Augur Documentation

Release v0.63.0 (Ides of March)

Augur Contributors

Apr 20, 2024

CONTENTS

1 Welcome! 1
- 1.1 Quickstart . . . 1
- 1.2 Explanations of Technologies . . . 19
- 1.3 Deployment . . . 21
- 1.4 Getting Started . . . 27
- 1.5 Development Guide . . . 47
- 1.6 REST API Documentation . . . 72
- 1.7 Docker . . . 103
- 1.8 Schema . . . 107
- 1.9 Augur OAuth Flow . . . 148

2 What is Augur? 153
- 2.1 Current maintainers . . . 153
- 2.2 Former maintainers . . . 153
- 2.3 Contributors . . . 154
- 2.4 GSoC 2022 participants . . . 154
- 2.5 GSoC 2021 participants . . . 154
- 2.6 GSoC 2020 participants . . . 154
- 2.7 GSoC 2019 participants . . . 155
- 2.8 GSoC 2018 participants . . . 155

HTTP Routing Table 157

CHAPTER

ONE

WELCOME!

1.1 Quickstart

Select installation instructions from those most closely related to the operating system that you use below. Note that Augur's dependencies do not consistently support python 3.11 at this time. Python 3.8 - Python 3.10 have been tested on each platform.

1.1.1 Ubuntu 22.x Setup

We default to this version of Ubuntu for the moment because Augur does not yet support python3.10, which is the default version of python3.x distributed with Ubuntu 22.0x.x

Git Platform Requirements (Things to have setup prior to initiating installation.)

1. Obtain a GitHub Access Token: https://github.com/settings/tokens
2. Obtain a GitLab Access Token: https://gitlab.com/-/profile/personal_access_tokens

Fork and Clone Augur

1. Fork https://github.com/chaoss/augur
2. Clone your fork. We recommend creating a `github` directory in your user's base directory.

Pre-Requisite Operating System Level Packages

Here we ensure your system is up to date, install required python libraries, install postgresql, and install our queuing infrastrucutre, which is composed of redis-server and rabbitmq-server

Executable

```
sudo apt update &&
sudo apt upgrade &&
sudo apt install software-properties-common &&
sudo apt install python3-dev &&
sudo apt install python3.10-venv &&
sudo apt install postgresql postgresql-contrib postgresql-client &&
sudo apt install build-essential &&
sudo apt install redis-server &&  # required
sudo apt install erlang && # required
sudo apt install rabbitmq-server && #required
sudo snap install go --classic && #required: Go Needs to be version 1.19.x or higher.␣
↪Snap is the package manager that gets you to the right version. Classic enables it to␣
↪actually be installed at the correct version.
sudo apt install nginx && # required for hosting
sudo add-apt-repository ppa:mozillateam/firefox-next &&
sudo apt install firefox=121.0~b7+build1-0ubuntu0.22.04.1 &&
sudo apt install firefox-geckodriver

# You will almost certainly need to reboot after this.
```

RabbitMQ Configuration

The default timeout for RabbitMQ needs to be set on Ubuntu 22.x.

```
sudo vi /etc/rabbitmq/advanced.config
```

Add this one line to that file (the period at the end matters):

```
[ {rabbit, [ {consumer_timeout, undefined} ]} ].
```

Git Configuration

There are some Git configuration parameters that help when you are cloning repos over time, and a platform prompts you for credentials when it finds a repo is deleted:

```
git config --global diff.renames true
git config --global diff.renameLimit 200000
git config --global credential.helper cache
git config --global credential.helper 'cache --timeout=9999999999999'
```

Postgresql Configuration

Create a PostgreSQL database for Augur to use

```
sudo su - &&
su - postgres &&
psql
```

Then, from within the resulting postgresql shell:

```
CREATE DATABASE augur;
CREATE USER augur WITH ENCRYPTED PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE augur TO augur;
```

If you're using PostgreSQL 15 or later, default database permissions will prevent Augur's installer from configuring the database. Add one last line after the above to fix this:

```
GRANT ALL ON SCHEMA public TO augur;
```

After that, return to your user by exiting `psql`

```
postgres=# \quit
```

Here we want to start an SSL connection to the `augur` database on port 5432:

```
psql -h localhost -U postgres -p 5432
```

Now type `exit` to log off the postgres user, and `exit` a SECOND time to log off the root user.

```
exit
exit
```

Rabbitmq Broker Configuration

You have to setup a specific user, and broker host for your augur instance. You can accomplish this by running the below commands:

```
sudo rabbitmq-plugins enable rabbitmq_management &&
sudo rabbitmqctl add_user augur password123 &&
sudo rabbitmqctl add_vhost augur_vhost &&
sudo rabbitmqctl set_user_tags augur augurTag administrator &&
sudo rabbitmqctl set_permissions -p augur_vhost augur ".*" ".*" ".*"
```

- We need rabbitmq_management so we can purge our own queues with an API call
- We need a user
- We need a vhost
- We then set permissions

NOTE: it is important to have a static hostname when using rabbitmq as it uses hostname to communicate with nodes.

RabbitMQ's server can then be started from systemd:

```
sudo systemctl start rabbitmq-server
```

If your setup of rabbitmq is successful your broker url should look like this:

broker_url = ``amqp://augur:password123@localhost:5672/augur_vhost``

RabbitMQ Developer Note:

These are the queues we create: - celery (the main queue) - secondary - scheduling

The endpoints to hit to purge queues on exit are:

```
curl -i -u augur:password123 -XDELETE http://localhost:15672/api/queues/AugurB/celery

curl -i -u augur:password123 -XDELETE http://localhost:15672/api/queues/AugurB/secondary

curl -i -u augur:password123 -XDELETE http://localhost:15672/api/queues/AugurB/scheduling
```

We provide this functionality to limit, as far as possible, the need for sudo privileges on the Augur operating system user. With sudo, you can accomplish the same thing with (Given a vhost named AugurB [case sensitive]):

1. To list the queues

```
sudo rabbitmqctl list_queues -p AugurB name messages consumers
```

2. To empty the queues, simply execute the command for your queues. Below are the 3 queues that Augur creates for you:

```
sudo rabbitmqctl purge_queue celery -p AugurB
sudo rabbitmqctl purge_queue secondary -p AugurB
sudo rabbitmqctl purge_queue scheduling -p AugurB
```

Where AugurB is the vhost. The management API at port 15672 will only exist if you have already installed the rabbitmq_management plugin.

During Augur installation, you will be prompted for this broker_url

Proxying Augur through Nginx

Assumes nginx is installed.

Then you create a file for the server you want Augur to run under in the location of your `sites-enabled` directory for nginx. In this example, Augur is running on port 5038: (the long timeouts on the settings page is for when a user adds a large number of repos or orgs in a single session to prevent timeouts from nginx)

```
server {
        server_name  ai.chaoss.io;

        location /api/unstable/ {
                proxy_pass http://ai.chaoss.io:5038;
                proxy_set_header Host $host;
        }

        location / {
                proxy_pass http://127.0.0.1:5038;
        }
```

(continues on next page)

(continued from previous page)

```
        location /settings {

                proxy_read_timeout 800;
                proxy_connect_timeout 800;
                proxy_send_timeout 800;
        }

        error_log /var/log/nginx/augurview.osshealth.error.log;
        access_log /var/log/nginx/augurview.osshealth.access.log;

}
```

Setting up SSL (https)

Install Certbot:

```
sudo apt update &&
sudo apt upgrade &&
sudo apt install certbot &&
sudo apt-get install python3-certbot-nginx
```

Generate a certificate for the specific domain for which you have a file already in the sites-enabled directory for nginx (located at `/etc/nginx/sites-enabled` on Ubuntu):

```
sudo certbot -v --nginx  -d ai.chaoss.io
```

In the example file above. Your resulting nginx sites-enabled file will look like this:

```
server {
        server_name  ai.chaoss.io;

        location /api/unstable/ {
                proxy_pass http://ai.chaoss.io:5038;
                proxy_set_header Host $host;
        }

   location / {
      proxy_pass http://127.0.0.1:5038;
   }

   location /settings {

                proxy_read_timeout 800;
                proxy_connect_timeout 800;
                proxy_send_timeout 800;
   }

        error_log /var/log/nginx/augurview.osshealth.error.log;
        access_log /var/log/nginx/augurview.osshealth.access.log;

    listen 443 ssl; # managed by Certbot
```

(continues on next page)

(continued from previous page)

```
    ssl_certificate /etc/letsencrypt/live/ai.chaoss.io/fullchain.pem; # managed by␣
↪Certbot
    ssl_certificate_key /etc/letsencrypt/live/ai.chaoss.io/privkey.pem; # managed by␣
↪Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {
    if ($host = ai.chaoss.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80;
        server_name  ai.chaoss.io;
    return 404; # managed by Certbot


}
```

Installing and Configuring Augur!

Create a Python Virtual Environment `python3 -m venv ~/virtual-env-directory`

Activate your Python Virtual Environment `source ~/virtual-env-directory/bin/activate`

From the root of the Augur Directory, type `make install`. You will be prompted to provide:

- "User" is the PSQL database user, which is `augur` if you followed instructions exactly
- "Password" is the above user's password
- "Host" is the domain used with nginx, e.g. `ai.chaoss.io`
- "Port" is 5432 unless you reconfigured something
- "Database" is the name of the Augur database, which is `augur` if you followed instructions exactly
- The GitHub token created earlier
- Then the username associated with it
- Then the same for GitLab
- and finally a directory to clone repositories to

Post Installation of Augur

Redis Broker Configuration

If applications other than Augur are running on the same server, and using `redis-server` it is important to ensure that Augur and these other applications (or additional instances of Augur) are using distinct "cache_group". You can change from the default value of zero by editing the `augur_operations.config` table directly, looking for the "Redis" section_name, and the "cache_group" setting_name. This SQL is also a template:

```
UPDATE augur_operations.config
SET value = 2
WHERE
section_name='Redis'
AND
setting_name='cache_group';
```

What does Redis Do?

Redis is used to make the state of data collection jobs visible on an external dashboard, like Flower. Internally, Augur relies on Redis to cache GitHub API Keys, and for OAuth Authentication. Redis is used to maintain awareness of Augur's internal state.

What does RabbitMQ Do?

Augur is a distributed system. Even on one server, there are many collection processes happening simultaneously. Each job to collect data is put on the RabbitMQ Queue by Augur's "Main Brain". Then independent workers pop messages off the RabbitMQ Queue and go collect the data. These tasks then become standalone processes that report their completion or failure states back to the Redis server.

Edit the `/etc/redis/redis.conf` file to ensure these parameters are configured in this way:

```
supervised systemd
databases 900
maxmemory-samples 10
maxmemory 20GB
```

NOTE: You may be able to have fewer databases and lower maxmemory settings. This is a function of how many repositories you are collecting data for at a given time. The more repositories you are managing data for, the close to these settings you will need to be.

Consequences : If the settings are too low for Redis, Augur's maintainer team has observed cases where collection appears to stall. (TEAM: This is a working theory as of 3/10/2023 for Ubuntu 22.x, based on EC2 experiments.)

Possible EC2 Configuration Requirements

With virtualization there may be issues associated with redis-server connections exceeding available memory. In these cases, the following workarounds help to resolve issues.

Specifically, you may find this error in your augur logs:

```
redis.exceptions.ConnectionError: Error 111 connecting to 127.0.0.1:6379. Connection␣
↪refused.
```

INSTALL `sudo apt install libhugetlbfs-bin`

COMMAND:

```
sudo hugeadm --thp-never &&
sudo echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

```
sudo vi /etc/rc.local
```

paste into `/etc/rc.local`

```
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
   echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
```

EDIT : `/etc/default/grub` add the following line:

```
GRUB_DISABLE_OS_PROBER=true
```

Postgresql Configuration

Your postgresql instance should optimally allow 1,000 connections:

```
max_connections = 1000                  # (change requires restart)
shared_buffers = 8GB                    # min 128kB
work_mem = 2GB                  # min 64kB
```

Augur will generally hold up to 150 simultaneous connections while collecting data. The 1,000 number is recommended to accommodate both collection and analysis on the same database. Use of PGBouncer or other utility may change these characteristics.

Augur Commands

To access command line options, use `augur --help`. To load repos from GitHub organizations prior to collection, or in other ways, the direct route is `augur db --help`.

Start a Flower Dashboard, which you can use to monitor progress, and report any failed processes as issues on the Augur GitHub site. The error rate for tasks is currently 0.04%, and most errors involve unhandled platform API timeouts. We continue to identify and add fixes to handle these errors through additional retries. Starting Flower: `(nohup celery -A augur.tasks.init.celery_app.celery_app flower --port=8400 --max-tasks=1000000 &)` NOTE: You can use any open port on your server, and access the dashboard in a browser with http://servername-or-ip:8400 in the example above (assuming you have access to that port, and its open on your network.)

If you're using a virtual machine within Windows and you get an error about missing AVX instructions, you should kill Hyper-V. Even if it doesn't *appear* to be active, it might still be affecting your VM. Follow these instructions to disable Hyper-V, and afterward AVX should pass to the VM.

Starting your Augur Instance

Start Augur: `(nohup augur backend start &)`

When data collection is complete you will see only a single task running in your flower Dashboard.

Accessing Repo Addition and Visualization Front End

Your Augur instance will now be available at http://hostname.io:port_number

For example: http://chaoss.tv:5038

Note: Augur will run on port 5000 by default (you probably need to change that in augur_operations.config for OSX)

Stopping your Augur Instance

You can stop augur with `augur backend stop`, followed by `augur backend kill`. We recommend waiting 5 minutes between commands so Augur can shutdown more gently. There is no issue with data integrity if you issue them seconds apart, its just that stopping is nicer than killing.

Docker

1. Make sure docker, and docker compose are both installed
2. Modify the `environment.txt` file in the root of the repository to include your GitHub and GitLab API keys.
3. If you are already running postgresql on your server you have two choices:
    - Change the port mappings in the `docker-compose.yml` file to match ports for Postgresql not currently in use.
    - Change to variables in `environment.txt` to include the correct values for your local, non-docker-container database.
4. `sudo docker build -t augur-new -f docker/backend/Dockerfile .`
5. `sudo docker compose --env-file ./environment.txt --file docker-compose.yml up` to run the database in a Docker Container or `sudo docker compose --env-file ./environment.txt --file docker-compose.yml up` to connect to an already running database.

1.1.2 OSX Setup

NOTE: Currently, our machine learning dependencies allow Augur to only fully support python 3.8 to python 3.10. Python 3.11 will sometimes work, but often there are libraries at the operating system level that have not yet been updated to support machine learning libraries at python 3.11.

For OSX You Need to make sure to install XCode Command line tools:

```
xcode-select --install
```

WARNING: rabbitmq, redis, and postgresql will, by default, set themselves up to automatically start when your computer starts. This can be a significant battery drain if you are on battery and not using Augur. For those reasons, go into your system preferences, startup items menu (wherever it is now, because Apple changes it more than Zoolander changes outfits), and turn those "autostart" options off. :)

NOTE: If you do not shutoff rabbitmq and redis at the command line before shutting down, they will restart themselves anyway on restart. ``brew services stop rabbitmq ;brew services stop redis;``

You also need to install these libraries if you are using apple silicon as of June, 2023

brew install gfortran; brew install llvm; echo 'export PATH="/opt/homebrew/opt/llvm/bin:$PATH"' >> ~/.zshrc; brew install Pkg-config; brew install openblas;

Add these lines to your .zshrc file:

```
export LDFLAGS="-L/opt/homebrew/opt/llvm/lib"
export CPPFLAGS="-I/opt/homebrew/opt/llvm/include"
export LDFLAGS="-L/opt/homebrew/opt/openblas/lib "$LDFLAGS
export CPPFLAGS="-I/opt/homebrew/opt/openblas/include "$CPPFLAGS
export PKG_CONFIG_PATH="/opt/homebrew/opt/openblas/lib/pkgconfig"
```

Git Platform Requirements (Things to have setup prior to initiating installation.)

1. Obtain a GitHub Access Token: https://github.com/settings/tokens
2. Obtain a GitLab Access Token: https://gitlab.com/-/profile/personal_access_tokens

Fork and Clone Augur

1. Fork https://github.com/chaoss/augur
2. Clone your fork. We recommend creating a `github` directory in your user's base directory.

Pre-Requisite Operating System Level Packages

Here we ensure your system is up to date, install required python libraries, install postgresql, and install our queuing infrastrucutre, which is composed of redis-server and rabbitmq-server

Updating your Path: Necessary for rabbitmq on OSX

for macOS Intel

`export PATH=$PATH:/usr/local/sbin` #### for Apple Silicon `export PATH=$PATH:/opt/homebrew/sbin`

These should be added to your .zshrc or other environment file loaded when you open a terminal

for macOS Intel

`export PATH=$PATH:/usr/local/sbin:$PATH` #### for Apple Silicon `export PATH=$PATH:/opt/homebrew/sbin:$PATH`

Executable

```
brew update ;
brew upgrade ;
brew install rabbitmq ;
brew install redis ;
brew install postgresql@14 ;
brew install python3-yq ;
brew install python@3.11 ;
brew install postgresql@14 ;
brew install go ; #required: Go Needs to be version 1.19.x or higher.
brew install nginx ; # required for hosting
brew install geckodriver;

# You will almost certainly need to reboot after this.
```

RabbitMQ Configuration

The default timeout for RabbitMQ needs to be set.

```
sudo vi /opt/homebrew/etc/rabbitmq/advanced.config
```

Add this one line to that file (the period at the end matters):

```
[ {rabbit, [ {consumer_timeout, undefined} ]} ].
```

Rabbitmq Broker Configuration

You have to setup a specific user, and broker host for your augur instance. You can accomplish this by running the below commands:

```
rabbitmq-plugins enable rabbitmq_management;
rabbitmqctl add_user augur password123;
rabbitmqctl add_vhost augur_vhost;
rabbitmqctl set_user_tags augur augurTag administrator;
rabbitmqctl set_permissions -p augur_vhost augur ".*" ".*" ".*";
```

- We need rabbitmq_management so we can purge our own queues with an API call
- We need a user
- We need a vhost
- We then set permissions

NOTE: it is important to have a static hostname when using rabbitmq as it uses hostname to communicate with nodes.

If your setup of rabbitmq is successful your broker url should look like this:

broker_url = ``amqp://augur:password123@localhost:5672/augur_vhost``

You will be asked for the broker URL on install of Augur. You can copy and paste the line above (`amqp://augur:password123@localhost:5672/augur_vhost`) if you created the users and virtual hosts under "Broker Configuration", above.

Things to start before augur later

```
brew services start rabbitmq ;
brew services start redis;
brew services start postgresql@14;
```

If Issues Starting rabbitmq

If you get this error:

```
brew services start rabbitmq
Bootstrap failed: 5: Input/output error
Try re-running the command as root for richer errors.
Error: Failure while executing; `/bin/launchctl bootstrap gui/501 /Users/sean/Library/
↪LaunchAgents/homebrew.mxcl.rabbitmq.plist` exited with 5.
```

Execute this command:

```
launchctl unload -w /Users/sean/Library/LaunchAgents/homebrew.mxcl.rabbitmq.plist
```

Replace the specific path with the one after `/Users/sean/Library/LaunchAgents/` in your error message. This was tested on Apple Silicon.

Git Configuration

There are some Git configuration parameters that help when you are cloning repos over time, and a platform prompts you for credentials when it finds a repo is deleted:

```
git config --global diff.renames true;
git config --global diff.renameLimit 200000;
git config --global credential.helper cache;
git config --global credential.helper 'cache --timeout=9999999999999';
```

Postgresql Configuration

Create a PostgreSQL database for Augur to use

```
... really varies depending how you installed postgres. TBD
```

Then, from within the resulting postgresql shell:

```
CREATE DATABASE augur;
CREATE USER augur WITH ENCRYPTED PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE augur TO augur;
GRANT ALL ON SCHEMA public TO augur;
```

After that, return to your user by exiting `psql`

```
postgres=# \quit
```

Here we want to start an SSL connection to the `augur` database on port 5432:

```
psql -h localhost -U postgres -p 5432
```

Now type `exit` to log off the postgres user, and `exit` a SECOND time to log off the root user.

```
exit
exit
```

RabbitMQ Developer Note:

These are the queues we create: - celery (the main queue) - secondary - scheduling

The endpoints to hit to purge queues on exit are:

```
curl -i -u augur:password123 -XDELETE http://localhost:15672/api/queues/augur_vhost/
↪celery

curl -i -u augur:password123 -XDELETE http://localhost:15672/api/queues/augur_vhost/
↪secondary

curl -i -u augur:password123 -XDELETE http://localhost:15672/api/queues/augur_vhost/
↪scheduling
```

We provide this functionality to limit, as far as possible, the need for sudo privileges on the Augur operating system user. With sudo, you can accomplish the same thing with (Given a vhost named AugurB [case sensitive]):

1. To list the queues

```
rabbitmqctl list_queues -p AugurB name messages consumers
```

2. To empty the queues, simply execute the command for your queues. Below are the 3 queues that Augur creates for you:

```
rabbitmqctl purge_queue celery -p augur_vhost
rabbitmqctl purge_queue secondary -p augur_vhost
rabbitmqctl purge_queue scheduling -p augur_vhost
```

Where augur_vhost is the vhost. The management API at port 15672 will only exist if you have already installed the rabbitmq_management plugin.

During Augur installation, you will be prompted for this broker_url

Installing and Configuring Augur!

Create a Python Virtual Environment `python3 -m venv ~/virtual-env-directory`

Activate your Python Virtual Environment `source ~/virtual-env-directory/bin/activate`

From the root of the Augur Directory, type `make install`. You will be prompted to provide:

- "User" is the PSQL database user, which is `augur` if you followed instructions exactly
- "Password" is the above user's password
- "Host" is the domain used with nginx, e.g. `ai.chaoss.io`
- "Port" is 5432 unless you reconfigured something
- "Database" is the name of the Augur database, which is `augur` if you followed instructions exactly
- The GitHub token created earlier
- Then the username associated with it
- Then the same for GitLab
- and finally a directory to clone repositories to

Post Installation of Augur

Redis Broker Configuration

If applications other than Augur are running on the same server, and using `redis-server` it is important to ensure that Augur and these other applications (or additional instances of Augur) are using distinct "cache_group". You can change from the default value of zero by editing the `augur_operations.config` table directly, looking for the "Redis" section_name, and the "cache_group" setting_name. This SQL is also a template:

```
UPDATE augur_operations.config
SET value = 2
WHERE
section_name='Redis'
AND
setting_name='cache_group';
```

What does Redis Do?

Redis is used to make the state of data collection jobs visible on an external dashboard, like Flower. Internally, Augur relies on Redis to cache GitHub API Keys, and for OAuth Authentication. Redis is used to maintain awareness of Augur's internal state.

What does RabbitMQ Do?

Augur is a distributed system. Even on one server, there are many collection processes happening simultaneously. Each job to collect data is put on the RabbitMQ Queue by Augur's "Main Brain". Then independent workers pop messages off the RabbitMQ Queue and go collect the data. These tasks then become standalone processes that report their completion or failure states back to the Redis server.

Edit the `/etc/redis/redis.conf` file to ensure these parameters are configured in this way:

```
supervised systemd
databases 900
maxmemory-samples 10
maxmemory 20GB
```

NOTE: You may be able to have fewer databases and lower maxmemory settings. This is a function of how many repositories you are collecting data for at a given time. The more repositories you are managing data for, the close to these settings you will need to be.

Consequences : If the settings are too low for Redis, Augur's maintainer team has observed cases where collection appears to stall. (TEAM: This is a working theory as of 3/10/2023 for Ubuntu 22.x, based on EC2 experiments.)

(OPTIONAL: NOT FOR DEV: Proxying Augur through Nginx)

Assumes nginx is installed.

Then you create a file for the server you want Augur to run under in the location of your `sites-enabled` directory for nginx. In this example, Augur is running on port 5038: (the long timeouts on the settings page is for when a user adds a large number of repos or orgs in a single session to prevent timeouts from nginx)

For MacOS Intel:

This gist explains where sites-enabled is: `https://gist.github.com/jimothyGator/5436538`

Logs for nginx should go in this directory:

`mkdir /Library/Logs/nginx`

`/Library/Logs/nginx`

For Apple Silicon:

There is no `sites-enabled` directory. Server configurations go here: `/opt/homebrew/etc/nginx/servers`

Logs for nginx should go in this directory: `/opt/homebrew/var/log/nginx`

```
server {
        server_name  ai.chaoss.io;

        location /api/unstable/ {
                proxy_pass http://ai.chaoss.io:5038;
                proxy_set_header Host $host;
        }

        location / {
```

(continues on next page)

(continued from previous page)

```
                proxy_pass http://127.0.0.1:5038;
        }

        location /settings {

                proxy_read_timeout 800;
                proxy_connect_timeout 800;
                proxy_send_timeout 800;
        }

        error_log /var/log/nginx/augurview.osshealth.error.log;
        access_log /var/log/nginx/augurview.osshealth.access.log;

}
```

(OPTIONAL: NOT FOR DEV) Setting up SSL (https)

Install Certbot: **NOTE: certbot does not currently run on Apple Silicon, as it is looking for information in MacOS Intel directories**

```
brew update;
brew upgrade;
brew install certbot;
brew install openssl;
brew install brew install python-typing-extensions
```

Generate a certificate for the specific domain for which you have a file already in the sites-enabled directory for nginx (located at `/etc/nginx/sites-enabled` on Ubuntu):

```
brew certbot -v --nginx  -d ai.chaoss.io
```

In the example file above. Your resulting nginx sites-enabled file will look like this:

```
server {
        server_name  ai.chaoss.io;

        location /api/unstable/ {
                proxy_pass http://ai.chaoss.io:5038;
                proxy_set_header Host $host;
        }

   location / {
      proxy_pass http://127.0.0.1:5038;
   }

   location /settings {

                proxy_read_timeout 800;
                proxy_connect_timeout 800;
                proxy_send_timeout 800;
   }
```

(continues on next page)

(continued from previous page)

```
        error_log /var/log/nginx/augurview.osshealth.error.log;
        access_log /var/log/nginx/augurview.osshealth.access.log;

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/ai.chaoss.io/fullchain.pem; # managed by␣
↪Certbot
    ssl_certificate_key /etc/letsencrypt/live/ai.chaoss.io/privkey.pem; # managed by␣
↪Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {
    if ($host = ai.chaoss.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80;
        server_name  ai.chaoss.io;
    return 404; # managed by Certbot


}
```

Possible EC2 Configuration Requirements

With virtualization there may be issues associated with redis-server connections exceeding available memory. In these cases, the following workarounds help to resolve issues.

Specifically, you may find this error in your augur logs:

```
redis.exceptions.ConnectionError: Error 111 connecting to 127.0.0.1:6379. Connection␣
↪refused.
```

INSTALL `sudo apt install libhugetlbfs-bin`

COMMAND:

```
sudo hugeadm --thp-never &&
sudo echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

```
sudo vi /etc/rc.local
```

paste into `/etc/rc.local`

```
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
   echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
```

EDIT : `/etc/default/grub` add the following line:

```
GRUB_DISABLE_OS_PROBER=true
```

Postgresql Configuration

Your postgresql instance should optimally allow 1,000 connections:

```
max_connections = 1000                  # (change requires restart)
shared_buffers = 8GB                    # min 128kB
work_mem = 2GB                  # min 64kB
```

Augur will generally hold up to 150 simultaneous connections while collecting data. The 1,000 number is recommended to accommodate both collection and analysis on the same database. Use of PGBouncer or other utility may change these characteristics.

