with_silence_worker_status'
```

Runit/Background Worker logs

`/home/huginn/huginn/log/*/current`

Those files will contain error messages or backtraces if one of your agent is not performing as they should. The easiest way to debug an Agent is to watch all your log files for changes and trigger the agent to run via the Huginn web interface.

The log file location depends your `Procfile` configuration, this command will give you a list of the available logs:

```
ls -al /home/huginn/huginn/log/*/current
```

When you want to monitor the background processes you can easily watch all the files for changes:

```
tail -f /home/huginn/huginn/log/*/current
```

Still having problems? 😿

You probably found an error message or exception backtrace you could not resolve. Please create a new issue and include as much information as you could gather about the problem your are experiencing.

Update

You can also use Capistrano to keep your installation up to date.

0. Ensure depencies are up to date

```
cd /home/huginn/huginn
sudo bundle exec rake production:check
```

1. Stop server

```
sudo bundle exec rake production:stop
```

2. Store the current version

```
export OLD_VERSION=`git rev-parse HEAD`
```

3. Update the code

Back up changed files

```
sudo -u huginn -H cp Procfile Procfile.bak
```

Get the new code

```
sudo -u huginn -H git fetch --all
sudo -u huginn -H git checkout -- Procfile
sudo -u huginn -H git checkout master
sudo -u huginn -H git pull
```

Restore backed up files

```
sudo -u huginn -H cp Procfile.bak Procfile
```

4. Install gems, migrate and precompile assets

```
cd /home/huginn/huginn
sudo -u huginn -H bundle install --deployment --
without development test
# Run database migrations
sudo -u huginn -H bundle exec rake db:migrate
RAILS_ENV=production
# Clean up assets and cache
sudo -u huginn -H bundle exec rake assets:clean
assets:precompile tmp:cache:clear
RAILS_ENV=production
```

5. Update the Procfile

Check for changes made to the default `Procfile`

```
sudo -u huginn -H git diff $OLD_VERSION..master
Procfile
```

Update your `Procfile` if the default options of the version you are using changed

```
sudo -u huginn -H editor Procfile
```

6. Update the .env file

Check for changes made to the example `.env`

```
sudo -u huginn -H git diff
$OLD_VERSION..master .env.example
```

Update your `.env` with new options or changed defaults

```
sudo -u huginn -H editor .env
```

7. Export init script and start Huginn

```
# Export the init script
sudo bundle exec rake production:export
```

Deploy updates via Capistrano

After you followed the manual installation guide it is simple to push updates to your huginn instance using capistrano.

1. Ensure you have SSH access to your server via the huginn user

Either set a password for the huginn user or add your public SSH key:

```
# Set password
sudo passwd huginn
# Or add a SSH key
sudo -u huginn -H mkdir -p /home/huginn/.ssh
sudo -u huginn -H editor
/home/huginn/.ssh/authorized_keys
sudo -u huginn -H chmod -R 700 /home/huginn/.ssh
```

2. Configure Capistrano on your local machine

Add Capistrano configuration to you local `.env`:

```
CAPISTRANO_DEPLOY_SERVER=<IP or FQDN of your server>
CAPISTRANO_DEPLOY_USER=huginn
CAPISTRANO_DEPLOY_REPO_URL=https://github.com/cantino
/huginn.git
```

3. Run Capistrano

You can now run Capistrano and update your server:

```
cap production deploy
```

If you want to deploy a different branch, pass it as environment variable:

```
cap production deploy BRANCH=awesome-feature
```

Changes to remote .env and Procfile

If you want to change the `.env`, `Procfile` or `config/unicorn.rb` of your installation you still need to do it on your server, do not forget to export the init scripts after your are done:

```
cd /home/huginn/huginn
# Whichever you want to change
sudo -u huginn -H editor Procfile
```

```
sudo -u huginn -H editor .env
sudo -u huginn -H editor config/unicorn.rb
# Export init scripts and restart huginn
sudo rake production:export
```

Creating a new agent

Please note: Huginn's API is evolving and at some point Agents will likely be extracted into Ruby Gems. We'd very much like your input into how this should work and what should be changed in this API. See #60 and #293.

Huginn's Agents can create and receive events, and can be scheduled to run code at certain times or intervals. Creating a new Huginn Agent is not difficult, you simply create a new subclass of Agent which defines a set of required methods.

Agents are stored in `app/models/agents`, with RSpec specs in `spec/models/agents`.

Description

Use the `description` class method to set a Markdown description for your agent. For example:

```
description <<-MD
  The WeatherAgent creates an event for the following day's weather at `zipcode`.
 You must setup an API key for Wunderground in order to use this Agent.
MD
```

Options

Agents are configured with a JSON structure from the user, accessible raw via `options` and available with Liquid interpolation performed in `interpolated`. You should define a method called `default_options` that returns an example default configuration for your type of Agent. Additionally, you should define a method called `validate_options` that performs Rails validation on the contents of `options`, if any fields are required. You generally don't want to look at `interpolated` within `validate_options`. Here's an example of `default_options`:

```
def default_options
```

```
  { 'zipcode' => '94103' }
end

def validate_options
  errors.add(:base, 'zipcode is required') unless
options['zipcode'].present?
end
```

Scheduling

Agents can be scheduled to run at certain times, or on certain intervals. When a schedule is triggered, the `check` method in your Agent will be called. If your agent should be schedulable, use the `default_schedule` class method to declare a default, otherwise you should call `cannot_be_scheduled!`. Possible schedules are: `every_1m`, `every_2m`, `every_5m`, `every_10m`, `every_30m`, `every_1h`, `every_2h`, `every_5h`, `every_12h`, `every_1d`, `every_2d`, `every_7d`, `midnight`, `1am`, `2am`, `3am`, `4am`, `5am`, `6am`, `7am`, `8am`, `9am`, `10am`, `11am`, `noon`, `1pm`, `2pm`, `3pm`, `4pm`, `5pm`, `6pm`, `7pm`, `8pm`, `9pm`, `10pm`, and `11pm`

```
default_schedule "8pm"
def check
wunderground.forecast_for(interpolated['zipcode'])['f
orecast']['simpleforecast']['forecastday'].each do
|day|
    if is_tomorrow?(day)
      create_event :payload => day.merge('zipcode' =>
interpolated['zipcode'])
    end
  end
end
```

If your Agent creates events, as this example from the WeatherAgent does, then you should use the `event_description` class method to detail what data those events contain.

```
event_description <<-MD
  Events look like this:
```

```
      {
        'zipcode' => 12345,
        ...
        'maxhumidity' => 93,
        'minhumidity' => 63
      }
```
MD

Receiving Events

If your Agent can receive events, define a method called `receive` that accepts an array of incoming events. Otherwise, please annotate it with a call to `cannot_receive_events!`.

Creating Events

In code, your Agent can create events with `create_event :payload => { ... }`. If your Agent will never create events, please annotate it with a call to `cannot_create_events!`.

Memory

Agents have memory that can be used to maintain state between scheduled intervals or received events. It will be loaded and saved automatically for you and is available in `memory`.

Logging

Your Agent should create AgentLogs when interesting things happen, especially errors. Call `log` or `error` with a log message and, optionally, `:outbound_event` or `:inbound_event` to keep track of events tied to the log message.

Is it working?

It's nice to be able to tell the user if their instance of your Agent is working correctly. You should define a method called `working?` that returns true when everything seems good. Here's an example for an Agent that primarily creates events and has an `expected_update_period_in_days` option:

```
def working?
event_created_within?(interpolated['expected_update_p
eriod_in_days']) && !recent_error_logs?
```

```
end
```

And here is an example for an Agent that primarily receives events and has an `expected_receive_period_in_days` option:

```
def working?
  last_receive_at && last_receive_at >
interpolated['expected_receive_period_in_days'].to_i.
days.ago && !recent_error_logs?
end
```

You can, of course, write Agent-specific code in `working?`.

UI

Agents can have a custom UI by defining a show view at: `app/views/agents/agent_views/<agent name>/_show.html.erb`. If they need to have server-side functionality, you may POST data to the `handle_details_post_agent_path` and handle it with `handle_details_post` in your Agent. See the ManualEventAgent and its details view for an example.

Receiving Web Requests

Your Agent can receive web requests by implementing `receive_web_request`. The URL for your Agent will be something like `http://yourserver.com/users/:user_id/web_requests/:agent_id/:secret` where `:user_id` is a User's id, `:agent_id` is an Agent's id, and `:secret` is a token that should be user-specifiable in your Agent's configuration and checked by `receive_web_request`. It is highly recommended that every Agent verify this token whenever `receive_web_request` is called. For example, one of your Agent's options could be `secret` and you could compare this value to `params[:secret]` whenever `receive_web_request` is called on your Agent, rejecting invalid requests.

Your Agent's `receive_web_request` method should return an Array containing a response, a status code, and an optional MIME type. For example:

```
[{ status: "success" }, 200]
```

or

```
["not found", 404, 'text/plain']
```

Here is an example implementation of `receive_web_request`:

```
def receive_web_request(params, method, format)
  secret = params.delete('secret')
  return ["Please use POST requests only", 401] unless
method == "post"
  return ["Not Authorized", 401] unless secret ==
interpolated['secret']
 # do something with params here
  ['Done!', 200, 'text/plain']
end
```

If you need more parameters from the request object, you can also define `receive_web_request` like this:

```
def receive_web_request(request)
end
```

Please see the WebRequestsController for more documentation, as well as the implementations of `receive_web_request` in WebhookAgent and DataOutputAgent.

四、 參考文獻

https://www.dropbox.com/sh/rxzdoysy3kbzwvm/AAAuc239bjG3z6qLcgZm-tp4a?dl=0