Augur Commands

To access command line options, use `augur --help`. To load repos from GitHub organizations prior to collection, or in other ways, the direct route is `augur db --help`.

Start a Flower Dashboard, which you can use to monitor progress, and report any failed processes as issues on the Augur GitHub site. The error rate for tasks is currently 0.04%, and most errors involve unhandled platform API timeouts. We continue to identify and add fixes to handle these errors through additional retries. Starting Flower: `(nohup celery -A augur.tasks.init.celery_app.celery_app flower --port=8400 --max-tasks=1000000 &)` NOTE: You can use any open port on your server, and access the dashboard in a browser with http://servername-or-ip:8400 in the example above (assuming you have access to that port, and its open on your network.)

If you're using a virtual machine within Windows and you get an error about missing AVX instructions, you should kill Hyper-V. Even if it doesn't *appear* to be active, it might still be affecting your VM. Follow these instructions to disable Hyper-V, and afterward AVX should pass to the VM.

Starting your Augur Instance

Start Augur: `(nohup augur backend start &)`

When data collection is complete you will see only a single task running in your flower Dashboard.

Accessing Repo Addition and Visualization Front End

Your Augur instance will now be available at http://hostname.io:port_number

For example: http://chaoss.tv:5038

Note: Augur will run on port 5000 by default (you probably need to change that in augur_operations.config for OSX)

Stopping your Augur Instance

You can stop augur with `augur backend stop`, followed by `augur backend kill`. We recommend waiting 5 minutes between commands so Augur can shutdown more gently. There is no issue with data integrity if you issue them seconds apart, its just that stopping is nicer than killing.

Docker

1. Make sure docker, and docker compose are both installed
2. Modify the `environment.txt` file in the root of the repository to include your GitHub and GitLab API keys.
3. If you are already running postgresql on your server you have two choices:
   - Change the port mappings in the `docker-compose.yml` file to match ports for Postgresql not currently in use.
   - Change to variables in `environment.txt` to include the correct values for your local, non-docker-container database.
4. `sudo docker build -t augur-new -f docker/backend/Dockerfile .`
5. `sudo docker compose --env-file ./environment.txt --file docker-compose.yml up` to run the database in a Docker Container or `sudo docker compose --env-file ./environment.txt --file docker-compose.yml up` to connect to an already running database.

1.2 Explanations of Technologies

1.2.1 What does Redis Do?

Redis is used to make the state of data collection jobs visible on an external dashboard, like Flower. Internally, Augur relies on Redis to cache GitHub API Keys, and for OAuth Authentication. Redis is used to maintain awareness of Augur's internal state.

1.2.2 What does RabbitMQ Do?

Augur is a distributed system. Even on one server, there are many collection processes happening simultaneously. Each job to collect data is put on the RabbitMQ Queue by Augur's "Main Brain". Then independent workers pop messages off the RabbitMQ Queue and go collect the data. These tasks then become standalone processes that report their completion or failure states back to the Redis server.

Edit the `/etc/redis/redis.conf` file to ensure these parameters are configured in this way:

```
supervised systemd
databases 900
maxmemory-samples 10
maxmemory 20GB
```

NOTE: You may be able to have fewer databases and lower maxmemory settings. This is a function of how many repositories you are collecting data for at a given time. The more repositories you are managing data for, the close to these settings you will need to be.

Consequences : If the settings are too low for Redis, Augur's maintainer team has observed cases where collection appears to stall. (TEAM: This is a working theory as of 3/10/2023 for Ubuntu 22.x, based on EC2 experiments.)

1.2.3 Possible EC2 Configuration Requirements

With virtualization there may be issues associated with redis-server connections exceeding available memory. In these cases, the following workarounds help to resolve issues.

Specifically, you may find this error in your augur logs:

```
redis.exceptions.ConnectionError: Error 111 connecting to 127.0.0.1:6379. Connection␣
↪refused.
```

INSTALL `sudo apt install libhugetlbfs-bin`

COMMAND:

```
hugeadm --thp-never` &&
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

```
sudo vi /etc/rc.local
```

paste into `/etc/rc.local`

```
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
   echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
```

EDIT : `/etc/default/grub` add the following line:

```
GRUB_DISABLE_OS_PROBER=true
```

Postgresql Configuration

Your postgresql instance should optimally allow 1,000 connections:

```
max_connections = 1000                  # (change requires restart)
shared_buffers = 8GB                    # min 128kB
work_mem = 2GB                  # min 64kB
```

Augur will generally hold up to 150 simultaneous connections while collecting data. The 1,000 number is recommended to accommodate both collection and analysis on the same database. Use of PGBouncer or other utility may change these characteristics.

Augur Commands

To access command line options, use `augur --help`. To load repos from GitHub organizations prior to collection, or in other ways, the direct route is `augur db --help`.

Start a Flower Dashboard, which you can use to monitor progress, and report any failed processes as issues on the Augur GitHub site. The error rate for tasks is currently 0.04%, and most errors involve unhandled platform API timeouts. We continue to identify and add fixes to handle these errors through additional retries. Starting Flower: `(nohup celery -A augur.tasks.init.celery_app.celery_app flower --port=8400 --max-tasks=1000000 &)` NOTE: You can use any open port on your server, and access the dashboard in a browser with http://servername-or-ip:8400 in the example above (assuming you have access to that port, and its open on your network.)

Starting your Augur Instance

Start Augur: `(nohup augur backend start &)`

When data collection is complete you will see only a single task running in your flower Dashboard.

Accessing Repo Addition and Visualization Front End

Your Augur instance will now be available at http://hostname.io:port_number

For example: http://chaoss.tv:5038

Note: Augur will run on port 5000 by default (you probably need to change that in augur_operations.config for OSX)

Stopping your Augur Instance

You can stop augur with `augur backend stop`, followed by `augur backend kill`. We recommend waiting 5 minutes between commands so Augur can shutdown more gently. There is no issue with data integrity if you issue them seconds apart, its just that stopping is nicer than killing.

Docker

1. Make sure docker, and docker compose are both installed
2. Modify the `environment.txt` file in the root of the repository to include your GitHub and GitLab API keys.
3. If you are already running postgresql on your server you have two choices:
   - Change the port mappings in the `docker-compose.yml` file to match ports for Postgresql not currently in use.
   - Change to variables in `environment.txt` to include the correct values for your local, non-docker-container database.
4. `sudo docker build -t augur-new -f docker/backend/Dockerfile .`
5. `sudo docker compose --env-file ./environment.txt --file docker-compose.yml up` to run the database in a Docker Container or `sudo docker compose --env-file ./environment.txt --file docker-compose.yml up` to connect to an already running database.

1.3 Deployment

This section details describes production deployment of Augur.

1.3.1 Setting up an Augur Server

High-Level Steps to Server Installation of Augur

1. Have a list of repositories and groups you want them to be in, following the format in the files *schema/repo_group_load_sample.csv* and *schema/repo_load_sample.csv*.
2. Have access to a server that meets the augur installation pre-requisites (Python, NodeJS, etc).
3. Have nginx installed for front-end service. You can use another HTTP server, but we have instructions for nginx.
4. Make sure you have a database available, owned by a user who has the right to create tables.
5. Have a GitHub API Key handy.

Detailed Steps

1. Login to your server.
2. Create or activate the Python3 virtual environment for the Augur instance you want to deploy as a public server.
3. If you have not already done so, clone Augur.
4. Change into that directory.
5. `git checkout dev`, if you want to deploy the latest features, otherwise remain in the main branch.
6. You need a database owned by an Augur user.
7. `make install` ... now is a good time to go get some tea.
8. **When you return with your tea, follow the prompts:**
    - respond to the SERVER prompt with localhost.
    - the current standard is to put the repos in a repos/ directory in the root augur directory (these will never get checked in to VC).
9. Load repos, following instructions in docs.
10. **If you have more than one instance of Augur or another service on port 5000, you need to edit the `augur.config.json` to update the server port:**
    - `sudo lsof -i -P -n | grep LISTEN` shows you ports in use if you are not sure.
    - `sudo ufw status` lets you know if the port you are looking for is available as open through your firewall.
    - `sudo ufw status | grep 5005` checks to see if port 5005 is open, for example:

5005	ALLOW	Anywhere (this line is the most important)
5005/tcp	ALLOW	Anywhere
5005/udp	ALLOW	Anywhere
5005 (v6)	ALLOW	Anywhere (v6)
5005/tcp (v6)	ALLOW	Anywhere (v6)
5005/udp (v6)	ALLOW	Anywhere (v6)

11. In the Frontend block of `augur.config.json`, set the host value to be the domain you want the front end compiled for. For example, we set ours to `test.augurlabs.io`.

```
{
"Frontend": {
        "host": "test.augurlabs.io",
        "port": "5003"
    }
}
```

12. Then do a `make rebuild-dev`.

Next up: configure `nginx`!

1.3.2 Web Server Configuration

Configuring nginx for Augur to run behind nginx requires you to have certain options available for symlinks and other basic nginx options. The *nginx.conf* file below is one example of a configuration known to work.

Once you have nginx configured, run these commands to make sure everything is loaded and configured correctly:

1. **`sudo nginx -t` to make sure it's configured correctly.**
   nginx: the configuration file /etc/nginx/nginx.conf syntax is ok nginx: configuration file /etc/nginx/nginx.conf test is successful
2. `sudo systemctl restart nginx` on Ubuntu.
3. sudo nginx on OS X.

Server Compilation

Your Augur instance must compile with a publicly accessible domain that the frontend instance will be able to access.

1. Your `augur.config.json` server block **must** be set like this:

```
    {
        "Server": {
            "cache_expire": 3600,
            "host": "subdomain.domain.tld",
            "port": "5000",
            "workers": 8
        }
}
```

2. Compile Augur (this wires the host and port into the frontend so people pulling the web pages of Augur, in the *frontend/* subdirectory are referring to the right endpoints for this instance.): `make rebuild`
3. Run Augur: `nohup augur backend start >augur.log 2>augur.err &`

nginx

nginx.conf

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

worker_rlimit_nofile 30000;

events {
        worker_connections 768;
        # multi_accept on;
}

http {

        ##
        # Basic Settings
        ##
        disable_symlinks off;

        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        # server_tokens off;

         server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
        ssl_prefer_server_ciphers on;

        ##
        # Logging Settings
        ##

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        ##
        # Gzip Settings
        ##
```

(continues on next page)

(continued from previous page)

```
        gzip on;

        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json application/javascript text/
↪xml application/xml application/xml+rss text/javascript;

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

Site Configuration

This file will be located in the `/etc/nginx/sites-enabled` directory on most Linux distributions. Mac OSX keeps these files in the `/usr/local/etc/nginx/sites-enabled` directory. **Note that Augur's backend server must be running**

```
server {
        listen 80;
        server_name  <<your server subdomain.domain.tld>>;

        root /home/user/.../<<augur-instance-home>>/frontend/dist;
        index index.html index.htm;

        location / {
        root /home/user/.../<<augur-instance-home>>/frontend/dist;
        try_files $uri $uri/ /index.html;
        }

#        location /api/unstable/ {
#                proxy_pass http://census.osshealth.io:5000;
#                proxy_set_header Host $host;
#        }

        location /api_docs/ {
        root /home/user/.../<<augur-instance-home>>/frontend/dist;
        index index.html;
        }


        error_log /var/log/nginx/augur.censusscienceosshealth.error.log;
        access_log /var/log/nginx/augur.censusscienceosshealth.access.log;
```

(continues on next page)

(continued from previous page)

```
}
```

Enabling HTTPS

HTTPS is an extension of HTTP. It is used for secure communications over a computer networks by encrypting your data so it is not vulnerable to MIM(Man-in-the-Middle) attacks etc. While Augur's API data might not be very sensitive, it would still be a nice feature to have so something can't interfere and provide wrong data. Additionally, the user may not feel very comfortable using an application when the browser is telling the user it is not secure. Features such as logins is an example of information that would be particularly vulnerable to attacks. Lastly, search engine optimization actually favors applications on HTTPS over HTTP.

This guide will start on a fully configured EC2 Ubuntu 20.04 instance, meaning it is assumed to already have Augur installed and running with all of its dependencies(PostgreSQL, Nginx, etc).

Let's Encrypt/Certbot

The easiest way to get an HTTPS server up is to make use of Let's Encrypt's Certbot tool. It is an open source tool that is so good and it will even alter the nginx configuration for you automatically to enable HTTPS. Following their guide for `Ubuntu 20.04`, run `sudo snap install --classic certbot`, `sudo ln -s /snap/bin/certbot /usr/bin/certbot`, and then `sudo certbot --nginx`.

```
# Example Certificate Response Using Certbot

Which names would you like to activate HTTPS for?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1: augur.augurlabs.io
2: new.augurlabs.io
3: old.augurlabs.io
4: augur.chaoss.io
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel): 4
Requesting a certificate for augur.chaoss.io

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/augur.chaoss.io/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/augur.chaoss.io/privkey.pem
This certificate expires on 2022-07-12.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the␣
↪background.

Deploying certificate
Successfully deployed certificate for augur.chaoss.io to /etc/nginx/sites-enabled/augur.
↪chaoss.io
Congratulations! You have successfully enabled HTTPS on https://augur.chaoss.io

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
 * Donating to EFF:                    https://eff.org/donate-le
```

Fixing the Backend

Now our server is configured properly and our frontend is being served over HTTPS, but there's an extra problem: the backend APIs are still being served over HTTP resulting in a `blocked loading mixed active content` error. This issue is a deep rooted issue and several files need to be modified to accomodate HTTPS.

First, we will start with lines 29, 33, & 207 of `augur/frontend/src/AugurAPI.ts` and rewrite the URL to use the HTTPS protocol instead of HTTP. We will then do this again in `augur/frontend/src/common/index.tx` & `augur/frontend/src/compare/index.ts` where the `AugurAPI` constructor was called and passed an HTTP protocol. Next we need to configure gunicorn in the backend to support our SSL certificates, but by default certbot places these in a directory that requires root access. Copy these files by running `sudo cp /etc/letsencrypt/live/<server name here>/fullchain.pem /home/ubuntu/augur/fullchain.pem` and `sudo cp /etc/letsencrypt/live/<server name here>/privkey.pem /home/ubuntu/augur/privkey.pem` into augur's root directory, then change the user and group permissions with `sudo chown ubuntu <filename.pem>` and `sudo chgrp ubuntu <filename.pem` for both pem files. Now that the user permissions are set properly, gunicorn should be able to access them but we still need to add them to our gunicorn configuration document in `augur/application.py`. Change the corresponding code block to look like this:

```
self.gunicorn_options = {
        'bind': '%s:%s' % (self.config.get_value("Server", "host"), self.config.get_
↪value("Server", "port")),
        'workers': int(self.config.get_value('Server', 'workers')),
        'timeout': int(self.config.get_value('Server', 'timeout')),
        'certfile': '/home/ubuntu/augur/fullchain.pem',
        'keyfile': '/home/ubuntu/augur/privkey.pem'
    }
```

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

1.4 Getting Started

This section of the documentation is a no work experience required walkthrough of the Augur project. By the end, you'll hopefully have a fully functioning local installation of Augur ready to collect data.

If you want to get started as fast as possible, we have Docker images; however, if you're looking to use Augur for long-term data collection or if you want to install it for development, you'll need to follow this walkthrough.

Note: We currently officially support the local installation of Augur from source on macOS, Ubuntu, and Fedora (but most UNIX-like systems will probably work with a few tweaks). We recommend either using the Docker images or setting up a virtual machine with a supported operating system installed if you are using Windows.

To install from source, we'll need to do a few things:

1. Setup a PostgreSQL instance to store the data collected by Augur
2. Install and configure Augur's application server
3. Install and configure Augur's data collection workers

The next section will start with a database setup, and then you can continue with the following steps given below.

Happy hacking!

1.4.1 Database setup

One of the reasons that Augur is so powerful is because of its unified data model. To ensure this data model remains performant with large amounts of data, we use PostgreSQL as our database engine. We'll need to set up a PostgreSQL instance and create a database, after which Augur can take care of the rest. Make sure to save off the credentials you use when creating the database; you'll need them again to configure Augur.

PostgreSQL Installation

Before you can install our schema, you will need to make sure you have **write access** to a PostgreSQL 10 or later database. If you're looking for the fastest possible way to get Augur started, we recommend use our database container. If you're looking to collect data long-term, we recommend following the rest of this tutorial and setting up a persistent PostgreSQL installation.

> **Warning:** If you want to collect data over the long term, we strongly advise against using a Docker container for your database.

If you're a newcomer to PostgreSQL, you can follow their excellent instructions here to set it up for your machine of choice. We recommend using `Postgres.app` if you're on macOS, but if you're running UNIX or are looking for an alternative to `Postgres.app` then pgAdmin is a great open-source alternative.

Creating a Database

After you set up your PostgreSQL instance, you'll need to create a database and user with the correct permissions. You can do this with the SQL commands below, but be sure to change the password!

```
CREATE DATABASE augur;
CREATE USER augur WITH ENCRYPTED PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE augur TO augur;
```

For example, if you were using `psql` to connect to an instance on your machine `localhost` under the default user `postgres` on the default PostgreSQL port `5432`, you might run something like this to connect to the server:

```
$ psql -h localhost -U postgres -p 5432
```

Then, once you've connected to your PostgreSQL instance:

```
postgres=# CREATE DATABASE augur;
postgres=# CREATE USER augur WITH ENCRYPTED PASSWORD 'password';
postgres=# GRANT ALL PRIVILEGES ON DATABASE augur TO augur;
```

Once you've got the database setup, Augur will install the schema for you. You're now ready to install Augur!

1.4.2 Installation Guide

This section shows how to install Augur's Python library from the source. If you don't have a required dependency, please follow the provided links to install and configure it. .. note:

```
There are three main issues new developers encounter when first installing Augur:

1. The absence of a `GCC` or `Fortran` compiler; which are required by NumPy and NLTK␣
↪Python libraries. Look up how to install these compilers for your local operating␣
↪system. Many times they need to be updated to a more current version.

2. Conflicting versions of Python: The fix is platform-specific. On Mac OS X, multiple␣
↪versions of Python often have been installed by the OS, brew, Anaconda, or both. The␣
↪result is some python commands draw from different paths because of how they link in `/
↪usr/local/bin`

3. Multiple, or conflicting versions of PostgreSQL, sometimes due to the absence of a␣
↪functional `psql` function at the command line.
```

General Requirements

Backend

Required:

- GitHub Access Token (`repo` and all `read` scopes except `enterprise`)
- GitLab Access Token
- Python 3.6 - 3.8

Python 3.9 is not yet supported because TensorFlow, which we use in our machine learning workers, does not yet support Python 3.9.

Our REST API & data collection workers write in Python 3.6. We query the GitHub & GitLab API to collect data about issues, pull requests, contributors, and other information about a repository, so GitLab and GitHub access tokens are **required** for data collection.

Optional:

- Go 1.12 or later

The `value_worker` uses a Go package called scc to run COCOMO calculations. Once you've installed Go, follow the appropriate steps for your system to install the `scc` package.

- Install gcc OpenMP Support: `sudo apt-get install libgomp1` – Ubuntu

The `message_insights_worker` uses a system-level package called OpenMP. You will need this installed at the system level for that worker to work.

Caching System (Redis)

- Linux Installation
- Mac Installation
- Windows Installation

Message Broker (RabbitMQ)

- Linux Installation
- Mac Installation
- Windows Installation

After installation, you must also set up your rabbitmq instance by running the below commands:

```
sudo rabbitmqctl add_user augur password123

sudo rabbitmqctl add_vhost augur_vhost

sudo rabbitmqctl set_user_tags augur augurTag

sudo rabbitmqctl set_permissions -p augur_vhost augur ".*" ".*" ".*"
```

Note: it is important to have a static hostname when using rabbitmq as it uses hostname to communicate with nodes.

Then, start rabbitmq server with .. code-block:: bash

    sudo systemctl start rabbitmq.service

If your setup of rabbitmq is successful your broker url should look like this:

broker_url = 'amqp://augur:password123@localhost:5672/augur_vhost'

During installation you will be prompted for this broker url.

Frontend

If you're interested in using our visualizations, you can optionally install the frontend dependencies:

- Node
- npm
- Vue.js
- Vue-CLI

We use Vue.js as our frontend web framework and `npm` as our package manager.

Visualization API calls

On Ubuntu and other Linux flavors: if you want to use the new Augur API Calls that generate downloadable graphics developed in the *https://github.com/chaoss/augur-community-reports* repository, you need to install the *firefox-geckodriver* (on Ubuntu or Red Hat Fedora) or *geckodriver* on Mac OSX, at the system level. This dependency exists because the Bokeh libraries we use for these APIs require a web browser engine.

For Ubuntu you can use:

```
- which firefox-geckodriver
- if nothing returned, then:
- sudo apt install firefox-geckodriver
```

For Fedora you can use

```
- which firefox-geckodriver
- if nothing returned, then:
- sudo dnf install firefox-geckodriver
```

For Mac OSX you can use:

```
-  which geckodriver
-  if nothing returned, then:
-  brew install geckodriver
```

Note: If you have BOTH Firefox-geckodriver AND ChromeDriver installed the visualization API will not work.

We have fully tested with Firefox-gecko driver on Linux platforms, and geckodriver on OSX. If you have ONLY ChromeDriver installed, it will probably work. Open an issue if you have a functioning ChromeDriver implementation.

Installation

Now you're ready to build! The steps below outline how to create a virtual environment (**required**) and start the installation process, after which you'll move on to the next section to configure the workers. The instructions are written in a way that you can follow for your respective Operating System.

Note: Lines that start with a $ denote a command that needs to run in an interactive terminal.

Warning: Do **NOT** install or run Augur using `sudo`. It is not required, and using it will inevitably cause some permissions trouble.

For macOS Errata

If you're running Augur on macOS, we strongly suggest updating your shell's initialization script in the following:

In a terminal, open the script:

nano .bash_profile

Add the following line to the end of the file:

export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES

Save the file and exit. Run this command to reload bash_profile:

source .bash_profile

Check if it is updated:

env

`env` should contain `OBJC_DISABLE_INITIALIZE_FORK_SAFETY`.

macOS takes "helpful" measures to prevent Python subprocesses (which Augur uses) from forking cleanly, and setting this environment variable disables these safety measures to restore regular Python functionality.

Warning: If you skip this step, you'll likely see all housekeeper jobs randomly exiting for no reason, and the Gunicorn server will not behave nicely either. Skip this step at your own risk!

General Augur Installation Steps (Irrespective of Operating System)

1. Clone the repository and change to the newly-created directory.

```
$ git clone 'https://github.com/chaoss/augur.git'
$ cd augur/
```

2. Create a virtual environment in a directory of your choosing. Be sure to use the correct `python` command for your installation of Python 3: on most systems, this is `python3`, but yours may differ (you can use `python -V` or `python3 -V` to check).

```
# to create the environment
$ python3 -m venv $HOME/.virtualenvs/augur_env

# to activate the environment
$ source $HOME/.virtualenvs/augur_env/bin/activate
```

3. Set AUGUR_DB environment variable with a postgres database connection string (if you have not setup a database yet, refer to *database setup*) Note: Three terminals will be needed to collect data for augur, and AUGUR_DB needs to be set for 2 out of the 3. If you don't want to add it to both terminals you can add it permanently in your .bashrc file if running bash, or .zshrc file if in running zsh.

```
# set postgres database connection string to AUGUR_DB environment variable
# replace <> variables with actual values
$ export AUGUR_DB=postgresql+psycopg2://<user>:<password>@<host>:<port>/<database_name>
```

4. Run the install script. This script will:

- Install Augur's Python library and application server

- Install Augur's schema in the configured database
- Prompt you for GitHub and GitLab keys
- Add GitHub and GitLab keys to config table in the database

Note: The install script will also generate an Augur API key for your database at the very end. This key will be automatically inserted into your database and printed to your terminal. It requires to use the repo & repo group creation endpoints, so **make sure you save it off somewhere!** There is only one key per database.

```
# run the install script
$ make install
```

```
# If you want to develop with Augur, use this command instead
$ make install-dev
```

If you think something went wrong, check the log files in `logs/`. If you want to try again, you can use `make clean` to delete any build files before running `make install` again.

MacOS users:

If your build fails and in gunicorn.log you see this error: `Connection in use: ('0.0.0.0', 5000)`, that means port 5000 is being used by another process. To solve this issue, go to System Preferences -> Sharing -> Disable Airplay Receiver.

If you want to test new code you have written, you can rebuild Augur using:

```
$ make rebuild-dev
```

Note: If you chose to install Augur's frontend dependencies, you might see a bunch of `canvas@1.6.x` and `canvas-prebuilt@1.6.x` errors in the installation logs. These are harmless and caused by a few of our dependencies having *optional* requirements for old versions of these libraries. If they seem to be causing you trouble, feel free to open an issue.

To enable log parsing for errors, you need to install Elasticsearch and Logstash.

> **Warning:** Please note, that Logstash v7.0 and above have unresolved issues that affect this functionality.
>
> In order to use it in the near future, please download v6.8.
>
> If you use a package manager, it defaults to v7+, so we recommend downloading binary .
>
> This change is tested with Elasticsearch v7.8.0_2 and Logstash v6.8.10.

Once everything installs, you're ready to configure your data collection workers!

1.4.3 Collecting data

Now that you've installed Augur's application server, it's time to configure data collection if needed. If you just want to run Augur using the default repositories in the default database, and default celery collection settings, all you need to do is start the redis server in one terminal, make sure rabbitmq is running, and the augur application in the other terminal. (Don't forget that the AUGUR_DB environment variable needs to be set in the terminal, or set permanently)

```
 # Terminal Window 1

# Starts the redis server
 redis-server
```

```
 # Terminal Window 3

# To Start Augur:
(nohup augur backend start)

# To Stop Augur:
augur backend stop
augur backend kill
```

Now, here's a ton of brain-splitting detail about celery collection. There are 2 pieces to data collection with Augur: the celery worker processes, and the job messages passed through rabbitmq. The jobs to collect are determined by a monitor process started through the cli that starts the rest of augur. The monitor process generates the jobs messages to send to rabbitmq through the collection_status table that informs the status of jobs that have yet to be run. The celery collection workers can then accept these jobs, after which they will use the information provided in the job to find the repositories in question and collect the requested data.

Since the default setup will work for most use cases, we'll first cover how to configure some specific data collection jobs and then briefly touch on the celery configuration options, after which we'll cover how to add repos and repo groups to the database.

Configuring Collection

There are many collection jobs that ship ready to collect out of the box:

- `augur.tasks.git.facade_taks` (collects raw commit and contributor data by parsing Git logs)
- `augur.tasks.github` (parent module of all github specific collection jobs)
- `augur.tasks.github.contributors.tasks` (collects contributor data from the GitHub API)
- `augur.tasks.github.pull_requests.tasks` (collects pull request data from the GitHub API)
- `augur.tasks.github.repo_info.tasks` (collects repository statistics from the GitHub API)
- `augur.tasks.github.releases.tasks` (collects release data from the GitHub API)
- `augur.tasks.data_analysis.insight_worker.tasks` (queries Augur's metrics API to find interesting anomalies in the collected data)

All worker configuration options are found in the config table generated when augur was installed. The config table is located in the augur_operations schema of your postgresql database. Each configurable data collection job set has its subsection with the same or similar title as the task's name. We recommend leaving the defaults and only changing them when explicitly necessary, as the default parameters will work for most use cases. Read on for more on how to make sure your workers are properly configured.

Worker-specific configuration options

Next up are the configuration options specific to some collection tasks (but some tasks require no additional configuration beyond the defaults). The most pertinent of these options is the `Facade` section `repo_directory`, so make sure to pay attention to that one.

Facade

- `repo_directory`, which is the local directory where the facade tasks will clone the repositories it needs to analyze. You should have been prompted for this during installation, but if you need to change it, make sure that it's an absolute path (environment variables like `$HOME` are not supported) and that the directory already exists. Defaults to `repos/`, but it's highly recommended you change this.
- `limited_run`, toggle between 0 and 1 to determine whether to run all facade tasks or not. Runs all tasks if set to 0
- `pull_repos`, toggle whether to pull updates from repos after cloning them. If turned off updates to repos will not be collected.
- `run_analysis`, toggle whether to process commit data at all. If turned off will only clone repos and run tertiary tasks such as resolving contributors from any existing commits or collecting dependency relationships. Mainly used for testing.
- `run_facade_contributors`, toggle whether to run contributor resolution tasks. This will process and parse through commit data to link emails to contributors as well as aliases, etc.
- `force_invalidate_caches`, set every repo to reset the status of commit email affillation, which is the organization that an email is associated with.
- `rebuild_caches`, toggle whether to enable parsing through commit data to determine affillation and web cache

Insight_Task

We recommend leaving the defaults in place for the insight worker unless you are interested in other metrics, or anomalies for a different time period.

- `training_days`, which specifies the date range that the `insight_worker` should use as its baseline for the statistical comparison. Defaults to `365`, meaning that the worker will identify metrics that have had anomalies compared to their values over the course of the past year, starting at the current date.
- `anomaly_days`, which specifies the date range in which the `insight_worker` should look for anomalies. Defaults to `2`, meaning that the worker will detect anomalies that have only occured within the past two days, starting at the current date.
- `contamination`, which is the "sensitivity" parameter for detecting anomalies. Acts as an estimated percentage of the training_days that are expected to be anomalous. The default is `0.041` for the default training days of 365: 4.1% of 365 days means that about 15 data points of the 365 days are expected to be anomalous.
- `switch`, toggles whether to run insight tasks at all.
- `workers`, number of worker processes to use for insight tasks.

Task_Routine

This section is for toggling sets of jobs on or off.

- `prelim_phase`, toggles whether to run preliminary tasks that check to see whether repos are valid or not.
- `primary_repo_collect_phase`, toggle the standard collection jobs, mainly pull requests and issues
- `secondary_repo_collect_phase`, toggle the secondary collection jobs, mainly jobs that take a while
- `facade_phase`, toggle all facade jobs
- `machine_learning_phase`, toggle all ml related jobs

Celery Configuration

We strongly recommend leaving the default celery blocks generated by the installation process, but if you would like to know more, or fine-tune them to your needs, read on.

The celery monitor is responsible for generating the tasks that will tell the other worker processes what data to collect, and how. The `Celery` block has 2 keys; one for memory cap and one for materialized views interval. - `worker_process_vmem_cap`, float between zero and one that determines the maximum percentage of total memory to use for worker processes

- `refresh_materialized_views_interval_in_days`, number of days to wait between refreshes of materialized views.

Adding repos for collection

If you're using the Docker container, you can use the provided UI to load your repositories. Otherwise, you'll need to use the Augur CLI or the augur frontend to load your repositories. Please reference the respective sections of the documentation for detailed instructions on how to accomplish both of these steps.

Running collections

Congratulations! At this point you (hopefully) have a fully functioning and configured Augur instance.

After you've loaded your repos, you're ready for your first collection run. We recommend running only the default jobs first to gather the initial data.

You can now run Augur and start the data collection by issuing the `augur backend start` command in the root `augur` directory. All your logs (including worker logs and error files) will be saved to a `logs/` subdirectory in that same folder, but this can be customized - more on that and other logging utilities in the development guide.

Once you've finished the initial data collection, we suggest then running the `value_worker` (if you have it installed) and the `insight_worker`. This is because the `value_worker` depends on the source files of the repositories cloned by the `facade_worker`, and the `insight_worker` uses the data from all the other workers to identify anomalies in the data by by performing statistical analysis on the data returned from Augur's metrics API.

You're now ready to start exploring the data Augur can gather and metrics we can generate. If you're interested in contributing to Augur's codebase, you can check out the development guide. For information about Augur's frontend, keep reading!

Happy collecting!

1.4.4 Augur's Frontend

To compile Augur's frontend for deployment in a production environment (i.e., behind an nginx server), you must go through the folowing steps

1. Run `augur config init-frontend` from within your python virtual environment.
2. Enter Augur's home directory
3. Run `npm install`, and then `npm run build` in the frontend directory.
4. After that, follow the instructions for configuring Augur behind Nginx.

Augur's frontend source code can be found at `<root_augur_directory>/frontend/src/`. It uses Vue.js as its primary architecture framework, and Vega/Vega-Lite for its visualizations. It's configured via the `frontend.config.json` file in `<root_augur_directory>/frontend/`.

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

1.4.5 Command Line Interface

Augur provides a command line interface (CLI) for interacting with your Augur installation. It's broken up into a few categories: `db`, `backend`, `util`, `config`, and `logging`.

Each command is invoked by first specifying the category, then the command name, and then the parameters/options; e.g. the `list` command under `augur util` would be invoked as `augur backend start --option1 ...`.

Note: Throughout this section of the documentation, all lines that start with a `$` denote a `bash` command, and lines with > denote some sample output of command.

Database Commands

`augur db`

The collection of `augur db` commands is for interacting with the database.

`add-repo-groups`

The `add-repo-groups` command is used to create new repo groups. When given a path to a correctly formatted `.csv` file, it will create each repo group specified in the file with the corresponding ID and name.

The `.csv` file must have the following format:

```
<repo_group_id>,<repo_group_name>
...
```

Where `<repo_group_id>` is the desired ID of the new repo group, and `<repo_group_name>` is the desired name of the new repo group.

Each pair of values should be on its own line (indicated by the ...), without quotes, and there should be no column headers.

Example usage:

```
# to add new repos to the database
$ augur db add-repo-groups repo_groups.csv

# contents of repo_groups.csv
10,Repo Group 1
20,Repo Group 2

# successful output looks like:
> CLI: [db.add_repo_groups] [INFO] Inserting repo group with name Repo Group 1 and ID 10.
↪..
> CLI: [db.add_repo_groups] [INFO] Inserting repo group with name Repo Group 2 and ID 20.
↪..
```

get-repo-groups

The `get-repo-groups` command will return the ID, name, and description of all repo groups in the database.

Example usage:

```
# to retrieve the repo groups
$ augur db get-repo-groups

# successful output looks like:
>    repo_group_id             rg_name                                        rg_description
> 0              1  Default Repo Group  The default repo group created by the schema g...
> 1             10        Repo Group 1
> 2             20        Repo Group 2
```

add-repos

The `add-repos` command is used to load new repos. When given a path to a correctly formatted `.csv` file, it will insert each repo specified in the file into its corresponding repo group in the database specified in the config file.

The `.csv` file must have the following format:

```
<git_repo_url>,<repo_group_id>,
...
```

where <repo_group_id> is an **existing** repository group ID, and <git_repo_url> is the url to the repository's Git repository, e.g. `https://github.com/chaoss/augur.git`. Each pair of values should be on its own line (indicated by the ...), without quotes, and there should be no column headers.

Note: If you don't know what `repo_group_id` to use, run the `augur db get-repo-groups` command to view the repo groups that are currently in your DB; unless you've deleted it, there should be a default one that you can use.

Example usage:

```
# contents of repos.csv
10,https://github.com/chaoss/augur.git
10,https://github.com/chaoss/grimoirelab.git
```

(continues on next page)

(continued from previous page)

```
20,https://github.com/chaoss/wg-evolution.git
20,https://github.com/chaoss/wg-risk.git
20,https://github.com/chaoss/wg-common.git
20,https://github.com/chaoss/wg-value.git
20,https://github.com/chaoss/wg-diversity-inclusion.git
20,https://github.com/chaoss/wg-app-ecosystem.git

# to add repos to the database
$ augur db add-repos repos.csv

# successful output looks like
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/
↪augur.git` into repo group 10
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/
↪grimoirelab.git` into repo group 10
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/wg-
↪evolution.git` into repo group 20
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/wg-
↪risk.git` into repo group 20
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/wg-
↪common.git` into repo group 20
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/wg-
↪value.git` into repo group 20
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/wg-
↪diversity-inclusion.git` into repo group 20
> CLI: [db.add_repos] [INFO] Inserting repo with Git URL `https://github.com/chaoss/wg-
↪app-ecosystem.git` into repo group 20
```

generate-api-key

The `generate-api-key` command will generate a new Augur API key and update the database with the new key. Output is the generated key.

Example usage:

```
# to generate a key
$ augur db generate-api-key

# successful output looks like (this will be an actual key):
> CLI: [db.update_api_key] [INFO] Updated Augur API key to: new_key_abc_123
> new_key_abc_123
```

get-api-key

The `get-api-key` command will return the API key of the currently configured database. Output is the API key.

Example usage:

```
# to retrieve the key
$ augur db get-api-key

# successful output looks like (this will be an actual key):
> existing_key_def_456
```

print-db-version

The `print-db-version` command will give user the current version of the configured database on their system.

Example usage:

```
# to return the current database version
$ augur db print-db-version

# successful output looks like:
> 1
```

upgrade-db-version

The `upgrade-db-version` command will upgrade the user's current database to the latest version.

Example usage:

```
# to upgrade the user's database to the current version
$ augur db upgrade-db-version

# successful output if your DB is already up to date
> CLI: [db.check_pgpass_credentials] [INFO] Credentials found in $HOME/.pgpass
> CLI: [db.upgrade_db_version] [INFO] Your database is already up to date.

# successful output if your DB needs to be upgraded
> [INFO] Attempting to load config file
> [INFO] Config file loaded successfully
> CLI: [db.check_pgpass_credentials] [INFO] Credentials found in $HOME/.pgpass
> CLI: [db.upgrade_db_version] [INFO] Upgrading from 16 to 17
> ALTER TABLE "augur_data"."repo"
>   ALTER COLUMN "forked_from" TYPE varchar USING "forked_from"::varchar;
> ALTER TABLE
> ALTER TABLE "augur_data"."repo"
>   ADD COLUMN "repo_archived" int4,
>   ADD COLUMN "repo_archived_date_collected" timestamptz(0),
>   ALTER COLUMN "forked_from" TYPE varchar USING "forked_from"::varchar;
> ALTER TABLE
> update "augur_operations"."augur_settings" set value = 17 where setting = 'augur_data_
↪version';
```

(continues on next page)

(continued from previous page)

```
> UPDATE 1
> CLI: [db.upgrade_db_version] [INFO] Upgrading from 17 to 18
> etc...
```

create-schema

The `create-schema` command will attempt to create the Augur schema in the database defined in your config file.

Example usage:

```
# to create the schema
$ augur db create-schema
```

Note: If this runs successfully, you should see a bunch of schema creation commands fly by pretty fast. If everything worked you should see: `update "augur_operations"."augur_settings" set value = xx where setting = 'augur_data_version';` at the end.

Backend Commands

augur backend

The `augur backend` CLI group is for controlling Augur's API server & data collection workers. All commands are invoked like:

```
$ augur backend <command name>
```

start

This command is for starting Augur's API server & (optionally) data collection workers. Example usages are shown below the parameters. After starting up, it will run indefinitely (but might not show any output, unless it's being queried or the housekeeper is working).

--disable-housekeeper Flag that turns off the housekeeper. Useful for testing the REST API or if you want to pause data collection without editing your config.

--skip-cleanup Flag that disables the old process cleanup that runs before Augur starts. Useful for Python scripts where Augur needs to be run in the background: see the *test/api/runner.py* file for an example.

To start the backend normally:

```
augur backend start

# successful output looks like:
>[43389] augur [INFO] Augur application initialized
>[43389] augur [INFO] Booting manager
>[43389] augur [INFO] Booting broker
>[43389] augur.housekeeper [INFO] Booting housekeeper
>[43389] augur.housekeeper [INFO] Preparing housekeeper jobs
```

(continues on next page)

(continued from previous page)

```
>[43389] augur.housekeeper [INFO] Scheduling update processes
>[43389] augur [INFO] Booting facade_worker #1
>[43389] augur [INFO] Booting github_worker #1
>[43389] augur [INFO] Booting linux_badge_worker #1
>[43389] augur [INFO] Booting pull_request_worker #1
>[43389] augur [INFO] Booting repo_info_worker #1
>[43389] augur [INFO] Booting contributor_worker #1
>[43389] augur [INFO] Booting gitlab_issues_worker #1
>[43389] augur [INFO] Booting release_worker #1
>[43389] augur [INFO] Starting Gunicorn server in the background...
>[43389] augur [INFO] Housekeeper update process logs will now take over.
>[43645] augur.jobs.insights [INFO] Housekeeper spawned insights model updater process␣
↪for repo group id 0
>[43639] augur.jobs.issues [INFO] Housekeeper spawned issues model updater process for␣
↪repo group id 0
>[43646] augur.jobs.badges [INFO] Housekeeper spawned badges model updater process for␣
↪repo group id 0
>[43640] augur.jobs.pull_request_commits [INFO] Housekeeper spawned pull_request_commits␣
↪model updater process for repo group id 0
>[43642] augur.jobs.commits [INFO] Housekeeper spawned commits model updater process for␣
↪repo group id 0
>[43647] augur.jobs.value [INFO] Housekeeper spawned value model updater process for␣
↪repo group id 0
>[43644] augur.jobs.contributors [INFO] Housekeeper spawned contributors model updater␣
↪process for repo group id 0
>[43641] augur.jobs.repo_info [INFO] Housekeeper spawned repo_info model updater process␣
↪for repo group id 0
>[43643] augur.jobs.pull_requests [INFO] Housekeeper spawned pull_requests model updater␣
↪process for repo group id 0
>[43648] augur.jobs.pull_request_files [INFO] Housekeeper spawned pull_request_files␣
↪model updater process for repo group id 0
> ...
> From this point on, the housekeeper and broker logs detailing the worker's progress␣
↪will take over
```

To start the backend as a background process:

```
nohup augur backend start >logs/base.log 2>logs/base.err &
```

Successful output looks like the generation of standard Augur logfiles in the logs/ directory.

To start the backend server without the housekeeper:

```
augur backend start --disable-housekeeper
```

Successful output looks like:

```
> [14467] augur [INFO] Augur application initialized
> [14467] augur [INFO] Using config file: /Users/carter/workspace/chaoss/augur/augur.
↪config.json
> [14467] augur [INFO] Starting Gunicorn webserver...
> [14467] augur [INFO] Augur is running at: http://0.0.0.0:5000
> [14467] augur [INFO] Gunicorn server logs & errors will be written to logs/gunicorn.log
```

stop

Gracefully attempts to stop all currently running backend Augur processes, including any workers. Will only work in a virtual environment.

Example usage:

```
augur backend stop
```

Successful output looks like:

```
> CLI: [backend.stop] [INFO] Stopping process 33607
> CLI: [backend.stop] [INFO] Stopping process 33775
> CLI: [backend.stop] [INFO] Stopping process 33776
> CLI: [backend.stop] [INFO] Stopping process 33777
```

kill

Forcefully terminates (using `SIGKILL`) all currently running backend Augur processes, including any workers. Will only work in a virtual environment. Should only be used when `augur backend stop` is not working.

Example usage:

```
augur backend kill

# successful output looks like:
> CLI: [backend.kill] [INFO] Killing process 87340
> CLI: [backend.kill] [INFO] Killing process 87573
> CLI: [backend.kill] [INFO] Killing process 87574
> CLI: [backend.kill] [INFO] Killing process 87575
> CLI: [backend.kill] [INFO] Killing process 87576
```

processes

Outputs the process ID (PID) of all currently running backend Augur processes, including any workers. Will only work in a virtual environment.

Example usage:

```
augur backend processes
```

Successful output looks like:

```
> CLI: [backend.processes] [INFO] Found process 14467
> CLI: [backend.processes] [INFO] Found process 14725
```

To enable log parsing for errors, you need to install Elasticsearch and Logstash.

Warning: Please note, that Logstash v7.0 and above has unresolved issues that affect this functionality. In order to use it in the near future, please download v6.8. If you use a package manager, it defaults to v7+, so we recommend downloading binary. This change is tested with Elasticserach v7.8.0_2 and Logstash v6.8.10.

Set ELASTIC_SEARCH_PATH and LOGSTASH_PATH variables to point to elasticsearch and logstash binaries. For example:

```
# If not specified, defaults to /usr/local/bin/elasticsearch
$ export ELASTIC_SEARCH_PATH=<path_to_elastic_search_binary>

# If not specified, defaults to /usr/local/bin/logstash
$ export LOGSTASH_PATH=<path_to_logstash_binary>

$ export ROOT_AUGUR_DIRECTORY=<path_to_augur>
```

Start the http server with:

```
$ cd $ROOT_AUGUR_DIRECTORY/log_analysis/http
$ python http_server.py
```

Then start Augur with logstash flag:

```
$ augur backend start --logstash
```

If you'd like to clean all previously collected errors, run:

```
$ augur backend start --logstash-with-cleanup
```

Open http://localhost:8003 and select workers to check for errors.

export-env

Exports your GitHub key and database credentials to 2 files. The first is augur_export_env.sh which is an executable shell script that can be used to initialize environment variables for some of your credentials. The second is docker_env.txt which specifies each credential in a key/value pair format that is used to configure the backend Docker containers.

Example usage:

```
# to export your environment
$ augur util export-env
```

Successful output looks like:

```
> CLI: [util.export_env] [INFO] Exporting AUGUR_GITHUB_API_KEY
> CLI: [util.export_env] [INFO] Exporting AUGUR_DB_HOST
> CLI: [util.export_env] [INFO] Exporting AUGUR_DB_NAME
> CLI: [util.export_env] [INFO] Exporting AUGUR_DB_PORT
> CLI: [util.export_env] [INFO] Exporting AUGUR_DB_USER
> CLI: [util.export_env] [INFO] Exporting AUGUR_DB_PASSWORD

# contents of augur_export_env.sh
#!/bin/bash
export AUGUR_GITHUB_API_KEY="your_key_here"
export AUGUR_DB_HOST="your_host"
export AUGUR_DB_NAME="your_db_name"
export AUGUR_DB_PORT="your_db_port"
export AUGUR_DB_USER="your_db_user"
```

(continues on next page)

(continued from previous page)

```
export AUGUR_DB_PASSWORD="your_db_password"

# contents of docker_env.txt
AUGUR_GITHUB_API_KEY="your_key_here"
AUGUR_DB_HOST="your_host"
AUGUR_DB_NAME="your_db_name"
AUGUR_DB_PORT="your_db_port"
AUGUR_DB_USER="your_db_user"
AUGUR_DB_PASSWORD="your_db_password"
```

repo-reset

Refresh repo collection to force data collection. Mostly for debugging.

Example usage:

```
# to reset the repo collection status to "New"
$ augur util repo-reset

# successful output looks like:
> CLI: [util.repo_reset] [INFO] Repos successfully reset
```

Configuration Commands

augur config

The `augur config` commands are for interacting with Augur's configuration file.

init

The `init` command is used to create a configuration file, by default named `augur.config.json`. Each of the available parameters is optional, and can also be configured using an existing environment variable. Below is the list of available parameters, their defaults, and the corresponding environment variable.

--db_name	Database name for your data collection database. Defaults to `augur`. Set by the `AUGUR_DB_NAME` environment variable
--db_host	Host for your data collection database. Defaults to `localhost`. Set by the `AUGUR_DB_HOST` environment variable
--db_user	User for your data collection database. Defaults to `augur`. Set by the `AUGUR_DB_USER` environment variable
--db_port	Port for your data collection database. Defaults to `5432`. Set by the `AUGUR_DB_PORT` environment variable
--db_password	Password for your data collection database. Defaults to `augur`. Set by the `AUGUR_DB_PASSWORD` environment variable
--github_api_key	GitHub API key for data collection from the GitHub API. Defaults to `key`. Set by the `AUGUR_GITHUB_API_KEY` environment variable

--facade_repo_directory The directory on this machine where Facade should store its cloned repos. Defaults to `repos/`. Set by the `AUGUR_FACADE_REPO_DIRECTORY` environment variable

--rc-config-file Path to an existing Augur config file whose values will be used as the defaults. Defaults to `None`. This parameter does not support being set by an environment variable.

--write-to-src Flag for writing the generated config file to the source code tree, instead of the default `$HOME/.augur`. For developers use only. Defaults to `False`.

Example usage:

```
# to generate an augur.config.json file with all the defaults
$ augur config init

# to generate an augur.config.json given all credentials as literals
$ augur config init --db_name "db_name" --db_host "host" --db_port "port" --db_user "db_
↪user" --db_password "password" --github_api_key "github_api_key" --facade_repo_
↪directory "facade_repo_directory"

# to generate an augur.config.json given all credentials and environment variables
$ augur config init --db_name $AUGUR_DB_NAME --db_host $AUGUR_DB_HOST --db_port $AUGUR_
↪DB_PORT --db_user $AUGUR_DB_DB_USER --db_password $AUGUR_DB_PASSWORD --github_api_key
↪$AUGUR_GITHUB_API_KEY --facade_repo_directory $AUGUR_FACADE_REPO_DIRECTORY

# successful output looks like:
> CLI: [config.init] [INFO] Config written to /Users/carter/.augur/augur.config.json
```

Logging Commands

augur logging

The collection of the `augur logging` commands is for interacting with the database.

directory

Prints the location of the directory to which Augur is configured to write its logs.

Example usage:

```
# to stop the server and workers
$ augur logging directory

# successful output looks like:
> /Users/carter/projects/work/augur/logs/
```

`tail`

Prints the last `n` lines of each `.log` and `.err` file in the logs directory. `n` defaults to 20.

Example usage:

```
# to stop the server and workers
$ augur logging tail

# successful output looks like:
> ********** Logfile: augur.log
  <contents of augur.log>

> ********** Logfile: augur.err
  <contents of augur.err>
```

1.5 Development Guide

This is the development guide for Augur. See our Contributing to Augur guide for specifics on how we review pull requests.

1.5.1 Installing for Development

Installing Augur for local development is pretty similar to the normal installation process. This guide will primarily detail the differences between the two instead of regurgitating all the information in the Getting Started section. If you are completely new to Augur, we recommend following the aforementioned Getting Started section first; once you feel more comfortable with Augur and how to use it, come back to this document.

Setting up the Database

If they so desire, developers can set up a persistent instance of PostgreSQL on either the local machine or a remote server. The instructions for doing so can be found in the database portion of the Getting Started section

However, during development, you might find that you need to reset your database often, especially if you are working on the data collection components of Augur. To this end, we recommend developers make use of our Docker images to quickly provision and terminate database instances in a lightweight and reproducible manner.

More information about Augur's Docker images can be found here. If you're new to our Docker process, we recommend following the introduction section first.

Installing from Source

The process for installing Augur's source code for development is essentially the same as detailed in the Installation section of the Getting Started guide.

However, when running the installation script, use the following command instead:

```
$ make install-dev
```

This will install a few extra dependencies for testing and documentation, as well as install all the Python packages in editable mode. This means you will not have to reinstall the package every time you make a change to the Python source code.

This command will also create your `augur.config.json` file in the root of your cloned source code directory **instead of** the default location in `$HOME/.augur/`. This is purely for convenience's sake, as it will allow you to open this file in your text editor with all the other source code files, and also allows you to have multiple developer installations of Augur on the same machine if needed. If Augur finds a config file in both the root of the cloned directory AND in the default location, it will always use the one in the root of the cloned directory.

Note: You can still use `make clean` to get rid of the installed binaries if something went wrong and you want to try again.

Conclusion

All in all, it's pretty similar. For further reading, the Makefile documentation and the Creating a Metric guide are good places to start.

Happy hacking!

1.5.2 Make commands

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

Installation

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

This section explicitly explains the commands that are used to manage the installation of Augur locally.

`make install`

This command installs the project dependencies, sets up the default configuration file, and gathers database credentials.

Example:

```
$ make install
```

`make install-dev`

The same as `make install`, except it installs the additional developer dependencies and installs the packages in editable mode.

Example:

```
$ make install-dev
```

---

make clean

Removes logs, caches, and some other cruft that can get annoying. This command is used when things aren't building properly or you think an old version of augur is getting in the way.

Example:

```
$ make clean
```

---

make rebuild

Used in conjunction with `make clean` to remove all build/compiled files and binaries and reinstall the project. Useful for upgrading in place.

Example:

```
$ make rebuild
```

---

make rebuild-dev

The same as `make rebuild`, except it installs the additional developer dependencies and installs the packages in editable mode.

Note: You can still use `make clean` as normal if something went wrong.

Example:

```
$ make rebuild-dev
```

Development

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

These commands are used to control Augur's backend and frontend servers simultaneously.

---

make dev

If the above command doesn't work, try running ``make dev-start`` instead.

This command starts the frontend and backend servers together in the background. The output of the backend are in `logs/augur.log`, and the logs for the frontend are in `logs/frontend.log`. The backend output should like something like this (note that your process IDs and hostname will be different):

```
2020-03-22 12:39:28 kaiyote augur[19051] INFO Booting broker and its manager...
2020-03-22 12:39:29 kaiyote augur[19051] INFO Booting housekeeper...
2020-03-22 12:39:51 kaiyote root[19051] INFO Starting update processes...
2020-03-22 12:39:52 kaiyote root[19083] INFO Housekeeper spawned issues model updater␣
↪process for subsection 0 with PID 19083
2020-03-22 12:39:52 kaiyote augur[19051] INFO Starting server...
2020-03-22 12:39:52 kaiyote root[19084] INFO Housekeeper spawned pull_requests model␣
↪updater process for subsection 0 with PID 19084
[2020-03-22 12:39:52 -0500] [19051] [INFO] Starting gunicorn 19.9.0
[2020-03-22 12:39:52 -0500] [19051] [INFO] Listening at: http://0.0.0.0:5000 (19051)
[2020-03-22 12:39:52 -0500] [19051] [INFO] Using worker: sync
[2020-03-22 12:39:52 -0500] [19085] [INFO] Booting worker with pid: 19085
[2020-03-22 12:39:52 -0500] [19086] [INFO] Booting worker with pid: 19086
[2020-03-22 12:39:52 -0500] [19087] [INFO] Booting worker with pid: 19087
[2020-03-22 12:39:53 -0500] [19088] [INFO] Booting worker with pid: 19088
[2020-03-22 12:39:53 -0500] [19089] [INFO] Booting worker with pid: 19089
[2020-03-22 12:39:53 -0500] [19090] [INFO] Booting worker with pid: 19090
[2020-03-22 12:39:53 -0500] [19091] [INFO] Booting worker with pid: 19091
[2020-03-22 12:39:53 -0500] [19092] [INFO] Booting worker with pid: 19092
```

The frontend output should like something like this:

```
  # a whole bunch of stuff about compiling
  ...
  ...
  ...
  ...

Version: typescript 3.5.3, tslint 5.18.0
Time: 9311ms

App running at:
- Local:   http://localhost:8080/
- Network: http://192.168.1.141:8080/
```

Note: You'll likely see some linting warnings in the frontend section (indicated here by the ...). Don't worry about them: it's the last 3 lines that indicate success. Once you see this you're good to go! Head to the specified URL (in this example it's `http://localhost:8080/`) to check it out!

---

make dev-stop

Stops both the frontend and the backend server.

Example:

```
$ make dev-stop
```

Testing

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

These commands are used to run specific subsets of unit tests. We use `tox` to manage the test environments, and `pytest` as the test runner. Each of these commands except for `make test-pythons-versions` will use your default Python version, while `make test-python-versions` will test all supported Python versions.

---

make test

This command runs ALL available tests for both the metric functions and their API endpoints.

Example:

```
$ make test
```

---

make test-metrics

This command will run ALL unit tests for the metric functions.

Example:

```
$ make test-metrics
```

---

make test-metrics-api

The above command runs ALL tests for the metrics API.

Example:

```
$ make test-metrics-api
```

---

make test-python-versions

The above command runs all tests under all currently supported versions of **Python 3.6 and above**.

Example:

```
$ make test-python-versions
```

Documentation

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

These commands are used to build and view Augur's documentation. Before making any documentation changes, please read the documentation guide.

---

make docs

Generate both library and API documentation.

Example:

```
$ make docs
```

---

make library-docs

Generate the library documentation (the documentation you're reading).

Example:

```
$ make library-docs
```

---

make library-docs-view

Generate the library documentation, and automatically open a new browser tab to view it.

Example:

```
$ make library-docs-view
```

---

make api-docs

Generate the API documentation.

Example:

```
$ make api-docs
```

`make api-docs-view`

Generate the API documentation, and automatically open a new browser tab to view it.

Example:

```
$ make api-docs-view
```

1.5.3 Logging

Augur's log output can be configured with some basic verbosity and log levels. If you are contributing to Augur, we recommend you set the `debug` flag in the `Logging` section of your config file to `1`. This will turn the verbosity up, capture **all** logs of every level, and it will allow the data collection tasks to print their output to the screen if they are being run manually in a separate terminal.

The verbosity and minimum log level can be controlled with the `verbose` (boolean flag) and `log_level` (one of `DEBUG`, `INFO`, `WARNING`, `ERROR`, or `CRITICAL`) options respectively. There is also a `quiet` flag that will disable all logging output entirely.

If you need to change where the logs are written to, you can use the `logs_directory` option. If there is no `/` at the beginning, Augur assumes you are specifying a path relative to the root augur directory, otherwise it will set the log location to be exactly what you configured. The log directory itself will be created if it doesn't exist, but only if its parent DOES already exist.

1.5.4 Writing documentation

Currently, we maintain a set of library and usage documentation (which is what you are reading!) that we update with each release. The following sections briefly outline how to contribute to our documentation.

Note: **All** PRs which require a documentation change will not be merged until that change has been made.

Library and Usage Documentation

The library documentation is written using reStructuredText for the raw markdown and then built into web pages using Sphinx.

We'll avoid going over reStructuredText in detail here, but here is a good reference document.

Similarly, we'll avoid going over Sphinx in great detail as well; here is a good reference document for the most commonly used directives.

Building

To see your changes and make sure everything rendered correctly, First activate the python virtual enivroment and run `make docs` in the root `augur/` directory, and then open `docs/build/html/index.html` in your web browser to view it.

```
$ cd augur
$ python3 -m venv $HOME/.virtualenvs/augur_env
$ source $HOME/.virtualenvs/augur_env/bin/activate
$ make docs
```

Or, you can use the shortcut which does exactly this:

```
# to build and then open to the locally built documentation
$ make docs-view
```

After opening it once, Make your changes in the regular `docs/source` folder and just run `make docs` Everytime you make any change and refresh the browser

```
# after opening the documentation
$ make docs
```

Hosting

Our documentation is graciously hosted by Read the Docs.

Enabled branches of the main `chaoss/augur` repository will each have their own documentation, with the default `main` corresponding to `main` on the readthedocs. The documentation will automatically be built and deployed on a push to one of these branches or on any incoming PR, but please don't forget to check before you push!

1.5.5 Workers

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

Creating a New Worker

Worker Setup

1. If you are hitting an API on a platform like GitHub, or GitLab, follow the pattern in those workers.
2. If you are analyzing Augur data, the *value_worker* provides a good example.

What are the key sections?

The key sections you can copy from any worker are illustrated in this example from the Pull Request Worker:

```
#SPDX-License-Identifier: MIT
import ast
import json
import logging
import os
import sys
import time
import traceback
import requests
import copy
from datetime import datetime
from multiprocessing import Process, Queue
import pandas as pd
import sqlalchemy as s
```

(continues on next page)

(continued from previous page)

```
from sqlalchemy.sql.expression import bindparam
from workers.worker_base import Worker

class GitHubPullRequestWorker(Worker):
    """
    Worker that collects Pull Request related data from the
    Github API and stores it in our database.

    :param task: most recent task the broker added to the worker's queue
    :param config: holds info like api keys, descriptions, and database connection␣
↪strings
    """
    def __init__(self, config={}):

        worker_type = "pull_request_worker"

        # Define what this worker can be given and know how to interpret
        given = [['github_url']]
        models = ['pull_requests', 'pull_request_commits', 'pull_request_files']

        # Define the tables needed to insert, update, or delete on
        data_tables = ['contributors', 'pull_requests',
            'pull_request_assignees', 'pull_request_events', 'pull_request_labels',
            'pull_request_message_ref', 'pull_request_meta', 'pull_request_repo',
            'pull_request_reviewers', 'pull_request_teams', 'message', 'pull_request_
↪commits',
            'pull_request_files', 'pull_request_reviews', 'pull_request_review_message_
↪ref']
        operations_tables = ['worker_history', 'worker_job']

        self.deep_collection = True
        self.platform_id = 25150 # GitHub

        # Run the general worker initialization
        super().__init__(worker_type, config, given, models, data_tables, operations_
↪tables)

        # Define data collection info
        self.tool_source = 'GitHub Pull Request Worker'
        self.tool_version = '1.0.0'
        self.data_source = 'GitHub API'
```

Getting Your Worker to Talk to Augur

In the house keeper block, you need to add something following this pattern, inside the "jobs" section:

```
"Housekeeper": {
    "update_redirects": {
        "switch": 0,
        "repo_group_id": 0
    },
    "jobs": [
        {
            "delay": 150000,
            "given": [
                "github_url"
            ],
            "model": "contributor_breadth",
            "repo_group_id": 0
        },
        {
            "all_focused": 1,
            "delay": 150000,
            "given": [
                "github_url"
            ],
            "model": "issues",
            "repo_group_id": 0
        },
        {
            "delay": 150000,
            "given": [
                "<given specified in your worker>"
            ],
            "model": "<model specified in your worker>",
            "repo_group_id": 0
        },
```

In the Worker block you need to add something like this:

```
"Workers": {
    "contributor_breadth_worker": {
        "port": 48234,
        "switch": 0,
        "workers": 1
    },
    "facade_worker": {
        "port": 48868,
        "repo_directory": "/Volumes/repo_two/repos/augur-prwrt/",
        "switch": 1,
        "workers": 1
    },
    "your_worker": {
        "port": <some port not otherwise in use>,
        "switch": 1,
```

(continues on next page)

(continued from previous page)

```
        "workers": 1
    },
```

There should NOT be a comma after the final entry in each block.

ALSO, if you wanted to have those blocks installed with auger itself when you do the PR, you need to add them to the *$AUGUR_ROOT/augur/config.py* file. The recommended way is to set a port range not already in use and assign a random variable range with the others, like this *your_new_worker_p = randint(56500, 56999)* … its totally ok to compress a couple other port ranges for this process.

You can copy the housekeeper block verbatim from what you added to your own *augur.config.json*. For the worker block, in the *config.py* it would look like this:

```
"your_worker": {
    "port": your_worker_p ,
    "switch": 1,
    "workers": 1
},
```

The *switch* variable tells Augur to run your worker. The *worker* variable tells Augur how many to run. We recommend you begin with the number *1*.

Let us know if that works. I will add this to the documentation.

Clustering Task

The task analyzes the comments in issues and pull requests, and clusters the repositories based on contents of those messages. The task also performs topic modeling using Latent Dirichlet allocation

Clustering of text documents

Clustering is a type of unsupervised machine learning technique that involves grouping together similar data points. In case of textual data, it involves grouping together semantically similar documents. The document is a collection of sentences. In our case, document represents the collection of comments across issues and pull requests across a particular repository. Since, clustering algorithm works with numerical features, we need to first convert documents into vector representation.

Implementation

The task performs two tasks — clustering of the repositories represented as documents (collection of all messages from issues and pull requests within the repository) and topic modeling. If the pre-trained model doesn't exist in the clustering task's folder, the data from all the repository in the connected database are used to train the model. After the training, the following model files are dumped in the clustering task's folder

- vocabulary : the set of features obtained from TF-IDF vectorization on text data (required in prediction phase)
- kmeans_repo_messages : trained kmeans clustering model on tfidf features
- vocabulary_count: a set of features obtained from count vectorization on text data (required in prediction phase)
- lda_model : trained latent Dirichlet analysis model for topic modeling

The hyperparameters for the training are obtained from the configuration file. In addition, the training phase populates the 'topic words' database table with the top words belonging to a topic.

Prediction If the trained model exists in the task directory, the prediction is made on the documents corresponding to the repositories in the repo groups specified in the configuration. The task populates the following tables repo_topic : stores probability distribution over the topics for a particular repository repo_cluster_messages : stores clustering label assigned to a repository

Task Configuration

For this task's configuration, in workers configuration block, we need to define port, switch and number of workers.

```
"clustering_worker":{
        "port" : 51500,
        "switch": 1,
        "workers": 1,
        "max_df" : 0.9,
        "max_features" : 1000,
        "min_df" : 0.1,
        "num_clusters" : 4
}
```

Additional Worker Parameters:

In addition to standard worker parameters, clustering worker requires some worker-specific parameters which are described below:

- **max_df** :sets the threshold which filters out terms that have higher document frequency (corpus specific stop words)
- **min_df** : filters out uncommon words
- **max_features** - defines maximum number of features to be used in calculating tfidf matrix
- **num_clusters** - defines number of clusters to segment the repositories into

Discourse Analysis Worker

The worker analyzes the sequential conversation data in GitHub issues and pull requests. The worker classifies the messages into one of the following discourse acts.

- Question
- Answer
- Elaboration
- Announcement
- Agreement
- Disagreement

Structured Prediction Using Conditional Random Field

Structured prediction is a type of supervised machine learning technique that involves predicting a structured group of multiple correlated outputs at the same time. In the problem of discourse act classification, it means predicting discourse labels for each utterance in a discourse sequence simultaneously. Structured prediction methods take advantage of the inherent dependency among the sentences in a discussion thread. One of the popular methods in structured prediction problem is "Conditional Random Fields" (CRF). They belong to a class of undirected graphical models where each node represents a random variable and edge denotes dependency among the connected random variables. In CRF, the nodes are divided into two distinct sets - input features "X" (also called an observed variable)and output labels "Y". We can then model the conditional distribution p(Y | X) using maximum likelihood learning of parameters. The optimization problem is often convex and can be solved using a gradient descent algorithm like lbfgs. In case of discourse anlaysis, each message is dependent on the preceding and succeeding messages giving rise to the linear chain-like structure.

Worker Implementation

The feature vector consists of semantic and structural features. The messages are preprocessed and are converted into a vector using TF-IDF vectorizer. The lexical and structural features are added before grouping them based on conversation thread. sklearn-crfsuite library for building CRF model. The library provides scikitlearn compatible interface. The worker uses the conditional random field model pretrained with labeled Reddit data to make predictions on GitHub data.

In addition to the python files defining the worker, the worker directory consists of following model files

- **trained_crf_model** - the trained crf model which can be used to predict sequential discourse acts
- **tfidf_transformer** - the trained tfidf transformer can be used to convert text into feature vector
- **word_to_emotion_map** - the dictionary provides mapping from word to label signifying particular emotion

Worker Configuration

Like standard worker configuration, we need to define delay, given, model and repo_group_id in housekeeper configuration block.

```
{
    "delay": 10000,
    "given":["git_url"],
    "model" : "discourse_analysis",
    "repo_group_id" : 60003
}
```

Further, in workers configuration block, we need to define port, switch and number of workers.

```
"discourse_analysis_worker":{
            "port" : 51400,
            "switch": 1,
            "workers": 1
        }
```

Insight_worker

It uses Augur's metrics API to discover insights for every time_series metrics like `issues`, `reviews`, `code-changes`, `code-changes-lines` etc.. of every repos present in the database.

We used BiLSTM(Bi-directional Long Short Term Memory)model as it is capable of capturing trend, long-short seasonality in the data. A bidirectional LSTM (BiLSTM) layer learns bidirectional long-term dependencies between time steps of time series or sequence data. These dependencies can be useful when you want the network to learn from the complete time series at each time step.

Worker Configuration

Worker has three main configurations that are standard across all workers.And it also have few more configurations that are mainly for Machine Learning model inside the worker.

The standard options are:

- `switch`, a boolean flag indicating if the worker should automatically be started with Augur. Defaults to `0` (false).
- `workers`, the number of instances of this worker that Augur should spawn if `switch` is set to `1`. Defaults to `1`.
- `port`, which is the base TCP port the worker will use t0 communicate with Augur's broker. The default is different for each worker, for the `insight_worker` it is `21311`.

Keeping workers at 1 should be fine for small collection sets, but if you have a lot of repositories to collect data for, you can raise it. We also suggest double checking that the default worker ports are free on your machine.

Configuration for ML models are:

We recommend leaving the defaults in place for the insight worker unless you interested in other metrics, or anomalies for a different time period.

- `training_days`, which specifies the date range that the `insight_worker` should use as its baseline for the statistical comparison. Defaults to `365`, meaning that the worker will identify metrics that have had anomalies compared to their values over the course of the past year, starting at the current date.
- `anomaly_days`, which specifies the date range in which the `insight_worker` should look for anomalies. Defaults to `14`, meaning that the worker will detect anomalies that have only occured within the past fourteen days, starting at the current date.
- `contamination`, which is the "sensitivity" parameter for detecting anomalies. Acts as an estimated percentage of the training_days that are expected to be anomalous. The default is `0.1` for the default training days of 365: 10% of 365 days means that about 36 data points of the 365 days are expected to be anomalous.
- `metrics`, which specifies which metrics the `insight_worker` should run the anomaly detection algorithm on. This is structured like so:

```
[
    'endpoint_name_1',
    'endpoint_name_1',
    'endpoint_name_2',
    ...
]

# defaults to the following

[
    "issues-new",
```

(continues on next page)

(continued from previous page)

```
    "code-changes",
    "code-changes-lines",
    "reviews",
    "contributors-new"
]
```

Methods inside the Insight_model

- `time_series_metrics`:It takes parameters `entry_info` , `repo_id` .Collects data of different metrics using API endpoints.Preprocesses data and creates a dataframe with date and each and every fields of the given endpoints as columns.Then this method calls another method that is `lstm_selection`.Structure of the dataframe is as follows:

```
df>>
index   date          endpoints1 _ field     endpoints2 _ field
0.      2020-03-20    5                      8
```

- `lstm_selection`:This method takes `entry_info`, `repo_id`, `df` as parameters.It Selects window_size or time_steps by checking sparsity and coefficient of variation in data which is passed into the `lstm_keras` method.
- `preprocess_data`:This method is called by the `lstm_keras` method with `data`, `tr_days`, `lback_days`, `n_features`, `n_predays` as parameters.It arranges training_data according to different parameters passed by `lstm_keras` method for the `BiLSTM` model.It returns two variables `features_set` and `labels` with the following structure:

```
features_set = [
    [[1],
     [2]],
    [[2],
     [3]],
    [[3],
     [4]]
]
labels = [ [3],[4],[5] ]

#tr_days : number of training days(it is not equal to the training days passed␣
↪into the configuration)
#lback_days : number of days to lookback for next-day prediction
#n_features : number of features of columns in dataframe for training
#n_predays : next number of days to predict for each entry in features_set


#tr_days = training_days - anomaly_days   (in configuration)

#here
tr_days = 4,
black_days = 2,
n_features = 1,
n_predays = 1
```

- `lstm_model`:It is the configuration of the multiple `BiLSTM` layers along with single `dense` layer and optimisers.This method called inside the `lstm_keras` method with `features_set`, `n_predays`, `n_features` as pa-

rameters.Configuation of the model is as follows:

```
model = Sequential()
model.add(Bidirectional(LSTM(90, activation='linear',return_sequences=True,
↪input_shape=(features_set.shape[1], n_features))))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(90, activation='linear',return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(90, activation='linear')))
model.add(Dense(1))
model.add(Activation('linear'))
model.compile(optimizer='adam', loss='mae')
```

This configuration is designed to acheive the best possible results for all kind of metrics.

- `lstm_keras`:This is the most important method in the `insights_model` called by the `lstm_selection` method with `entry_info` , `repo_id`, and `dataframe` as parameters.Here dataframe consists of two columns, one is `date` and another one is `endpoint1 _ field` .In this method model is trained on `tr_days` data and values were predicted for `anomaly_days` data.Baesd on the difference on actual and predicted values outliers were discovered.

  If any outliers discovered between the `anomaly_days` then those points will be inserted into to the `repo_insights` and `repo_insights_records` table by calling `insert_data` method.

  Before calling the `insert_data` method, performance of model on the training as well as test data will be evaluated and its summary will be inserted into the `lstm_anomaly_results` table along with the unique model configuration into the `lstm_anomaly_models` table.

- `insert_data`:It is called by the `lstm_keras` method with `entry_info`, `repo_id`, `anomaly_df`, `model` as parameters.Here `anomaly_df` is the dataframe which consists of points which are classified as outliers between the `anomaly_days`.

Insights_model consists of multiple independent methods like `time_series_metrics`, `insert_data` etc..These methods can be used independently with other Machine Learning models.Also `preprocess_data`, `model_lstm` methods can be easily modified according to the different LSTM networks configuration.

Message Insights Worker

Note:

- If you have an NVidia GPU available, you can install the *cuda* drivers to make this worker run faster.
- On Ubuntu 20.04, use the following commands: - On the Ubuntu machine, open a Terminal. Type in the following commands to add the Nvidia ppa repository: - *wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin - sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600 && sudo apt-key adv –fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/7fa2af80.pub - sudo add-apt-repository "deb https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/ /" - sudo apt-get update && sudo apt-get install -y nvidia-kernel-source-460 - sudo apt-get -y install cuda*

This worker analyzes the comments and text messages corresponding to all the issues and pull requests in a repository and performs two tasks:

- **Identifies novel messages** - Detects if a new message is semantically different from past messages in a repo
- **Performs sentiment analysis** - Assigns a score to every message indicating positive/negative/neutral sentiments

Worker Configuration

To kickstart the worker, it needs to receive a task from the Housekeeper, similar to other workers, and have a corresponding worker specific configuration.

The standard options are:

- `switch` - a boolean flag indicating if the worker should automatically be started with Augur. Defaults to `0` (false).
- `workers` - the number of instances of this worker that Augur should spawn if `switch` is set to `1`. Defaults to `1`.
- `port` - the TCP port the worker will use to communicate with Augur's broker, the default being `51300`.
- `insight_days` - the most recent period (in days) for which insights would be calculated.
- `models_dir` - the directory within the worker directory, where all trained machine learning models would be stored.

Note:

- `insight_days` can be kept at roughly 20-30 as the sentiment, or novelty do not fluctuate very fast!
- The `models_dir` would collect the trained ML models. A different model would be saved per repo for performing novelty detection and a common sentiment analysis model across all repos. Do not accidentally delete this, as subsequent runs leverage these!

Worker Pipeline

When a repo is being analyzed for the first time, the ML models are trained and saved on disk in the `models_dir` specified in the config block. During subsequent runs, these models are used to predict directly. The worker contains the files — `preprocess_text.py`, `message_novelty.py`, and `message_sentiment.py` to get insights. The `train_data` directory contains training files that are utilized by the worker when running for the first time.

```
message_insights_worker/
├── __init__.py
├── runtime.py
├── setup.py
├── message_insights_worker.py
├── message_novelty.py
├── message_sentiment.py
├── preprocess_text.py
├── message_models/
    ├── 27940_uniq.h5
    ├── tfidf_vectorizer.pkl
    └── XGB_senti.pkl
└── train_data/
    ├── custom_dataset.xlsx
    ├── doc2vec.model
    └── EmoticonLookupTable.txt
```

The `custom_dataset.xlsx` is a dataset containing technical code review and commit messages from Oracle & Jira database, and general sentiment, emoji-filled messages from StackOverflow. `EmoticonLookupTable` is used in sentiment analysis to parse emojis/emoticons and capture their sentiment. `doc2vecmodel` is the trained Doc2Vec model on the custom_dataset, utilized for forming word embeddings in novelty detection.

Novelty Detection

Novelty detection is an unsupervised classification problem to identify abnormal/outlier messages. Since a message that is novel in the context of 1 repo may be normal to another, it is essential to maintain the semantics of the past messages of every repo, to flag an incoming message as novel/normal accurately. Every message is transformed to a 250-dimensional Doc2Vec embedding. Deep Autoencoders are used to compress input data and calculate error thresholds. The architecture of the autoencoder model consists of two symmetrical deep neural networks — an encoder and a decoder that applies backpropagation, setting the target values to be equal to the inputs. It attempts to copy its input to its output and anomalies are harder to reconstruct compared with normal samples. 2 AEs are used, the first one identifies normal data based on the threshold calculated and then the second is trained using only the normal data. The Otsu thresholding technique is used in these steps to get the error threshold. Messages which are at least 10 characters long and have reconstruction error > threshold, get flagged as novel.

Sentiment Analysis

The sentiment analysis was modeled as a supervised problem. This uses the SentiCR tool with modifications to improved preprocessing, emoji parsing, and predicting. Messages are first cleaned using an 8 step preprocessing method. The messages are converted into embeddings using TF-IDF vectorization and then sent to an XGBoost classifier, which assigns sentiment labels.

Insights

After these tasks are done, the `message_analysis` table is populated with all details are about a message: the id and the sentiment and novelty scores. In order to get a view of these metrics at a repo level, the `message_analysis_summary` table is updated with the total ratio of positive/negative sentiment messages and count of novel messages every `insight_days` frequency.

The 3 types of insights provided are:

- Counts of positive/negative sentiment and novel messages
- Mean deviation of these in the most recent analyzed with the past to understand trends
- A list of timestamps which indicate possible anomaly durations with respect to sentiment trends

These are also sent to Auggie.

Pull Request Analysis Worker

This worker analyzes the open pull requests of every repository and predicts the probability of it getting accepted and merged. Pull requests having a low probability of getting merged, are indicative of being outlier/anomalous ones.

Worker Configuration

To kickstart the worker, it needs to receive a task from the Housekeeper, similar to other workers, and have a corresponding worker specific configuration.

The standard options are:

- `switch` - a boolean flag indicating if the worker should automatically be started with Augur. Defaults to `0` (false).
- `workers` - the number of instances of this worker that Augur should spawn if `switch` is set to `1`. Defaults to `1`.
- `port` - the TCP port the worker will use to communicate with Augur's broker, the default being `51400`.

- `insight_days` - open PRs created in the duration of the past `x` days are analyzed.

Note:

- `insight_days` can be adjusted to analyze very recently opened PRs as well. Run the data collection workers to have enough data to analyze!
- This worker uses some methods of the *Message Insights Worker*

Worker Pipeline

When a repo is analyzed, the trained ML models are used to predict the probability of acceptance. The major factors influencing this are:

1. **Pull request characteristics**: No. of commits, sentiment of PR title
2. **Contributor characteristics**: Acceptance rate of PRs created, no. of projects contributed to in the past
3. **Repo characteristics**: No. of open issues, no. of watchers, past acceptance rate of PRs
4. **Discussion characteristics**: No. of comments, no. of participants, average sentiment of all comments

After feature engineering, the following features were considered:

- No. of commits
- No. of comments
- Length of PR in days
- Relationship between PR creator and repo (member, collaborator, etc)
- Average sentiment score of comments
- Watch count of repo
- Past acceptance ratio of a PR in the repo

```
pull_request_analysis_worker/
├── __init__.py
├── pull_request_analysis_worker.py
├── runtime.py
├── setup.py
└── trained_pr_model.pkl
```

The `trained_pr_model.pkl` is the saved pre-trained model, used for the analysis.

After prediction, the `pull_request_analysis` table is populated with the predicted probabilities for every open PR.

Gitlab Merge Request Worker

Models & Tables they populate-

1. merge_requests: This model deals with the data related to Merge Requests(MR) of a project. The tables populated are mentioned below-

    1.1. pull_requests: The data relating to each MR is stored in this table.

    1.2. pull_request_labels: Stores Labels of each MR.

    1.3. pull_request_assignees: Stores the assignees of the MR.

    1.4. pull_request_reviewers: Stores the list of all possible reviewers for the MR.

    1.5. pull_request_events: Events like merged, closed, etc for each MR with their timestamp are stored in this table.

    1.6. pull_request_meta: Stores meta data of the MR, like the info of the base.

    1.7. pull_request_message_ref: Stores reference data for each message.

    1.8. pull_request_repo: Stores the data related to the project to which the MR belongs.

    1.9. message: Stores the messages on the MR thread.

    1.10: contributors: Stores the information related to each contributor of the project. Its implementation is in the base worker class. *query_gitlab_contributors*

2. merge_request_commits: This model deals with the commit data of each MR-

    2.1. pull_request_commits: Commits and their details are stored in this table along with the MR ID for mapping.

3. merge_request_files: This model deals with the details of changes made in a file.

    3.1. pull_request_files: Stores details of changes made in each file

This worker has an architecture same as the Pull Request Worker. Whenever you send a task for any model, it hits the API endpoints to fetch the data. Duplicates are ignored and only upsert operations (Update/Insert) are performed. The tables in which the data is populated are common for both Github Pull Request Worker & Gitlab Merge Request Worker. The tables and the columns are made with respect to the Github API naming and logic as initially Augur only supported Github worker. But it is easy to understand the mapping between the naming conventions of Github & Gitlab API.

There are only a few considerable differences between the APIs:

1. Project can be owned by a group of people in Gitlab while in Github, there is always a unique repo owner (basically the one who created the repo). Repo could be owned by an organization but the Github API values the repo creator and returns the creator as the unique owner. Although Gitlab doesn't differentiate between the group of owners and the project creator, we explicitly store the Gitlab project creator as the unique owner.

2.Gitlab API returns the email addresses of closed Gitlab accounts but they don't have a unique source id associated with them. They have a login/username which is enough for our use-case.

3.Gitlab allows 10X more requests per minute than Github API, so you may not need to store multiple API keys in the worker_oauth table.

Gitlab Issues Worker - Populated Models

1. issues: This model deals with the data related to the issues of a project. The tables populated are mentioned below-

    1.1. issues: The data related to each issue(issue name, gitlab issue id, date_created, etc..) is stored in this table.

    1.2. issue_labels: Stores Labels of each issue.

    1.3. issues_assignees: Stores the assignees of the issue.

    1.4. issue_messages: Stores all of the comments associated with a particular issue.

    1.5. issue_events: Events like opened, closed, etc for each issue are stored in this table.

This worker has an architecture similar to that of the Gitlab Issues Worker. Whenever you send a task for issue collection, it hits the API endpoints to fetch the data. Duplicates are ignored and only upsert operations (Update/Insert) are performed. The issues model acts as a central repository for Github & Gitlab issue workers. Some of the columns present in the tables might be a bit off with respect to the Gitlab Worker, but easily perceptible.

1.5.6 Create a Metric

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

Steps to Create a Metric API Endpoint

Summary

There are many paths, but we usually follow something along these lines:

1. What is the CHAOSS metric we want to develop?
2. Sometimes, there are metrics endpoints that integrate, or visualize several metrics.
3. Determine what tables in the Augur Schema contain the data we need to develop this metric
4. Construct a very basic query that does the work of joining those tables in a minimal way so we have a "baseline query."
5. Refine the query so that it takes the standard inputs for a "standard metric" if that's what type it is; alternatively, look at non-standard metrics as they are defined in `AUGUR_HOME/augur/routes`, or one of the visualization metrics in `AUGUR_HOME/augur/routes/contributor.py`, `AUGUR_HOME/augur/routes/pull_requests.py` or `AUGUR_HOME/augur/routes/nonstandard_metrics.py`. (This step is explained in the next section.)

Example Query

This is an example query to Get Us Started on a Labor Effort and Cost Endpoint.

1. What tables?

```
repo
repo_group
```

If we look at the Augur Schema, we can see that effort and cost are contained in the `repo_labor` table.

2. What might our initial query to explore building the endpoint be?

```
SELECT C.repo_id,
    C.repo_name,
    programming_language,
    SUM ( estimated_labor_hours ) AS labor_hours,
    SUM ( estimated_labor_hours * 50 ) AS labor_cost,
    analysis_date
    FROM
    (
            SELECT A
                    .repo_id,
                    b.repo_name,
                    programming_language,
                    SUM ( total_lines ) AS repo_total_lines,
                    SUM ( code_lines ) AS repo_code_lines,
                    SUM ( comment_lines ) AS repo_comment_lines,
                    SUM ( blank_lines ) AS repo_blank_lines,
                    AVG ( code_complexity ) AS repo_lang_avg_code_complexity,
                    AVG ( code_complexity ) * SUM ( code_lines ) + 20 AS estimated_
↪labor_hours,
                    MAX ( A.rl_analysis_date ) AS analysis_date
            FROM
                    repo_labor A,
                    repo b
            WHERE
                    A.repo_id = b.repo_id
            GROUP BY
                    A.repo_id,
                    programming_language,
                    repo_name
            ORDER BY
                    repo_name,
                    A.repo_id,
                    programming_language
            ) C
    GROUP BY
            repo_id,
            repo_name,
            programming_language,
            C.analysis_date
    ORDER BY
            repo_id,
            programming_language;
```

3. Over time, as CHAOSS develops a metric for labor investment, the way we calculate hours, and cost in this query will adapt to whatever the CHAOSS community determines is an apt formula.

4. We will fit this metric into one of the different types of metric API Endpoints discussed in the next section.

Note: Augur uses https://github.com/boyter/scc to calculate information contained in the `labor_value` table, which is populated by the `value_worker` tasks.

Parts of an Augur API Endpoint

1. Develop the query that will produce data needed to deliver the endpoint, usually parameterized by a *repo_id*.
2. Determine if the endpoint will be a "standard endpoint", or a custom endpoint.

Where Are the Endpoints?

JSON Metrics are here:

```
$ AUGUR_HOME/augur/metrics
```

Visualization Metrics are here:

```
$ AUGUR_HOME/augur/routes
```

Existing metrics files (JSON Metric) "Standard Metrics":

1. commit.py
2. contributor.py
3. experimental.py
4. insight.py
5. issue.py
6. message.py
7. platform.py
8. release.py
9. repo_meta.py

All "Standard Metrics" files generally share a set of imports

```
import datetime
import sqlalchemy as s
import pandas as pd
from augur.util import register_metric
```

You can see that one of the imports is our standard metric import from the util file, which is located in:

```
AUGUR_HOME/augur/routes/util.py
```

All "Standard Metrics" share declaration and a method signature

Declaration:

```
@register_metric()
```

Method Signature:

```
def contributors(self, repo_group_id, repo_id=None, period='day', begin_date=None, end_
↪date=None):
```

Standard metrics also, generally, include default setting blocks for date range, in the event parameters are not passed.

```
if not begin_date:
    begin_date = '1970-1-1 00:00:01'
if not end_date:
    end_date = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

There is also, generally, a block in a standard metric for pulling data by a repo_id or a repo_group_id. The default is a repo_group_id. Here is an abrdiged example from the contributors endpoint in *contributor.py*

```
if repo_id:
    contributorsSQL = s.sql.text("""
       SELECT id                          AS user_id,
            SUM(commits)                AS commits,
            SUM(issues)                 AS issues,
            SUM(commit_comments)        AS commit_comments,
            SUM(issue_comments)         AS issue_comments,
            SUM(pull_requests)          AS pull_requests,
            SUM(pull_request_comments)  AS pull_request_comments,
            SUM(a.commits + a.issues + a.commit_comments + a.issue_comments + a.pull_
↪requests +
                a.pull_request_comments) AS total,
            a.repo_id, repo.repo_name
        FROM (
                (SELECT gh_user_id AS id,
                    ommitted_lines as ommitted_from_example
                    AND created_at BETWEEN :begin_date AND :end_date
                    GROUP BY id, repo_id
                )
            ) a, repo
        WHERE a.repo_id = repo.repo_id
        GROUP BY a.id, a.repo_id, repo_name
        ORDER BY total DESC
    """)

    results = pd.read_sql(contributorsSQL, self.database, params={'repo_id': repo_id,
↪'period': period,
                                                                  'begin_date': begin_date,
↪'end_date': end_date})
else: ## This is if the repo_id is not specified
    contributorsSQL = s.sql.text("""
       SELECT id                          AS user_id,
            SUM(commits)                AS commits,
            SUM(issues)                 AS issues,
            SUM(commit_comments)        AS commit_comments,
            SUM(issue_comments)         AS issue_comments,
            SUM(pull_requests)          AS pull_requests,
            SUM(pull_request_comments)  AS pull_request_comments,
            SUM(a.commits + a.issues + a.commit_comments + a.issue_comments + a.pull_
↪requests +
                a.pull_request_comments) AS total, a.repo_id, repo_name
        FROM (
                (SELECT gh_user_id AS id,
                        repo_id,
```

(continues on next page)

(continued from previous page)

```
                    0          AS commits,
                    COUNT(*)   AS issues,
                    0          AS commit_comments,
                AND created_at BETWEEN :begin_date AND :end_date
                GROUP BY id, repo_id
                ommitted_lines as ommitted_from_example
            )
        ) a, repo
    WHERE a.repo_id = repo.repo_id
    GROUP BY a.id, a.repo_id, repo_name
    ORDER BY total DESC
""")

    results = pd.read_sql(contributorsSQL, self.database, params={'repo_group_id': repo_
↪group_id, 'period': period,
                                                                  'begin_date': begin_date,
↪'end_date': end_date})
return results
```

Existing Visualization Metrics Files:

1. augur/routes/contributor_reports.py
2. augur/routes/pull_request_reports.py

Existing Metrics Files:

1. augur/metrics/commit.py
2. augur/metrics/contributor.py
3. augur/metrics/deps.py
4. augur/metrics/experimental.py
5. augur/metrics/insight.py
6. augur/metrics/issue.py
7. augur/metrics/message.py
8. augur/metrics/platform.py
9. augur/metrics/pull_request.py
10. augur/metrics/release.py
11. augur/metrics/repo_meta.py

These files are not intended to be all inclusive. Rather, they are what we have developed, or imagined, based on existing CHAOSS metrics to date. New CHAOSS metrics are likely to result in the inclusion of new files under metrics, or routes, depending if they are standard metrics or not.

1.5.7 CLI (Command Line Interface)

To learn what's available in Augur's command line interface (CLI), simply use this set of commands:

```
augur --help
augur db --help
augur backend --help
augur config --help
augur logging --help
```

If you have questions or would like to help please open an issue on GitHub.

1.5.8 Testing

THIS SECTION IS UNDER CONSTRUCTION.

If you have questions or would like to help please open an issue on GitHub.

1.5.9 Configuration file reference

Augur's configuration template file, which generates your locally deployed `augur.config.json` file, is found at `augur/config.py`. You will notice a small collection of workers are turned on to start with, by examining the `switch` variable within the `Workers` block of the config file. You can also specify the number of processes to spawn for each worker using the `workers` command. The default is one, and we recommend you start here. If you are going to spawn multiple workers, be sure you have enough credentials cached in the `augur_operations.worker_oath` table for the platforms you use.

If you have questions or would like to help please open an issue on GitHub.

1.6 REST API Documentation

`GET /repo-groups`

Get all the downloaded repo groups.

Status Codes

- 200 OK – OK

`GET /repos`

Get all the downloaded repos.

Status Codes

- 200 OK – OK

`GET /repos/:id`

Get a downloaded repo by its ID in Augur. The schema block below says it is an array, but it is not.

Status Codes

- 200 OK – OK

`GET /owner/:owner/repo/:repo`

Get the repo_group_id and repo_id of a particular repo.

Parameters

- **owner** (*string*) – Repo Owner
- **repo** (*string*) – Repo Name

Status Codes

- 200 OK – OK

`GET /rg-name/:rg_name/repo-name/:repo_name`

Get the repo_group_id and repo_id of a particular repo.

Parameters

- **rg_name** (*string*) – Repo Group Name
- **repo_name** (*string*) – Repo Name

Status Codes

- 200 OK – OK

`GET /rg-name/:rg_name`

Get the repo_id of a particular repo group.

Parameters

- **rg_name** (*string*) – Repo Group Name

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/repos`

Get all the repos in a repo group.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/top-insights`

Get all the downloaded repo groups.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /metadata/repo_info`

Returns the metadata about all repositories in an Augur instance, using information from a git platform (GitHub, GitLab, etc.). Each record includes the default branch name, repository license file, forks, stars, watchers, and committers. Also includes metadata about current repository issue and pull request status and counts.

Status Codes

- 200 OK – OK

`GET /metadata/contributions_count`

Returns a list of repositories contributed to by all the contributors in an Augur Instance: INCLUDING all repositories on a platform, *not* merely those repositories in the Augur Instance. Numerical totals represent total CONTRIBUTIONS.

Status Codes

- 200 OK – OK

`GET /metadata/contributors_count`

Returns a list of repositories contributed to by all the contributors in an Augur Instance: INCLUDING all repositories on a platform, *not* merely those repositories in the Augur Instance. Numerical totals represent total CONTRIBUTORS.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/average-issue-resolution-time`

The average issue resolution time.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/average-issue-resolution-time`

The average issue resolution time.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/cii-best-practices-badge`

The CII Best Practices Badge level.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/cii-best-practices-badge`

The CII Best Practices Badge level.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/committers`

Number of persons opening an issue for the first time.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/committers`

Number of persons contributing with an accepted commit for the first time.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/fork-count`

Fork count.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/fork-count`

Fork count.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/forks`

A time series of fork count.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/forks`

A time series of fork count.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/license-coverage`

Number of persons opening an issue for the first time.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/license-coverage`

Number of persons contributing with an accepted commit for the first time.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/license-declared`

An enumeration of all the licenses declared in a repo group at the file level.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/license-declared`

An enumeration of all the licenses declared in a repo at the file level.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/languages`

The primary language of the repository.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/languages`

The primary language of the repository.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/license-count`

The declared software package license (fetched from CII Best Practices badging data).

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/license-count`

The declared software package license (fetched from CII Best Practices badging data).

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/aggregate-summary`

Returns the current count of watchers, stars, and forks and the counts of all commits, committers, and pull requests merged between a given beginning and end date (default between now and 365 days ago).

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/aggregate-summary`

Returns the current count of watchers, stars, and forks and the counts of all commits, committers, and pull requests merged between a given beginning and end date (default between now and 365 days ago).

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/annual-commit-count-ranked-by-new-repo-in-repo-group`

Annual commit count ranked by new repos in a repo group.

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/annual-commit-count-ranked-by-new-repo-in-repo-group`

Annual commit count ranked by new repos.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/annual-commit-count-ranked-by-repo-in-repo-group`

Annual commit count ranked by repos in a repo group.

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **repo_url_base** (*String*) – Base64 version of the URL of the GitHub repository as it appears in the Facade DB

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/annual-commit-count-ranked-by-repo-in-repo-group`

This is an Augur-specific metric. We are currently working to define these more formally.

Parameters

- **repo_id** (*string*) – Repository ID.
- **repo_url_base** (*String*) – Base64 version of the URL of the GitHub repository as it appears in the Facade DB

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/annual-lines-of-code-count-ranked-by-new-repo-in-repo-group`

This is an Augur-specific metric. We are currently working to define these more formally.

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **repo_url_base** (*String*) – Base64 version of the URL of the GitHub repository as it appears in the Facade DB

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/annual-lines-of-code-count-ranked-by-new-repo-in-repo-group`

This is an Augur-specific metric. We are currently working to define these more formally.

Parameters

- **repo_id** (*string*) – Repository ID.
- **repo_url_base** (*String*) – Base64 version of the URL of the GitHub repository as it appears in the Facade DB

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/annual-lines-of-code-count-ranked-by-repo-in-repo-group`

This is an Augur-specific metric. We are currently working to define these more formally.

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **repo_url_base** (*String*) – Base64 version of the URL of the GitHub repository as it appears in the Facade DB

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/annual-lines-of-code-count-ranked-by-repo-in-repo-group`

This is an Augur-specific metric. We are currently working to define these more formally.

Parameters

- **repo_id** (*string*) – Repository ID.
- **repo_url_base** (*String*) – Base64 version of the URL of the GitHub repository as it appears in the Facade DB

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issue-comments-mean`

Mean(Average) of issue comments per day.

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **group_by** (*string*) – Allows for results to be grouped by day, week, month, or year. E.g. values: year, day, month

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issue-comments-mean`

Mean(Average) of issue comments per day.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issue-comments-mean-std`

Mean(Average) and Standard Deviation of issue comments per day.

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **group_by** (*string*) – Allows for results to be grouped by day, week, month, or year. E.g. values: year, day, month

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issue-comments-mean-std`

Mean(Average) and Standard Deviation of issue comments per day.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/pull-request-acceptance-rate`

Timeseries of pull request acceptance rate (expressed as the ratio of pull requests merged on a date to the count of pull requests opened on a date)

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **group_by** (*string*) – Allows for results to be grouped by day, week, month, or year. E.g. values: year, day, month

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/pull-request-acceptance-rate`

Timeseries of pull request acceptance rate (expressed as the ratio of pull requests merged on a date to the count of pull requests opened on a date)

Parameters

- **repo_id** (*string*) – Repository ID.

- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/pull-requests-new`

Returns a time series of the number of new Pull Requests opened during a certain period for a specific repository group.

Parameters

- **repo_group_id** (*string*) – The repository group's id.

Query Parameters

- **period** (*string*) – Sets the periodicity to 'day', 'week', 'month', or 'year'.
- **begin_date** (*string*) – Specifies the begin date.
- **end_date** (*string*) – Specifies the end date.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/pull-requests-new`

Returns a time series of the number of new Pull Requests opened during a certain period for a specific repository.

Parameters

- **repo_id** (*string*) – The repository's id.

Query Parameters

- **period** (*string*) – Sets the periodicity to 'day', 'week', 'month', or 'year'.
- **begin_date** (*string*) – Specifies the begin date.
- **end_date** (*string*) – Specifies the end date.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/pull-requests-closed-no-merge`

Timeseries of pull request which were closed but not merged

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/pull-requests-closed-no-merge`

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

—

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/top-committers`

Returns a list of contributors contributing N% of all commits.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **year** (*string*) – Specify the year to return the results for. Default value: current year
- **threshold** (*string*) – Specify N%. Accepts a value between 0 & 1 where 0 specifies 0% and 1 specifies 100%.

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/top-committers`

Returns a list of contributors contributing N% of all commits.

Parameters

- **repo_id** (*string*) – Repository ID.
- **year** (*string*) – Specify the year to return the results for. Default value: current year
- **threshold** (*string*) – Specify N%. Accepts a value between 0 & 1 where 0 specifies 0% and 1 specifies 100%.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/abandoned-issues`

List of abandoned issues (last updated >= 1 year ago)

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/abandoned-issues`

List of abandoned issues (last updated >= 1 year ago)

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/lines-changed-by-author`

Count of closed issues.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/lines-changed-by-author`

Count of closed issues.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/code-changes`

Time series of number of commits during a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/code-changes`

Time series number of commits during a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/code-changes-lines`

Time series of lines added and removed during a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.

- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/code-changes-lines`

Time series of lines added and removed during a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/contributors`

List of contributors and their contributions.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/contributors`

List of contributors and their contributions.

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/contributors-new`

Time series of number of new contributors during a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.

- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/contributors-new`

Time series of number of new contributors during a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issue-backlog`

Number of issues currently open.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issue-backlog`

Time since an issue is proposed until it is closed.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issue-participants`

How many persons participated in the discussion of issues.

Parameters

- **repo_group_id** (*string*) – Repository Group ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issue-participants`

How many persons participated in the discussion of issues.

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issue-throughput`

Ratio of issues closed to total issues.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issue-throughput`

Ratio of issues closed to total issues.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-active`

Time series of number of issues that showed some activity during a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-active`

Time series of number of issues that showed some activity during a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01

- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-closed`

Time series of number of issues closed during a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-closed`

Time series of number of issues closed during a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-closed-resolution-duration`

Duration of time for issues to be resolved.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-closed-resolution-duration`

Duration of time for issues to be resolved.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-first-time-closed`

Number of persons closing an issue for the first time.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-first-time-closed`

Number of persons closing an issue for the first time.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-first-time-opened`

Number of persons opening an issue for the first time.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-first-time-opened`

Number of persons opening an issue for the first time.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-maintainer-response-duration`

Duration of time for issues to be resolved.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-maintainer-response-duration`

Duration of time for issues to be resolved.

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-new`

Time series of number of new issues opened during a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-new`

Time series of number of new issues opened during a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issues-open-age`

Age of open issues.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/issues-open-age`

Age of open issues.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/pull-requests-merge-contributor-new`

Number of persons contributing with an accepted commit for the first time.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/pull-requests-merge-contributor-new`

Number of persons contributing with an accepted commit for the first time.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/review-duration`

Time since an review/pull request is proposed until it is accepted.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/review-duration`

Time since an review/pull request is proposed until it is accepted.

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/reviews`

Time series of number of new reviews / pull requests opened within a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/reviews`

Time series of number of new reviews / pull requests opened within a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/reviews-accepted`

Time series of number of accepted reviews / pull requests opened within a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/reviews-accepted`

Time series of number of accepted reviews / pull requests opened within a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/reviews-declined`

Time series of number of declined reviews / pull requests opened within a certain period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/reviews-declined`

Time series of number of declined reviews / pull requests opened within a certain period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **period** (*string*) – Periodicity specification.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/sub-projects`

Number of sub-projects.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/sub-projects`

Number of sub-projects.

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/closed-issues-count`

Count of closed issues.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/closed-issues-count`

Count of closed issues.

Parameters

- **repo_id** (*string*) – Repository ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/issue-duration`

Time since an issue is proposed until it is closed.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/issue-duration`

Time since an issue is proposed until it is closed.

Parameters

- **repo_group_id** (*string*) – Repository ID
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_group_id/releases`

Time since an review/pull request is proposed until it is accepted.

Parameters

- **repo_id** (*string*) – Repository Group ID.

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/releases`

Time since an review/pull request is proposed until it is accepted.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/open-issues-count`

Count of open issues.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_id/open-issues-count`

Count of open issues.

Parameters

- **repo_id** (*string*) – Repository ID

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/watchers`

A time series of watchers count.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/watchers`

A time series of watchers count.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/watchers-count`

Watchers count.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/watchers-count`

Watchers count.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/stars`

A time series of stars count.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/stars`

A time series of stars count.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/stars-count`

Stars count.

Parameters

- **repo_group_id** (*string*) – Repository Group ID

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/stars-count`

Stars count.

Parameters

- **repo_id** (*string*) – Repository ID.

Status Codes

- 200 OK – OK

`GET /contributor_reports/new_contributors_bar/`

Get bar chart of new contributor counts.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **group_by** (*string*) – How the data is grouped. OPTIONS – "Month", "Year", or "Quarter". DEFAULT VALUE – Quarter
- **required_contributions** (*integer*) – Number of contributions required for a user to be a repeat contributor. DEFAULT VALUE – 4
- **required_time** (*integer*) – Amount of time in days that the user must have completed the required contributons to be a repeat contributor. DEFAULT VALUE – 365
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /contributor_reports/new_contributors_stacked_bar/`

Get stacked bar chart of new contributor counts, which shows the action of the contributor.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **group_by** (*string*) – How the data is grouped. OPTIONS – "Month", "Year", or "Quarter". DEFAULT VALUE – Quarter

- **required_contributions** (*integer*) – Number of contributions required for a user to be a repeat contributor. DEFAULT VALUE – 4
- **required_time** (*integer*) – Amount of time in days that the user must have completed the required contributons to be a repeat contributor. DEFAULT VALUE – 365
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /contributor_reports/returning_contributors_pie_chart/`

Get pie chart of the returning contributor counts.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **required_contributions** (*integer*) – Number of contributions required for a user to be a repeat contributor. DEFAULT VALUE – 4
- **required_time** (*integer*) – Amount of time in days that the user must have completed the required contributons to be a repeat contributor. DEFAULT VALUE – 365
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /contributor_reports/returning_contributors_stacked_bar/`

Get returning contributors stacked bar chart.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **group_by** (*string*) – How the data is grouped. OPTIONS – "Month", "Year", or "Quarter". DEFAULT VALUE – Quarter
- **required_contributions** (*integer*) – Number of contributions required for a user to be a repeat contributor. DEFAULT VALUE – 4
- **required_time** (*integer*) – Amount of time in days that the user must have completed the required contributons to be a repeat contributor. DEFAULT VALUE – 365
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/average_commits_per_PR/`

Get vertical bar chart of average commits per pull request.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/average_comments_per_PR/`

Get horizontal stacked bar chart of the average comments for closed pull requests.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/PR_counts_by_merged_status/`

Get vertical stacked bar chart of the number of pull requests by merged status.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/mean_response_times_for_PR/`

Get grouped horizontal bar chart of average response times for closed pull requests.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/mean_days_between_PR_comments/`

Get line graph of the average days between comments for closed pull requests.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/PR_time_to_first_response/`

Get scatter plot of the days to first response for closed pull requests based on merged status.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **remove_outliers** (*boolean*) – Whether outliers will be removed or not. If true values outside of 3 standard deviations will be removed. DEFAULT VALUE – true
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/average_PR_events_for_closed_PRs/`

Get heat map of average pull request event counts for closed pull requests.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)

- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /pull_request_reports/Average_PR_duration/`

Get heat map of the average duration of pull requests per month.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)
- **start_date** (*string*) – Minimum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – 1 year in the past
- **end_date** (*string*) – Maximum date for the graph. INPUT FORMAT – YYYY-MM-DD. DEFAULT VALUE – Current date
- **remove_outliers** (*boolean*) – Whether outliers will be removed or not. If true values outside of 3 standard deviations will be removed. DEFAULT VALUE – true
- **return_json** (*boolean*) – Whether the return type will be json or not. DEFAULT VALUE – false

Status Codes

- 200 OK – OK

`GET /complexity/project_lines`

Returns project line data for all repositories in an Augur instance, using information from a git platform (GitHub, GitLab, etc.). Each record includes the total and average number of lines in the project repository.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)

Status Codes

- 200 OK – OK

`GET /complexity/project_file_complexity`

Returns project file complexity data for all repositories in an Augur instance, using information from a git platform (GitHub, GitLab, etc.). Each record includes the total and average file complexity of the project repository.

Status Codes

- 200 OK – OK

`GET /complexity/project_blank_lines`

Returns project blank line data for all repositories in an Augur instance, using information from a git platform (GitHub, GitLab, etc.). Each record includes the total and average number of blank lines in the project repository.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)

Status Codes

- 200 OK – OK

`GET /complexity/project_comment_lines`

Returns project comment line data for all repositories in an Augur instance, using information from a git platform (GitHub, GitLab, etc.). Each record includes the total and average number of comment lines in the project repository.

Query Parameters

- **repo_id** (*integer*) – Repository Id (Required)

Status Codes

- 200 OK – OK

`GET /complexity/project_files`

Returns project file data for all repositories in an Augur instance, using information from a git platform (GitHub, GitLab, etc.). Each record includes the total number of files in the project repository.

Status Codes

- 200 OK – OK

`GET /complexity/project_languages`

Returns project language data for all repositories in an Augur instance, using information from a git platform (GitHub, GitLab, etc.). Each record includes the lines and files of a language in a repository.

Status Codes

- 200 OK – OK

`GET /repo-groups/:repo_group_id/repo-messages`

The number of messages exchanged for a repository group over a specified period.

Parameters

- **repo_group_id** (*string*) – Repository Group ID
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`GET /repos/:repo_id/repo-messages`

The number of messages exchanged for a repository over a specified period.

Parameters

- **repo_id** (*string*) – Repository ID.
- **begin_date** (*string*) – Beginning date specification. E.g. values: 2018, 2018-05, 2019-05-01
- **end_date** (*string*) – Ending date specification. E.g. values: 2018, 2018-05, 2019-05-01

Status Codes

- 200 OK – OK

`POST /user/session/generate`

The final step in the Augur Oauth authorization process. The Client Application uses this endpoint to exchange a temporary authorization code (generated in the previous authorization step) with a valid Bearer token.

Query Parameters

- **code** (*string*) – Temporary authorization code (Required)
- **grant_type** (*string*) – Required to be "code" (Required)

Status Codes

- 200 OK – OK
- 400 Bad Request – Missing Argument

Request Headers

- **Authorization** – Client [API_Key] (Required)

Response Headers

- **Cache-Control** – Always set to "no-store"

`POST /user/session/refresh`

Exchange a valid refresh token for a new Bearer token and a new refresh token. The Bearer token returned from this endpoint may be the same as the current Bearer token. If the new Bearer token returned from this endpoint is not the same as the current Bearer token, then the current token is now invalid. The same is true for the existing refresh token.

Query Parameters

- **refresh_token** (*string*) – The refresh token generated in the previous authorization request. (Required)
- **grant_type** (*string*) – Required to be "refresh_token" (Required)

Status Codes

- 200 OK – OK
- 400 Bad Request – Missing Argument

Request Headers

- **Authorization** – Client [API_Key] (Required)

Response Headers

- **Cache-Control** – Always set to "no-store"

`POST /dei/repo/add`

Add and start repo for DEI Badging

Query Parameters

- **id** (*string*) – The source badging ID (Required)
- **level** (*string*) – The badging level (Required)
- **url** (*string*) – The repo URL to track (Required)

Status Codes

- 200 OK – OK
- 400 Bad Request – Missing Argument

Request Headers

- **Authorization** – Client [API_Key] (Required)

`POST /dei/report`

Request the report for the given badging project. On success, the report PDF will be returned in a Binary response. On failure, a JSON response with a status will be returned.

Query Parameters

- **id** (*string*) – The source badging ID (Required)

Status Codes

- 200 OK – OK
- 400 Bad Request – Missing Argument

Request Headers

- **Authorization** – Client [API_Key] (Required)

1.7 Docker

1.7.1 Docker Quick Start

Before you get off to such a quick start, go ahead and

1. Create a fork from augur starting at https://github.com/chaoss/augur
2. Clone that fork locally
3. Checkout the appropriate branch to work on (see notes below):

```
git checkout dev
```

4. Usually, we'll have you checkout the *dev* branch.
5. Make sure to install all the pre-requisites here: https://oss-augur.readthedocs.io/en/main/getting-started/installation.html#dependencies

---

Note: Quick Start:

If you want to start running docker against an external database right away:

1. Follow the instructions to create a database, and database user (if you have just installed Postgresql locally, you may need to follow instructions to allow access to Postgresql from Docker on the next page. tl;dr, there are edits to the Postgresql *pg_hba.conf* and *postgresql.conf* files):

```
CREATE DATABASE augur;
CREATE USER augur WITH ENCRYPTED PASSWORD 'password';
GRANT ALL PRIVILEGES ON DATABASE augur TO augur;
```

2. Install Docker and docker-compose. If you're not familiar with Docker, their starting guide is a great resource.
3. Create a file to store all relevant enviroment variables for running docker. Below is an example file. This file should be named `.env`

```
AUGUR_GITHUB_API_KEY=xxxxxxxxxxxxxxxxxxxxxx
AUGUR_GITHUB_USERNAME=usernameGithub
AUGUR_GITLAB_API_KEY=xxxxxxxxxxxxxxxxxxxxxx
AUGUR_GITLAB_USERNAME=usernameGitlab
AUGUR_DB=yourDBString
```

4. Execute the code from the base directory of the Augur repository:

```
sudo docker build -t augur-docker -f docker/backend/Dockerfile .
sudo docker compose up
```

1.7.2 Getting Started

For the Docker Savvy Who Want to Understand How the Sausage is Made:

Augur provides several Docker images designed to get you started with our software as quickly as possible. They are:

- `augurlabs/augur:backend`, our backend data collection and metrics API
- `augurlabs/augur:frontend`, our metrics visualization frontend (Experimental, will be replaced in the future)

> **Warning:** The frontend is very out of date and will likely not work. It is still available, but it is in the process of being replaced with an entirely new frontend so the old frontend is not being actively fixed.

- `augurlabs/augur:database`, an empty PostgreSQL database with the Augur schema installed

If you're not familiar with Docker, their starting guide is a great resource.

The rest of this section of the documentation assumes you have a working installation of Docker as well as some familiarity with basic Docker concepts and a few basic Docker and Docker,Compose commands.

If you are less familiar with Docker, or experience issues you cannot resolve attempting our "quick start", please follow the instructions in this section, and the next few pages, to set up your environment.

Credentials

Before you get started with Docker, you'll need to set up a PostgreSQL instance either locally or using a remote host. Alternatively, you can also set up the database within a docker container either manually or through docker compose.

Note: Make sure your database is configured to listen to all addresses to work with the containers while running locally. These settings can be edited in your `postgresql.conf`. Additionally, edit the bottom section of your `pg_hba.conf` file with:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host      all             all             0.0.0.0/0               md5
```

If you're interested solely in data collection, we recommend using our test data with the Docker Compose script. This will start up the backend and frontend containers simultaneously, well as an optional database container; however, if you are looking to collect data long term, we **strongly suggest setting up a persistent database instance**; you can find instructions for doing so here. Remember to save off the credentials for your newly minted database; you'll need them shortly.

If you don't care if your data doesn't get persisted or are doing local development, you can use the database containers we provide.

Warning: Using a Docker container as a production database is not recommended. You have been warned!

If you're more interested in doing local development, we recommend using our Docker testing environment image - more on that later.

Configuration File

Besides a database instance, you will also need a GitHub Access Token (`repo` and all `read` scopes except `enterprise`). **This is required for all Docker users**.

First, you'll need to clone the repository. In your terminal, run:

```
$ git clone https://github.com/chaoss/augur.git
$ cd augur/
```

Now that you've got your external database credentials (if you are using one) and your access token, we'll need to set environment variables manually.

Your database credentials and other environment variables used at runtime are stored in a file when running manually and are taken from the active bash session when using docker compose.

You can provide your own `.env` file to pull from. The file should have the below format and set all the variables to some value.

```
AUGUR_GITHUB_API_KEY=xxxxxxxxxxxxxxxxxxxxxx
AUGUR_GITHUB_USERNAME=usernameGithub
AUGUR_GITLAB_API_KEY=xxxxxxxxxxxxxxxxxxxxxx
AUGUR_GITLAB_USERNAME=usernameGitlab
AUGUR_DB=yourDBString
```

Now that you've created your config file or are ready to generate it yourself, you're ready to get going .

1.7.3 Docker

Augur provides a separate Docker image for each layer of our application (database, backend, and frontend). This section details how to build and run these images locally for testing, and also describes how to set up our test environment using Docker.

Note: This page is primarily targeted at developers.

Building the images

All `Dockerfiles` and other Docker-related files are located in `util/docker/<service_name>`, where `<service_name>` is either `backend`, `frontend`, or `database`. To build these images locally, use the following command, being sure to replace `<tag_name>` and `<service_name>` as appropriate.

```
# in the root augur/ directory
$ docker build -t <tag_name> -f util/docker/<service name>/Dockerfile .
```

Note: You can set `<tag_name>` to whatever you like, we recommend something like `local_augur_backend` so you don't get it confused with the official images.

Running containers

To start a container, use the command below. `<container_name>` can be whatever you like, but `<tag_name>` should be the same as in the previous step or the tag of one of the official images.

```
$ docker run -p <host_port>:<docker_port> --name <container_name> --env-file <file_with_
↪enviroment_variables> --add-host host.docker.internal:host-gateway -t <tag_name>
```

Note: If you are running the `backend` service, then `<docker_port>` needs to be `5000`; for `frontend` and `database` the ports are `8080` and `5434`. You can set the `<host_port>` to any **available** port on your machine for any of the services.

Note: If you are running the backend service, you'll also need to add `--env-file docker_env.txt` to your command to make the container aware of your configuration file. You'll also need to add `--add-host host.docker.internal:host-gateway` to your command to make the container able to connect to services running on localhost. Make sure your database is configured to accept the container's connections by making sure that `listen_addresses = '*'` wherever the postgresql.conf is located on your machine and change the pg_hba.conf to accept hosts with a line similar to `host all all 0.0.0.0/0 md5`.

1.7.4 Interacting with the containers

Once the containers are up and running, you have a few options for interacting with them. They will automatically collect data for your repositories - but how do you add repositories? We're glad you asked!

Accessing the containers

If you need to access a running container (perhaps to check the worker logs) or run a CLI command, you can use the following helpful command, replacing `<service_name>` with the appropriate value:

```
$ docker exec -it <service_name> /bin/bash
```

You can also step into a running container at every step of the build process and see the status of the container. (This is typically used for debugging)

First, build the image to output build stages.

Then, run any stage by using the hash that the relevant stage prints out during the build process. The arguments are the same as a normal `docker run`

Viewing container logs

By default, the only logs shown by the container are the logs of Augur's main data collection process. If you started your container(s) in the background, and want to view these logs again, run the following command in the root `augur` directory:

```
# to quickly view the most recent logs
$ docker compose logs

# to watch the logs in real-time (like tail -f)
$ docker compose logs -f
```

As for worker logs. They are currently a work in progress to be made easier to view. Shortly, they will automatically populate on the host machine and it will not be necessary to step inside the container.

Conclusion

This wraps up the Docker section of the Augur documentation. We hope it was useful! Happy hacking!

1.8 Schema

1.8.1 Descriptions of Tables & Purposes

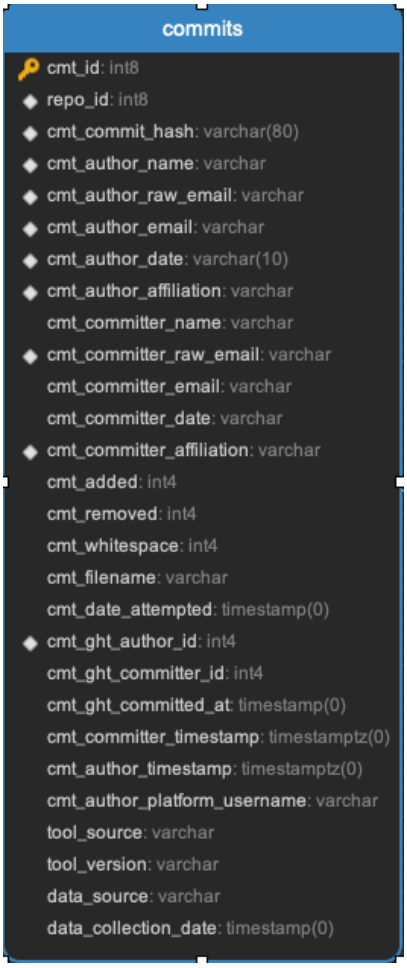
List of Regularly Used Data Tables In Augur

This is a list of data tables in augur that are regularly used and the various tasks attached to them.

Commits

This is where a record for every file in every commit in every repository in an Augur instance is kept.

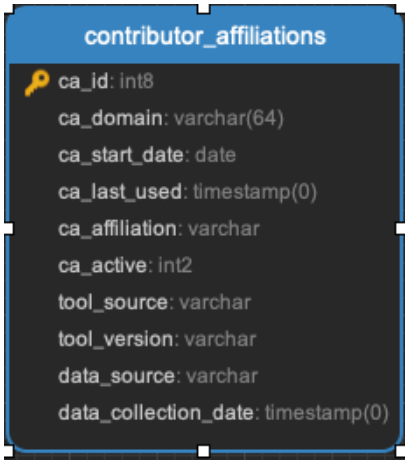
- Task: Facade tasks collect, and also stores platform user information in the commits table.



Contributor_affiliations

A list of emails and domains, with start and end dates for individuals to have an organizational affiliation.

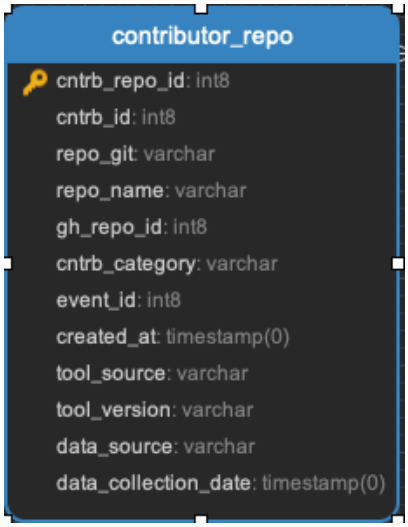
- Populated by default when augur is installed
- Can be edited so that an Augur instance can resolve a larger list of affiliations.
- These mappings are summarized in the **dm_** tables.



Contributor_repo

Storage of a snowball sample of all the repositories anyone in your schema has accessed on GitHub. So, for example, if you wanted to know all the repositories that people on your project contributed to, this would be the table.

- contributor_breadth_model populates this table
- Population of this table happens last, and can take a long time.

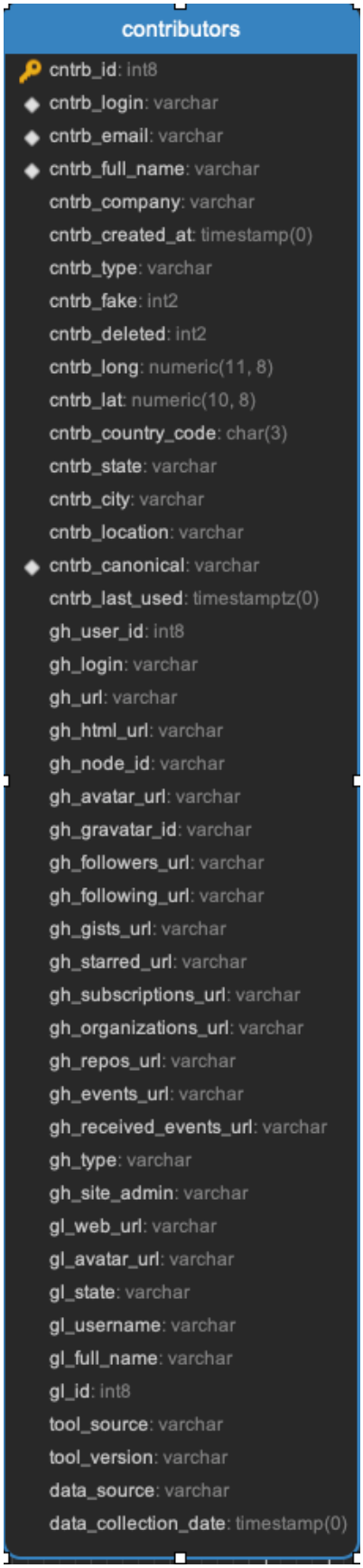


Contributors

These are all the contributors to a project/repo. In Augur, all types of contributions create a contributor record. This includes issue comments, pull request comments, label addition, etc. This is different than how GitHub counts contributors; they only include committers.

- Tasks Adding Contributors:
  - Github Issue Tasks
  - Pull Request Tasks
  - GitLab Issue Tasks

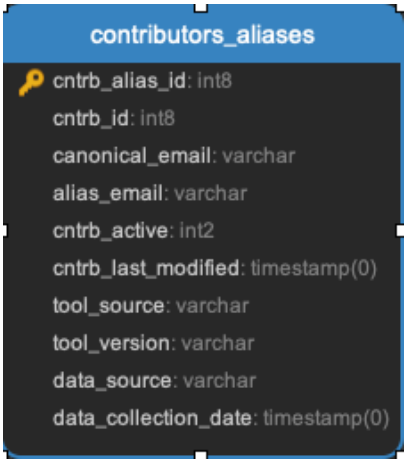
- GitLab Merge Request Tasks
- Facade Tasks


contributors
cntrb_id: int8
cntrb_login: varchar
cntrb_email: varchar
cntrb_full_name: varchar
cntrb_company: varchar
cntrb_created_at: timestamp(0)
cntrb_type: varchar
cntrb_fake: int2
cntrb_deleted: int2
cntrb_long: numeric(11, 8)
cntrb_lat: numeric(10, 8)
cntrb_country_code: char(3)
cntrb_state: varchar
cntrb_city: varchar
cntrb_location: varchar
cntrb_canonical: varchar
cntrb_last_used: timestamptz(0)
gh_user_id: int8
gh_login: varchar
gh_url: varchar
gh_html_url: varchar
gh_node_id: varchar
gh_avatar_url: varchar
gh_gravatar_id: varchar
gh_followers_url: varchar
gh_following_url: varchar
gh_gists_url: varchar
gh_starred_url: varchar
gh_subscriptions_url: varchar
gh_organizations_url: varchar
gh_repos_url: varchar
gh_events_url: varchar
gh_received_events_url: varchar
gh_type: varchar
gh_site_admin: varchar
gl_web_url: varchar
gl_avatar_url: varchar
gl_state: varchar
gl_username: varchar
gl_full_name: varchar
gl_id: int8
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)

Contributors_aliases

These are all the alternate emails that the same contributor might use. These records arise almost entirely from the commit log. For example, if I have two different emails on two different computers that I use when I make a commit, then an alias is created for whatever the 2nd to nth email Augur runs across. If a user's email cannot be resolved, it is placed in the unresolved_commit_emails table. Coverage is greater than 98% since Augur 1.2.4.

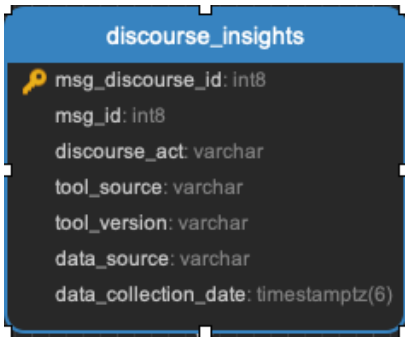
- Tasks:
  - Facade Tasks



Discourse_insights

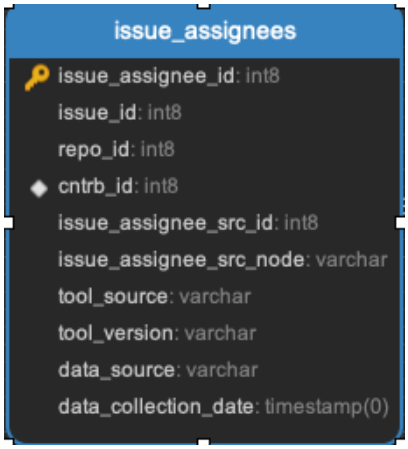
There are nine specific discourse act types identified by the computational linguistic algorithm that underlies the discourse insights task. This task analyzes each comment on each issue or pull request sequentially so that context is applied when determining the discourse act type. These types are:

- negative-reaction
- answer
- elaboration
- agreement
- question
- humor
- disagreement
- announcement
- appreciation
- Tasks:
  - Discourse Insights Task



issue_assignees || issue_events || issue_labels

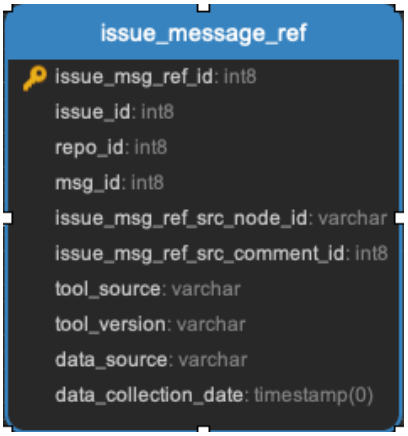
- Task:
  - Github or Gitlab Issues Task



issue_message_ref

A link between the issue and each message stored in the message table.

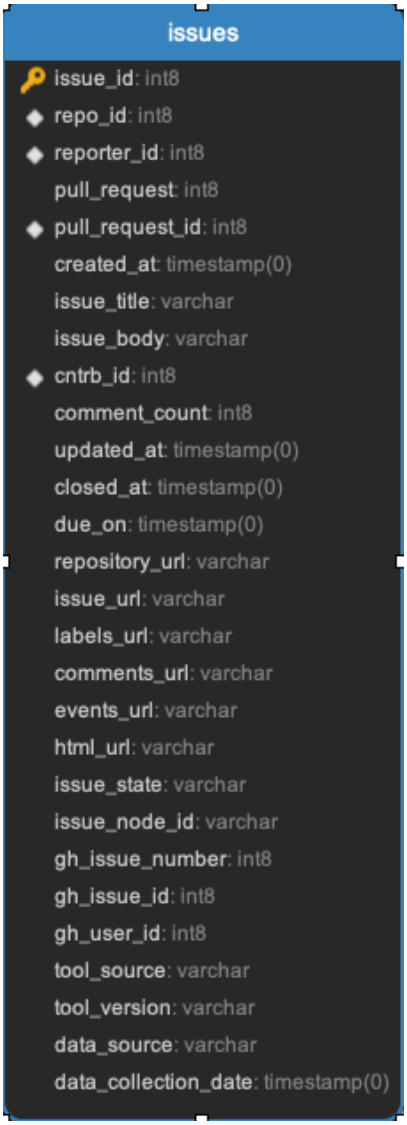
- Task:
  - Github or Gitlab Issues Task



issues

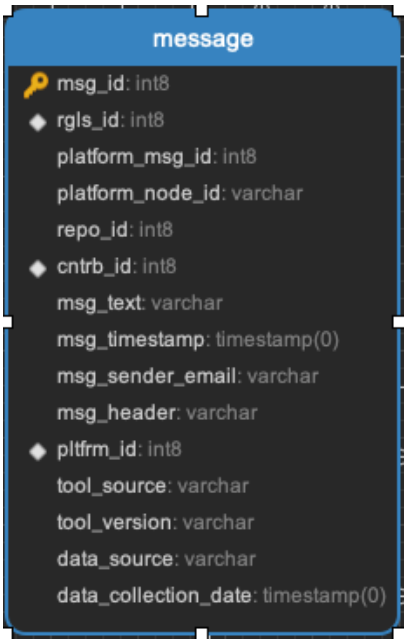
Is all the data related to a GitHub Issue.

- Task:
  - Github or Gitlab Issues Task



Message

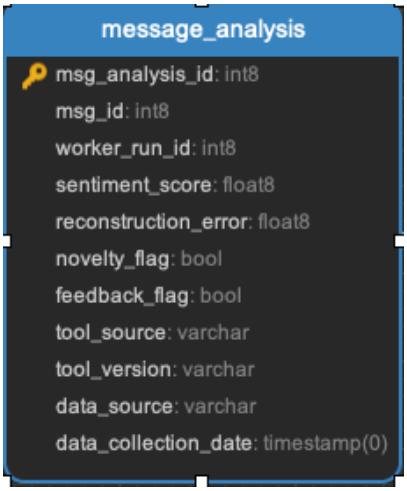
Every pull request or issue related message. These are then mapped back to either pull requests, or issues, using the __msg_ref tables



Message_analysis

Two factors evaluated for every pull request on issues message: What is the sentiment of the message (positive or negative), and what is the novelty of the message in the context of other messages in that repository.

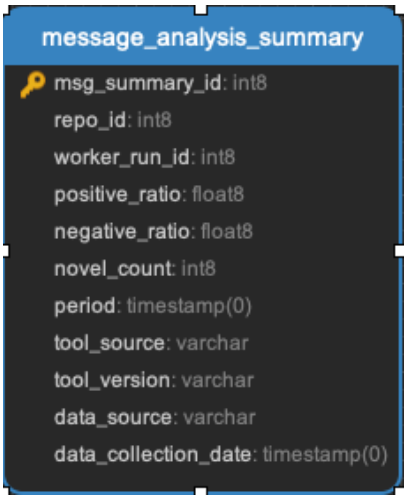
- Task:
  - Message Insights Task



Message_analysis_summary

A summary level representation of the granular data in message_analysis.

- Task:
  - Message Insights Task



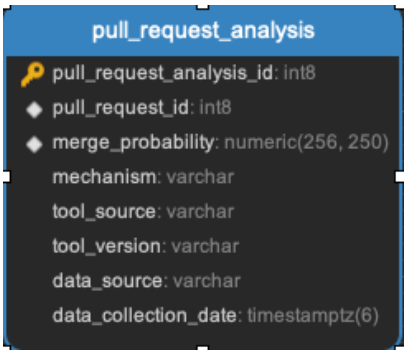
Platform

Reference data with two rows: one for GitHub, one for GitLab.

Pull_request_analysis

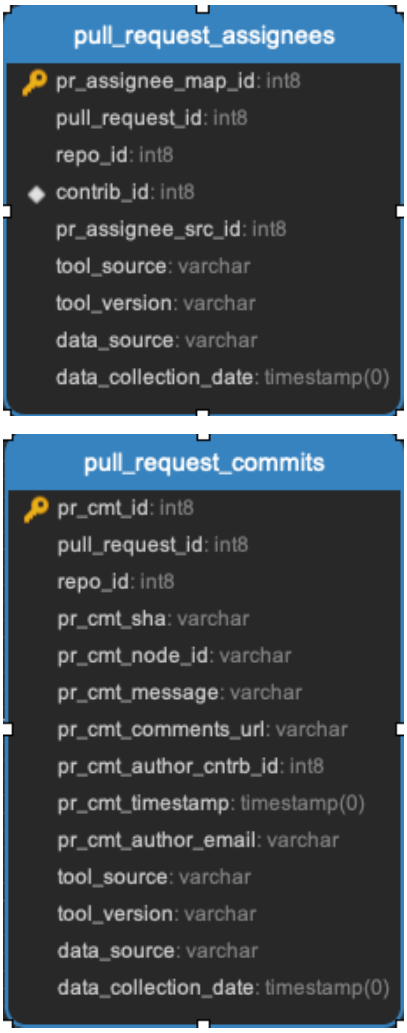
A representation of the probability of a pull request being merged into a repository, based on analysis of the properties of previously merged pull requests in a repository. (Machine learning tasks)

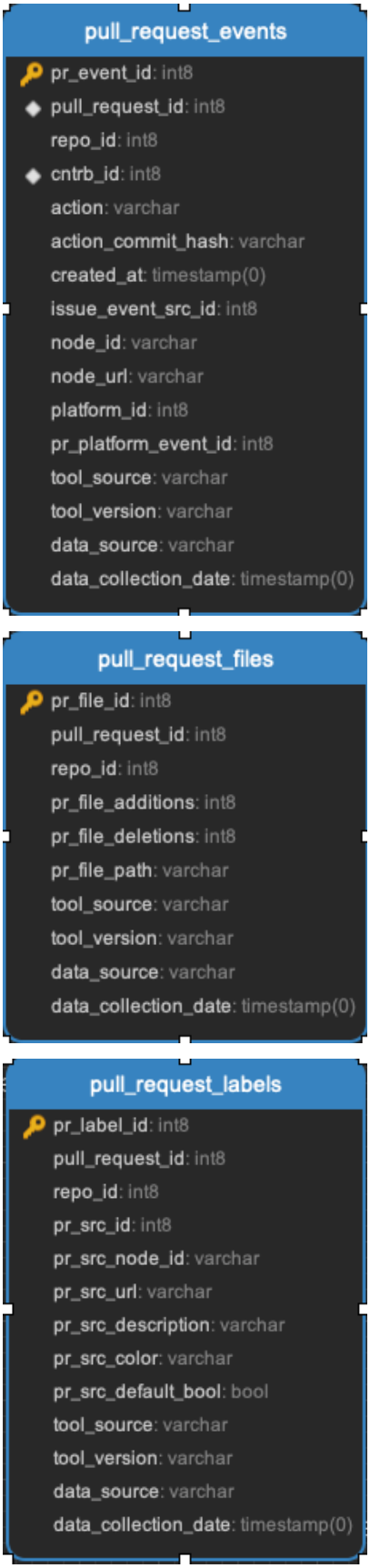
- Task:
  - Pull request analysis task



pull_request_assignees || pull_request_commits || pull_request_events || pull_request_files || pull_request_labels || pull_request_message_ref

All the data related to pull requests. Every pull request will be in the pull_requests data.



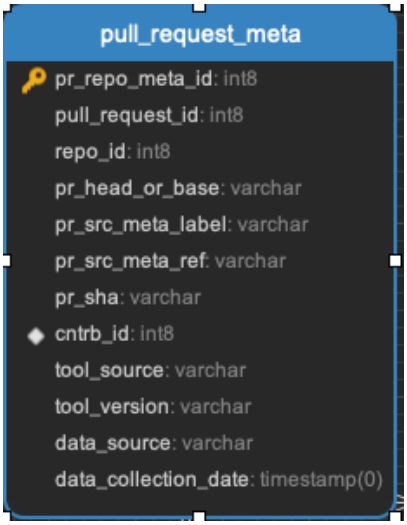

pull_request_events
pr_event_id: int8
pull_request_id: int8
repo_id: int8
cntrb_id: int8
action: varchar
action_commit_hash: varchar
created_at: timestamp(0)
issue_event_src_id: int8
node_id: varchar
node_url: varchar
platform_id: int8
pr_platform_event_id: int8
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)
pull_request_files
pr_file_id: int8
pull_request_id: int8
repo_id: int8
pr_file_additions: int8
pr_file_deletions: int8
pr_file_path: varchar
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)
pull_request_labels
pr_label_id: int8
pull_request_id: int8
repo_id: int8
pr_src_id: int8
pr_src_node_id: varchar
pr_src_url: varchar
pr_src_description: varchar
pr_src_color: varchar
pr_src_default_bool: bool
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)

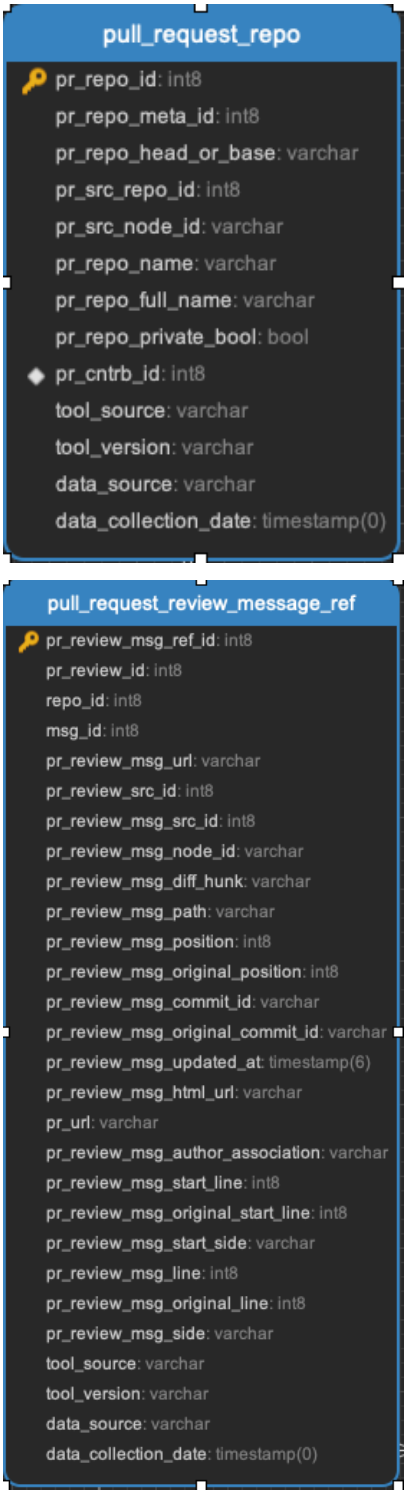
schema/images/pull_request_ref.png

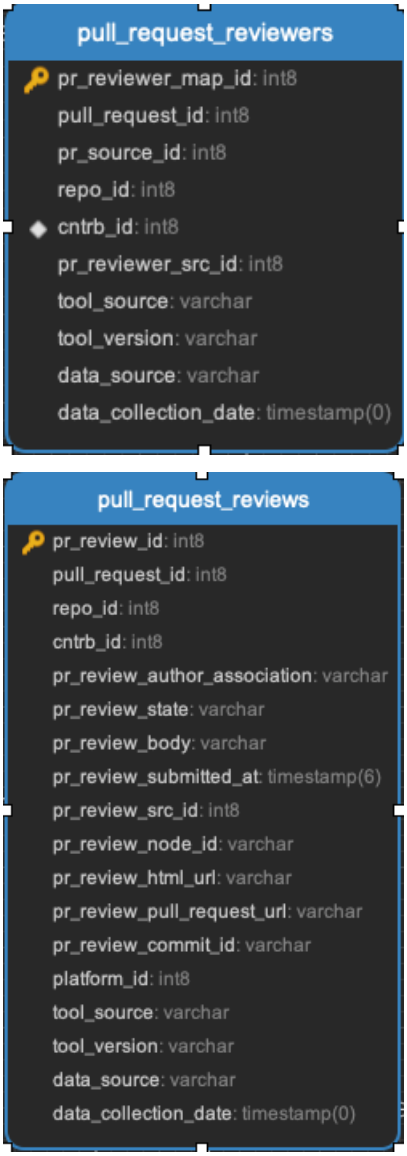
pull_request_meta || pull_request_repo || pull_request_review_message_ref || pull_request_reviewers || pull_request_reviews || pull_request_teams || pull_requests

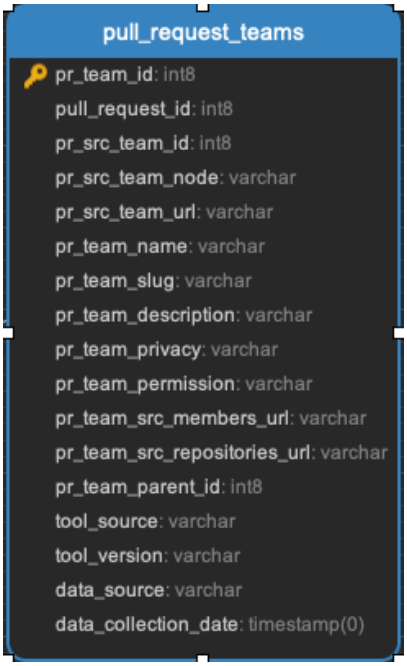
All the data related to pull requests. Every pull request will be in the pull_requests data.

schema/images/pull_request.png




pull_request_repo
pr_repo_id: int8
pr_repo_meta_id: int8
pr_repo_head_or_base: varchar
pr_src_repo_id: int8
pr_src_node_id: varchar
pr_repo_name: varchar
pr_repo_full_name: varchar
pr_repo_private_bool: bool
pr_cntrb_id: int8
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)
pull_request_review_message_ref
pr_review_msg_ref_id: int8
pr_review_id: int8
repo_id: int8
msg_id: int8
pr_review_msg_url: varchar
pr_review_src_id: int8
pr_review_msg_src_id: int8
pr_review_msg_node_id: varchar
pr_review_msg_diff_hunk: varchar
pr_review_msg_path: varchar
pr_review_msg_position: int8
pr_review_msg_original_position: int8
pr_review_msg_commit_id: varchar
pr_review_msg_original_commit_id: varchar
pr_review_msg_updated_at: timestamp(6)
pr_review_msg_html_url: varchar
pr_url: varchar
pr_review_msg_author_association: varchar
pr_review_msg_start_line: int8
pr_review_msg_original_start_line: int8
pr_review_msg_start_side: varchar
pr_review_msg_line: int8
pr_review_msg_original_line: int8
pr_review_msg_side: varchar
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)

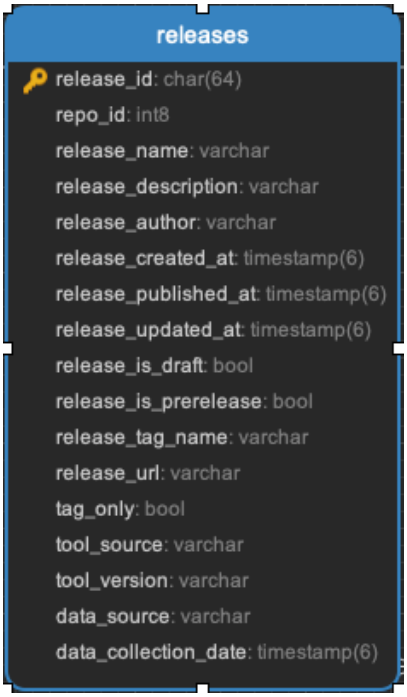

pull_request_reviewers
pr_reviewer_map_id: int8
pull_request_id: int8
pr_source_id: int8
repo_id: int8
cntrb_id: int8
pr_reviewer_src_id: int8
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)
pull_request_reviews
pr_review_id: int8
pull_request_id: int8
repo_id: int8
cntrb_id: int8
pr_review_author_association: varchar
pr_review_state: varchar
pr_review_body: varchar
pr_review_submitted_at: timestamp(6)
pr_review_src_id: int8
pr_review_node_id: varchar
pr_review_html_url: varchar
pr_review_pull_request_url: varchar
pr_review_commit_id: varchar
platform_id: int8
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)



Releases

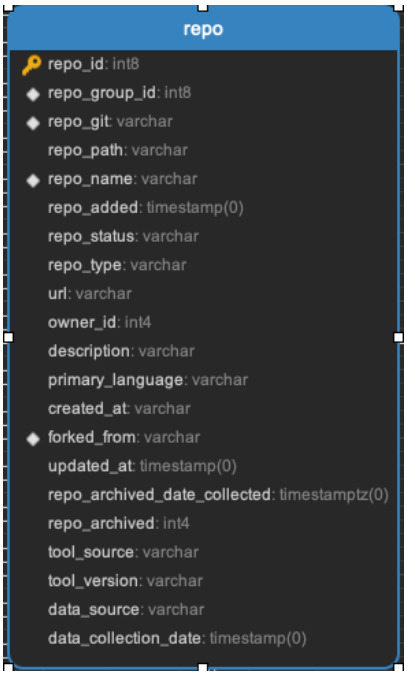
Github declared software releases or release tags. For example: https://github.com/chaoss/augur/releases

- Task:
  - Release Task.



Repo

A list of all the repositories.



Repo_badging

A list of CNCF badging information for a project. Reads this api endpoint: https://bestpractices.coreinfrastructure.org/projects.json

Repo_cluster_messages

Identifying which messages and repositories are clustered together. Identifies project similarity based on communication patterns.

- Task:
  - Clustering task

Repo_dependencies

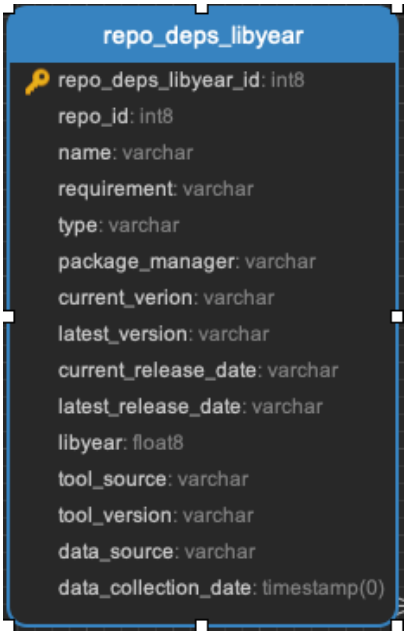
Enumerates every dependency, including dependencies that are not package managed.

- Task:
  - process_dependency_metrics

Repo_deps_libyear

(enumerates every package managed dependency) Looks up the latest release of any library that is imported into a project. Then it compares that release date, the release version of the library version in your project (and its release date), and calculates how old your version is, compared to the latest version. The resulting statistic is "libyear". This task runs with the facade tasks, so over time, you will see if your libraries are being kept up to date, or not.

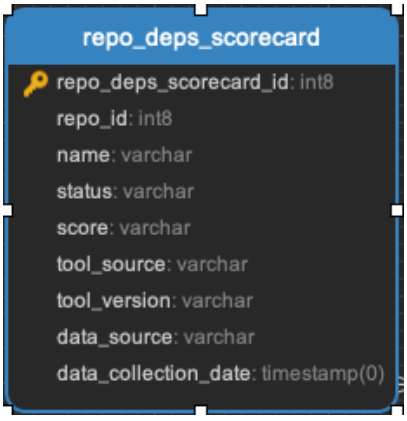
- **Scenarios:**
  - If a library is updated, but you didn't change your version, the libyear statistic gets larger
  - If you updated a library and it didn't get older, the libyear statistic gets smaller.
- Task:
  - process_libyear_dependency_metrics



Repo_deps_scorecard

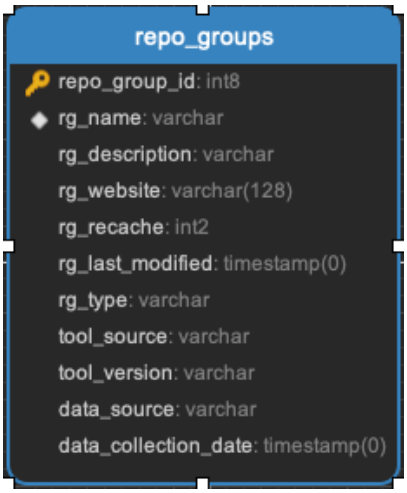
Runs the OSSF Scorecard over every repository (https://github.com/ossf/scorecard) : There are 16 factors that are explained at that repository location.

- Task:
  - process_ossf_scorecard_metrics



Repo_groups

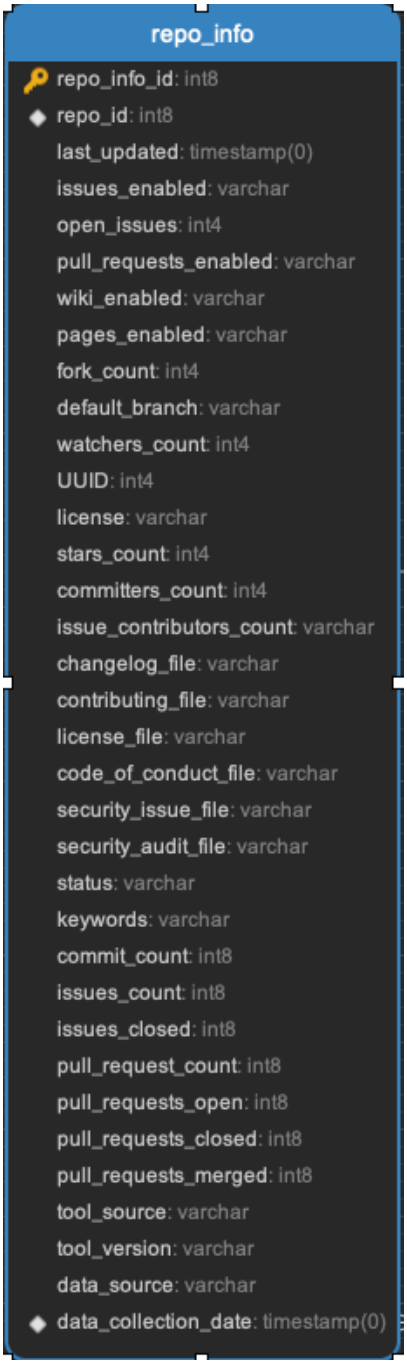
Reference data. The repo groups in an augur instance.



Repo_info

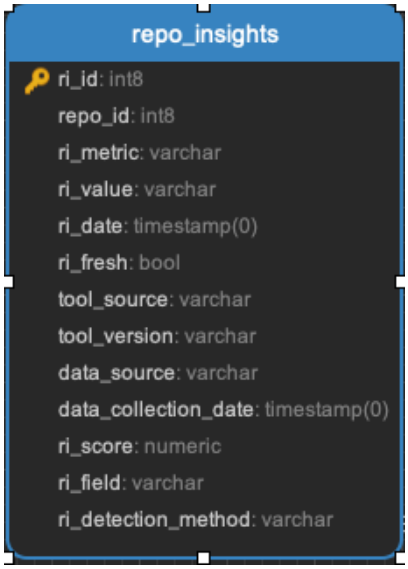
This task gathers metadata from the platform API that includes things like "number of stars", "number of forks", etc. AND it also gives us : Number of issues, number of pull requests, etc. .. THAT information we use to determine if we have collected all of the PRs and Issues associated with a repository.

- Task:
  - repo info task


repo_info
repo_info_id: int8
repo_id: int8
last_updated: timestamp(0)
issues_enabled: varchar
open_issues: int4
pull_requests_enabled: varchar
wiki_enabled: varchar
pages_enabled: varchar
fork_count: int4
default_branch: varchar
watchers_count: int4
UUID: int4
license: varchar
stars_count: int4
committers_count: int4
issue_contributors_count: varchar
changelog_file: varchar
contributing_file: varchar
license_file: varchar
code_of_conduct_file: varchar
security_issue_file: varchar
security_audit_file: varchar
status: varchar
keywords: varchar
commit_count: int8
issues_count: int8
issues_closed: int8
pull_request_count: int8
pull_requests_open: int8
pull_requests_closed: int8
pull_requests_merged: int8
tool_source: varchar
tool_version: varchar
data_source: varchar
data_collection_date: timestamp(0)

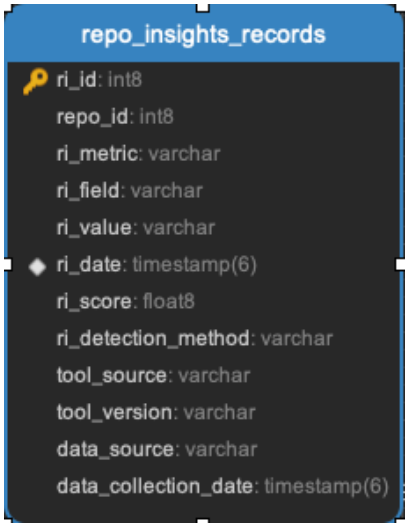
Repo_insights

- Task:
  - Insight task



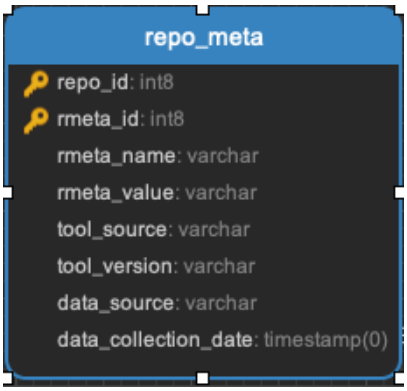
Repo_insights_records

- Task:
  - Insight task



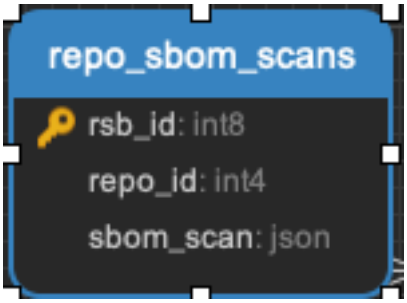
Repo_meta

Exists to capture repo data that may be useful in the future. Not currently populated.



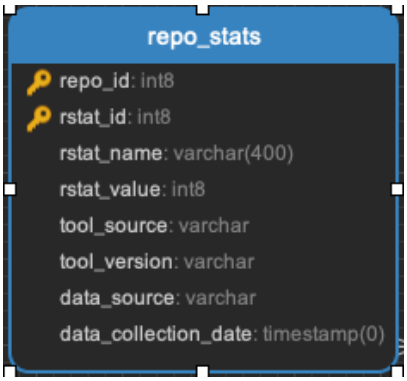
Repo_sbom_scans

This table links the augur_data schema to the augur_spdx schema to keep a list of repositories that need licenses scanned. (These are for file level license declarations, which are common in Linux Foundation projects, but otherwise not in wide use).



Repo_stats

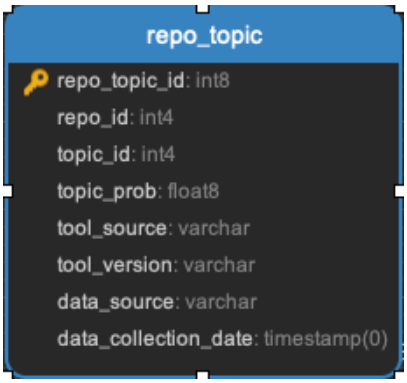
Exists to capture repo data that may be useful in the future. Not currently populated.



Repo_topic

Identifies probable topics of conversation in discussion threads around issues and pull requests.

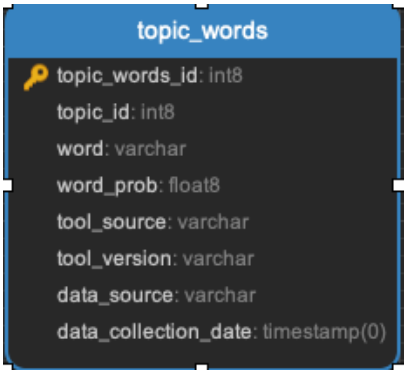
- Task:
  - Clustering task



Topic_words

Unigrams, bigrams, and trigrams associated with topics in the repo_topic table.

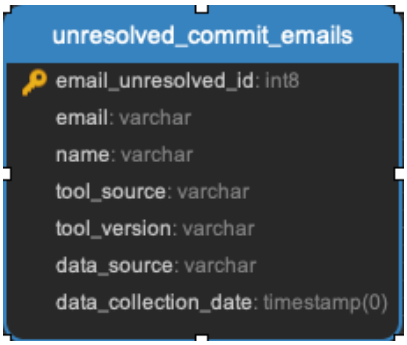
- Task:
  - Clustering task



Unresolved_commit_emails

Emails from commits that were not initially able to be resolved using automated mechanisms.

- Task:
  - Facade Tasks.

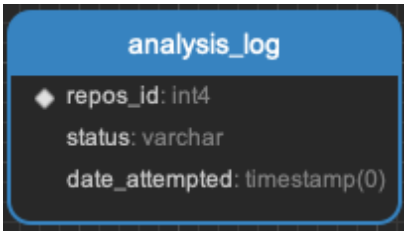


List of Working Data Tables In Augur

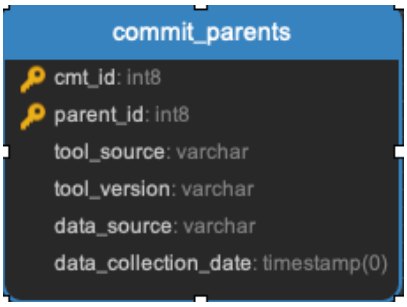
This Is A List of Working Tables In Augur and The Tasks Attached to Them.

They are in lowercase to represent exactly how they look like on the actual table.

- analysis_log - this table is a record of the analysis steps the facade tasks have taken on an augur instance. A listing of all the analysis steps taken for every repository is recorded as they are completed.
  - Tasks Associated With It?
    - Facade Tasks



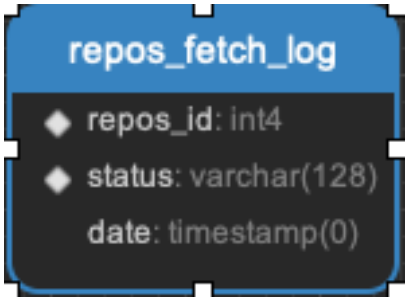
- commit_parents - this table keeps a record of parent commits that are squashed during Facade collection.



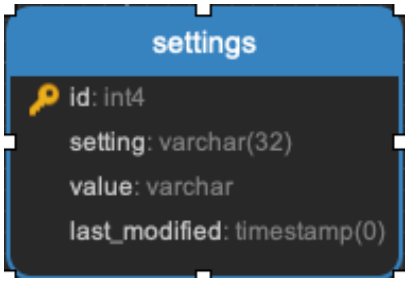
Other working tables are:

- **exclude**

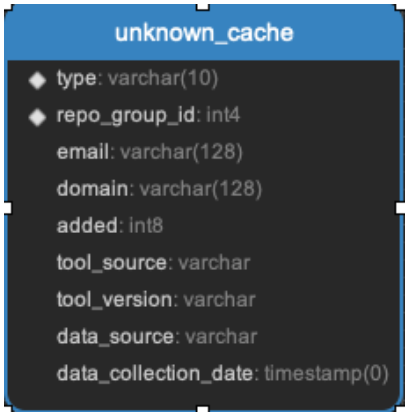
- repos_fetch_log



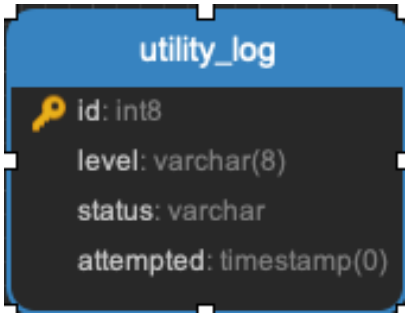
- settings



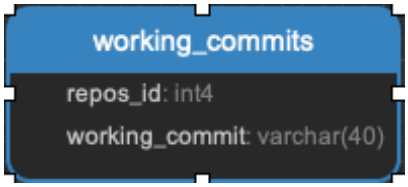
- unknown_cache



- utility_log



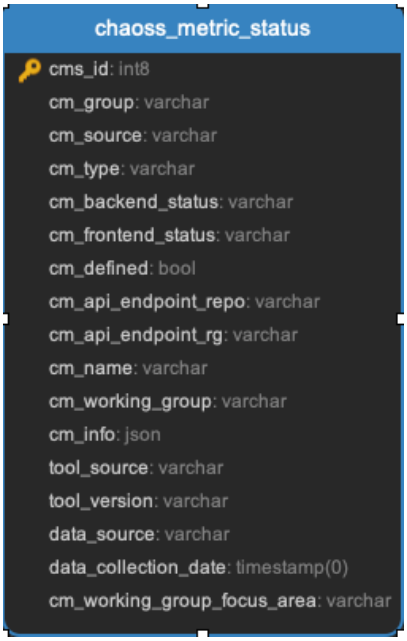
- working_commits



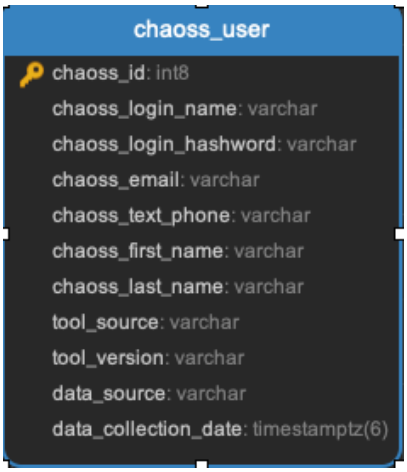
List of Unused Data Tables In Augur

** This is a list of data tables in augur that are not currently in use. **

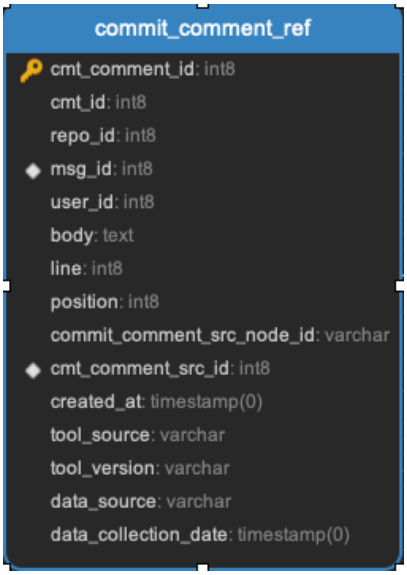
- chaoss_metric_status



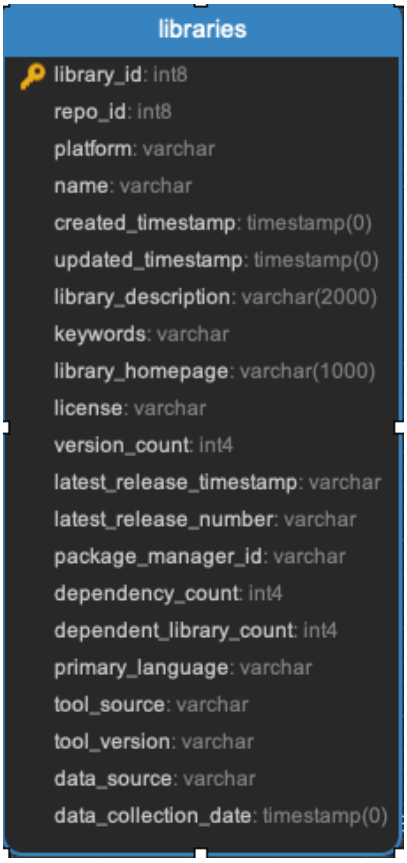
- chaoss_user



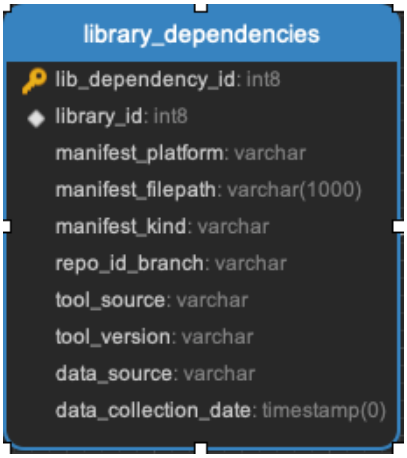
- commit_comment_ref



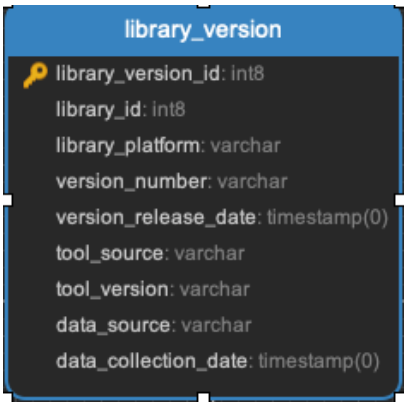
- libraries



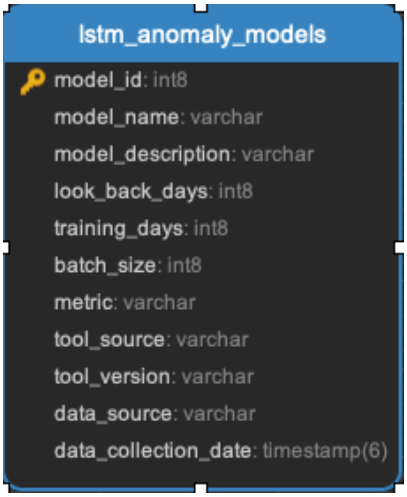
- library_dependencies



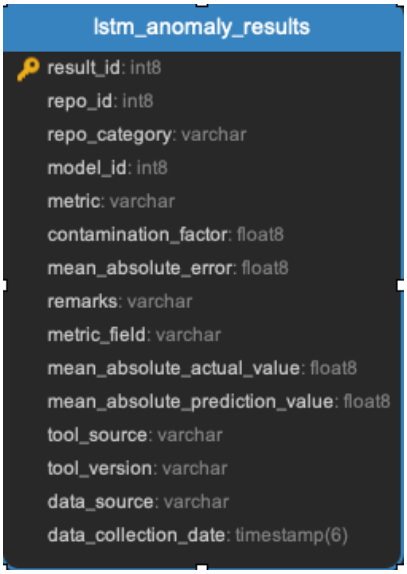
- library_version



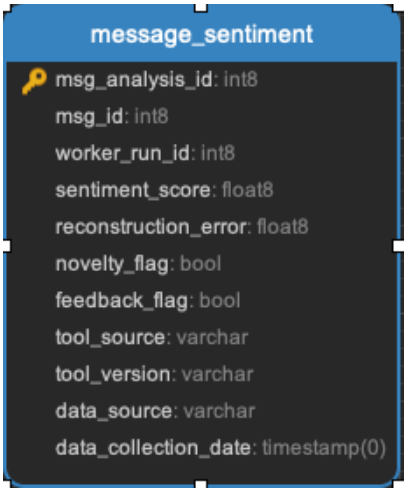
- lstm_anomaly_models



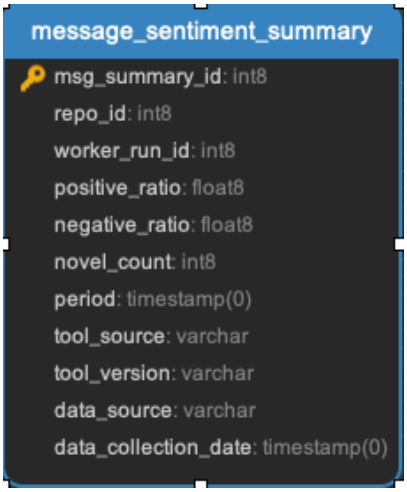
- lstm_anomaly_results



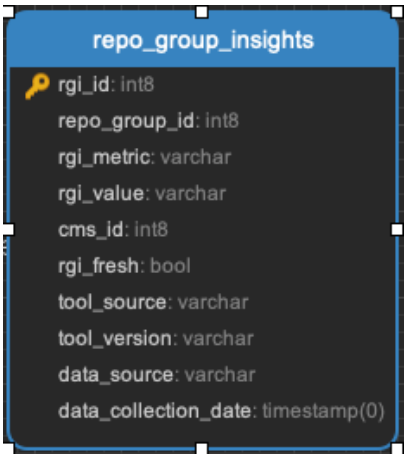
- message_sentiment



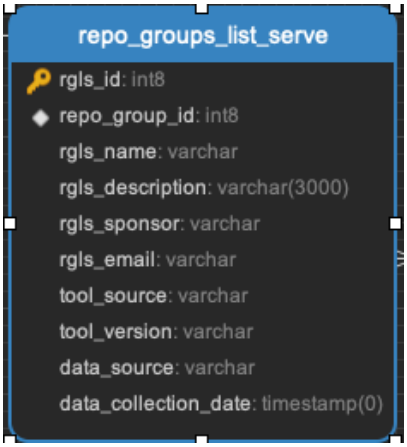
- message_sentiment_summary



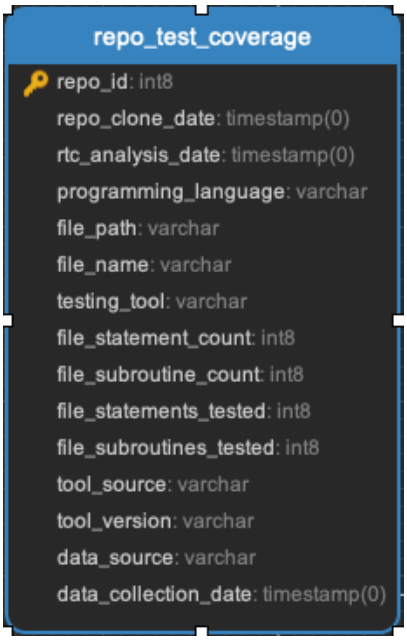
- Repo_group_insights



- repo_groups_list_serve



- repo_test_coverage

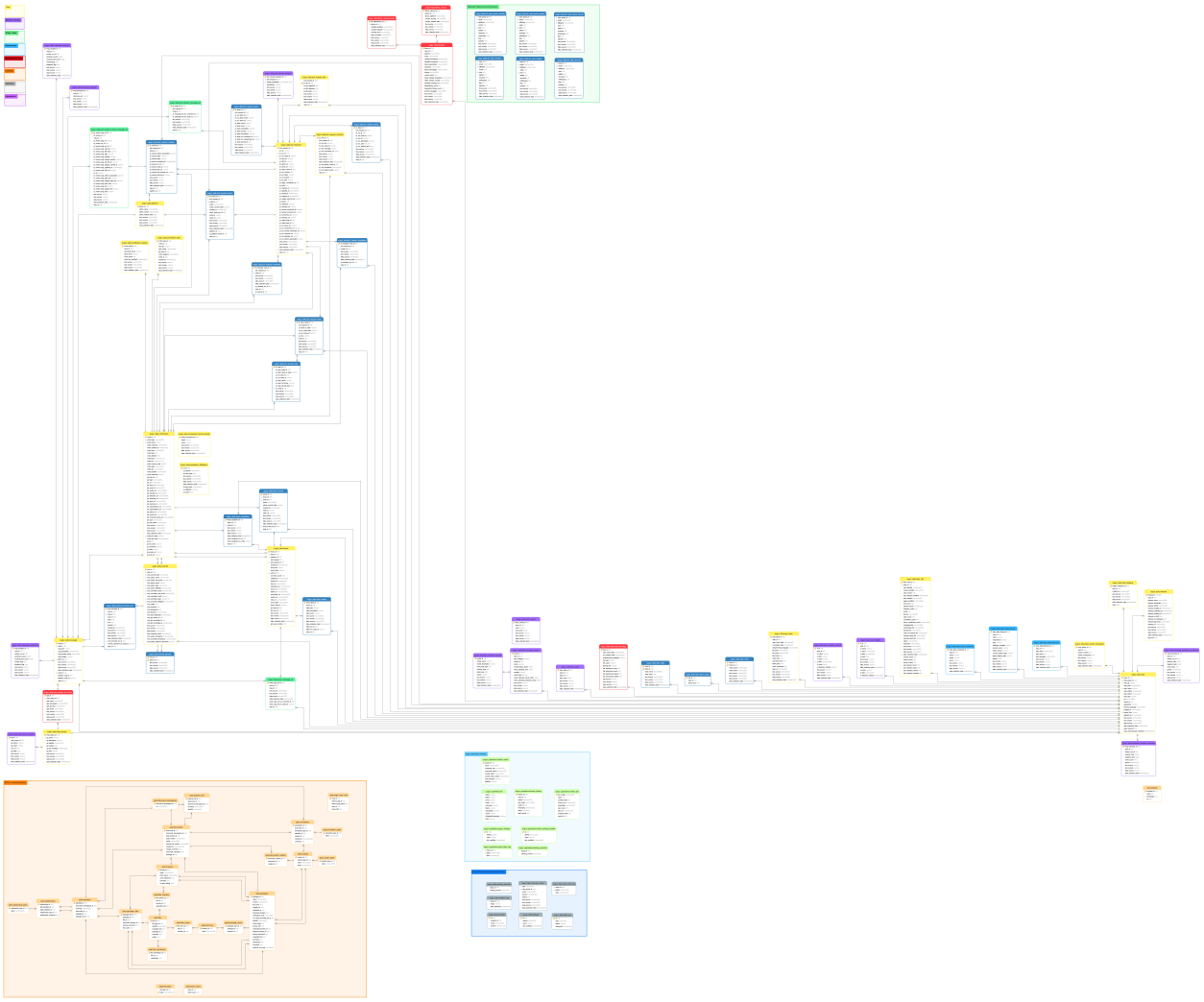


Complete Data Model on a Page

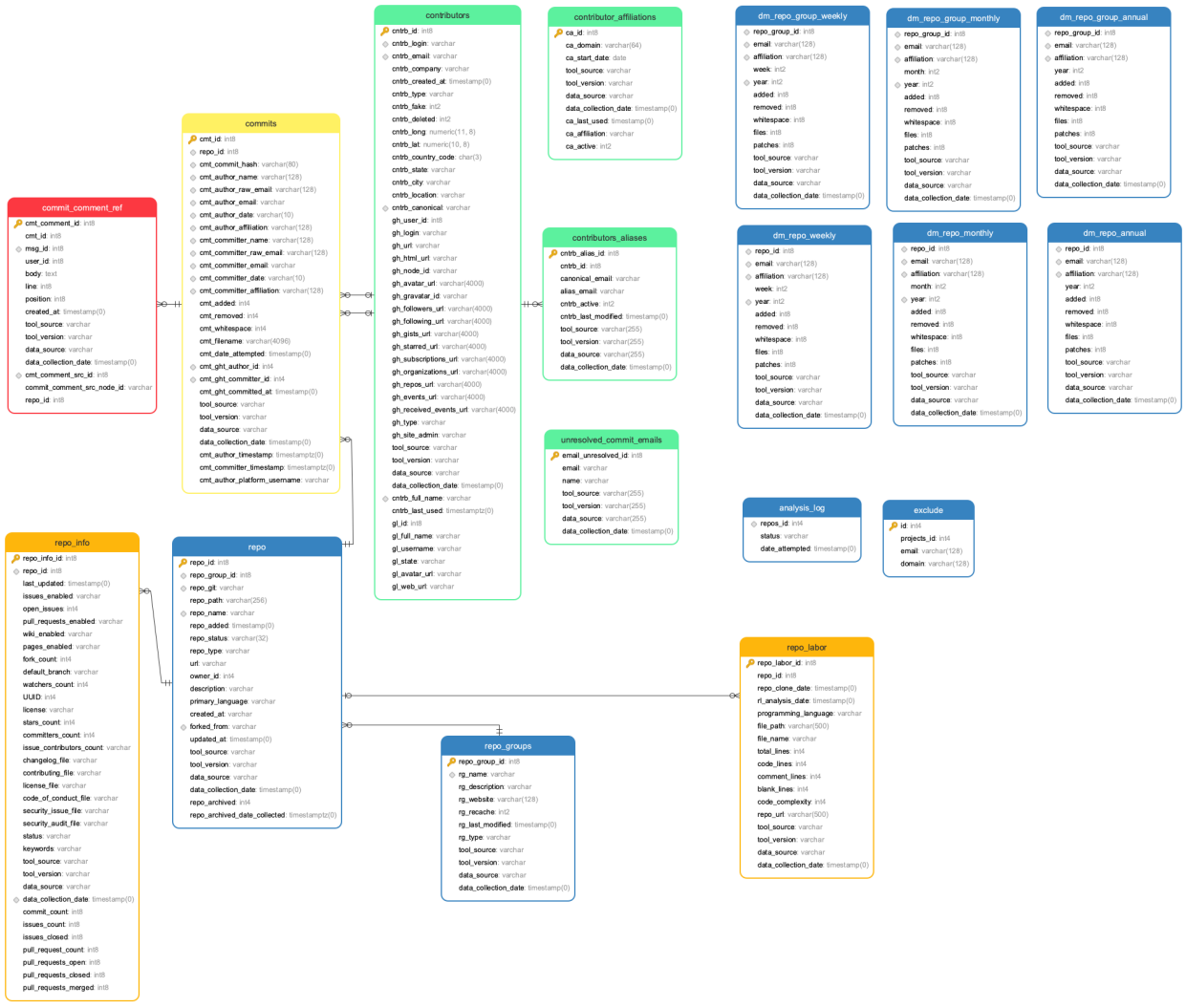
The latest version of Augur includes a *schema* that brings together data around key artifacts of open source software development.

This document details how to create the schema as well as some information on its contents and design.

Complete Data Model, With Key Table Types Highlighted



Complete Data Model, For Current Release



Creating the schema

The process for creating the schema is detailed in the database section of the Getting Started guide.

Schema Overview

Augur Data

The `augur_data` schema contains *most* of the information analyzed and constructed by Augur. The origin's of the data inside of augur are from data collection tasks and populate this schema.:

1. `augur.tasks.github.*`: Tasks that pull data from the GitHub API. Primarily, pull requests and issues are collected before more complicated data. Note that all messages are stored in Augur in the `messages` table. This is to facilitate easy analysis of the tone and characteristics of text communication in a project from one place.

2. `augur.tasks.git.facade_tasks`: Based on http://www.github.com/brianwarner/facade, but substantially modified in the fork located at http://github.com/sgoggins/facade. The modifications include modularization of code, connections to Postgresql data instead of MySQL and other changes noted in the commit logs. Further modifications have been made to work with augur as well as seemlessly integrate it into data collection.

3. `augur.tasks.data_analysis.insight_worker.tasks`: Generates summarizations from raw data gathered from commits, issues, and other info.

4. `augur.tasks.github.pull_requests.tasks`: Collects Pull Request related data such as commits, contributors,assignees, etc. from the Github API and stores it in the Augur database.

Augur Operations

The `augur_operations` tables are where most of the operations tables exist. There are a few, like `settings` that remain in `augur_data` for now, but will be moved. They keep records related to analytical history and data provenance for data in the schema. They also store information including API keys.

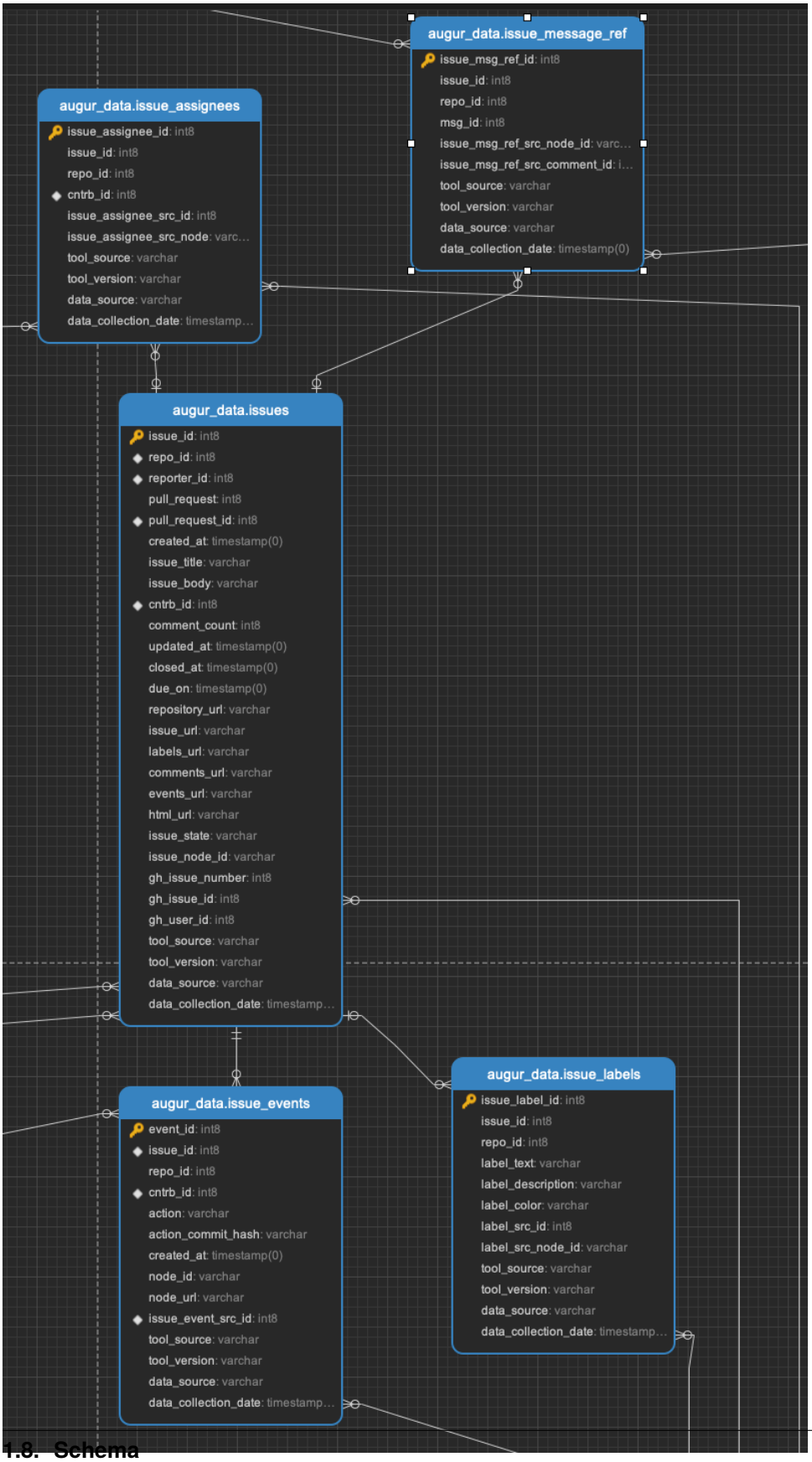
Some key tables in this schema include:

- `config`, which contains the config options for the application. Key options include the facade repo_directory as well as primary api key.
- `collection_status`, contains the status of each aspect of data collection for each repo added to Augur. For example, it shows the status of the facade jobs for every repository.

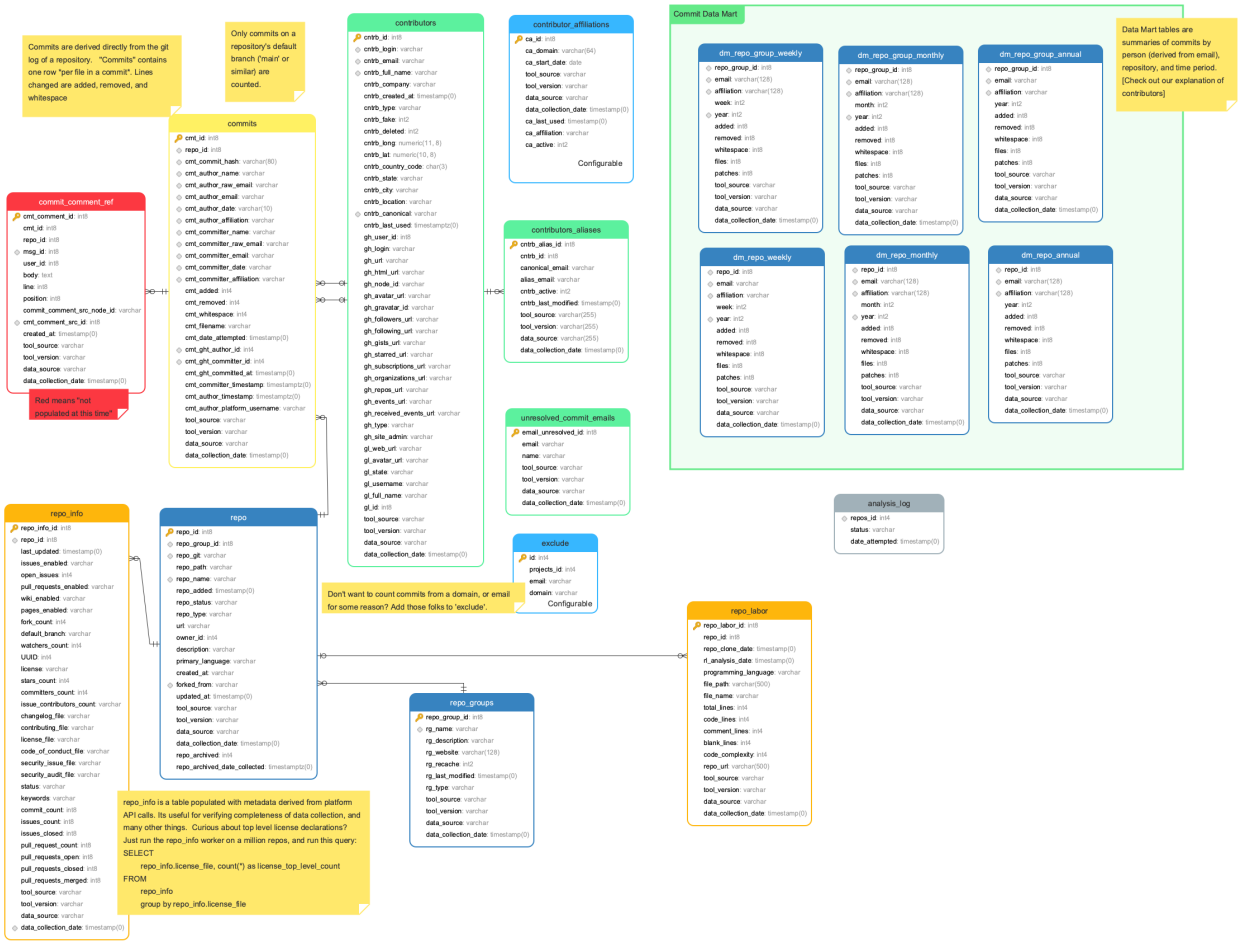
SPDX

The `spdx` schema serves the storage for software bill of materials and license declarations scans on projects, conducted using this fork of the DoSOCSv2 project: https://github.com/Nebrethar/DoSOCSv2

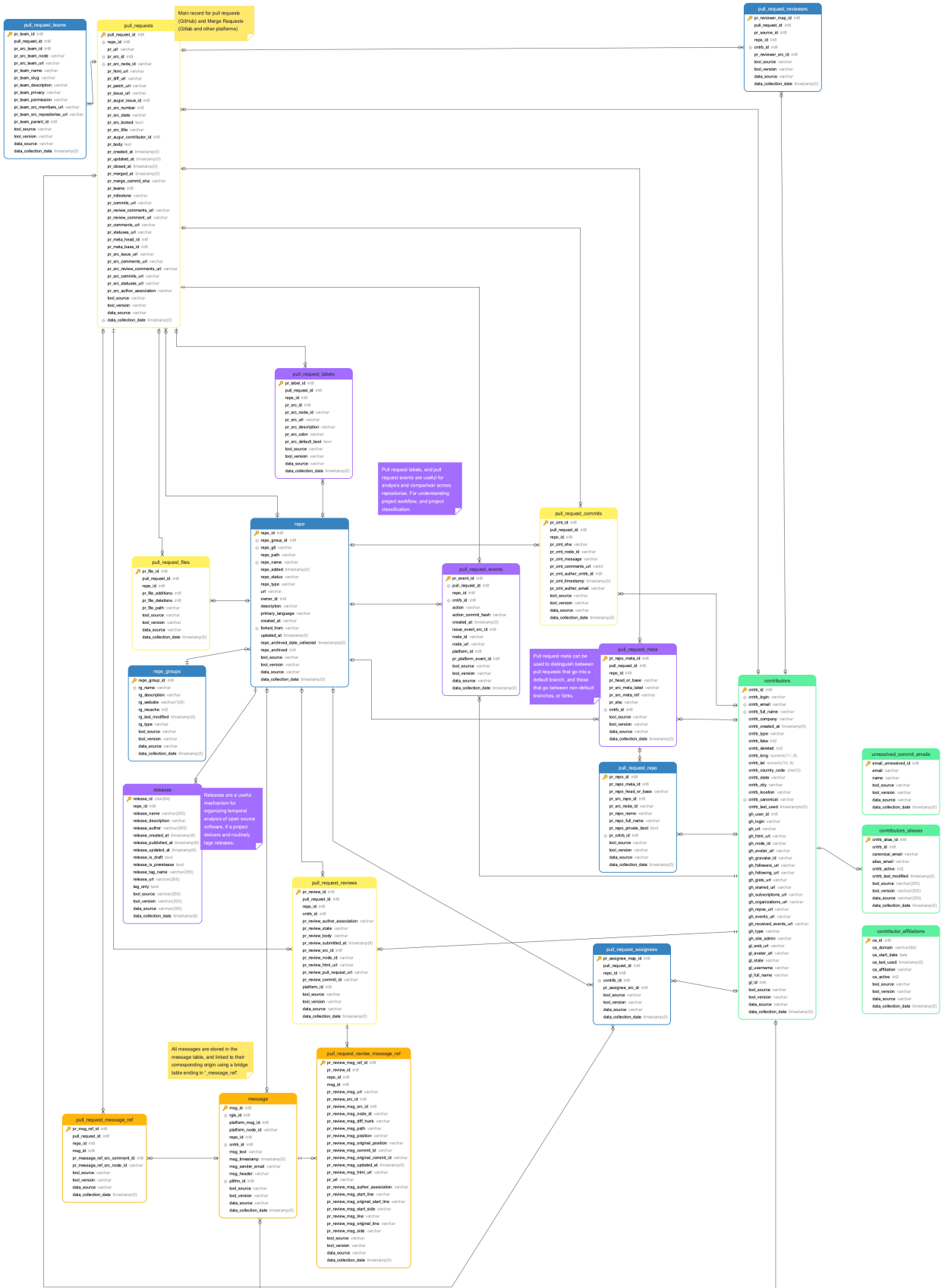
1.8.2 Focus: Issues



1.8.3 Focus: Commits

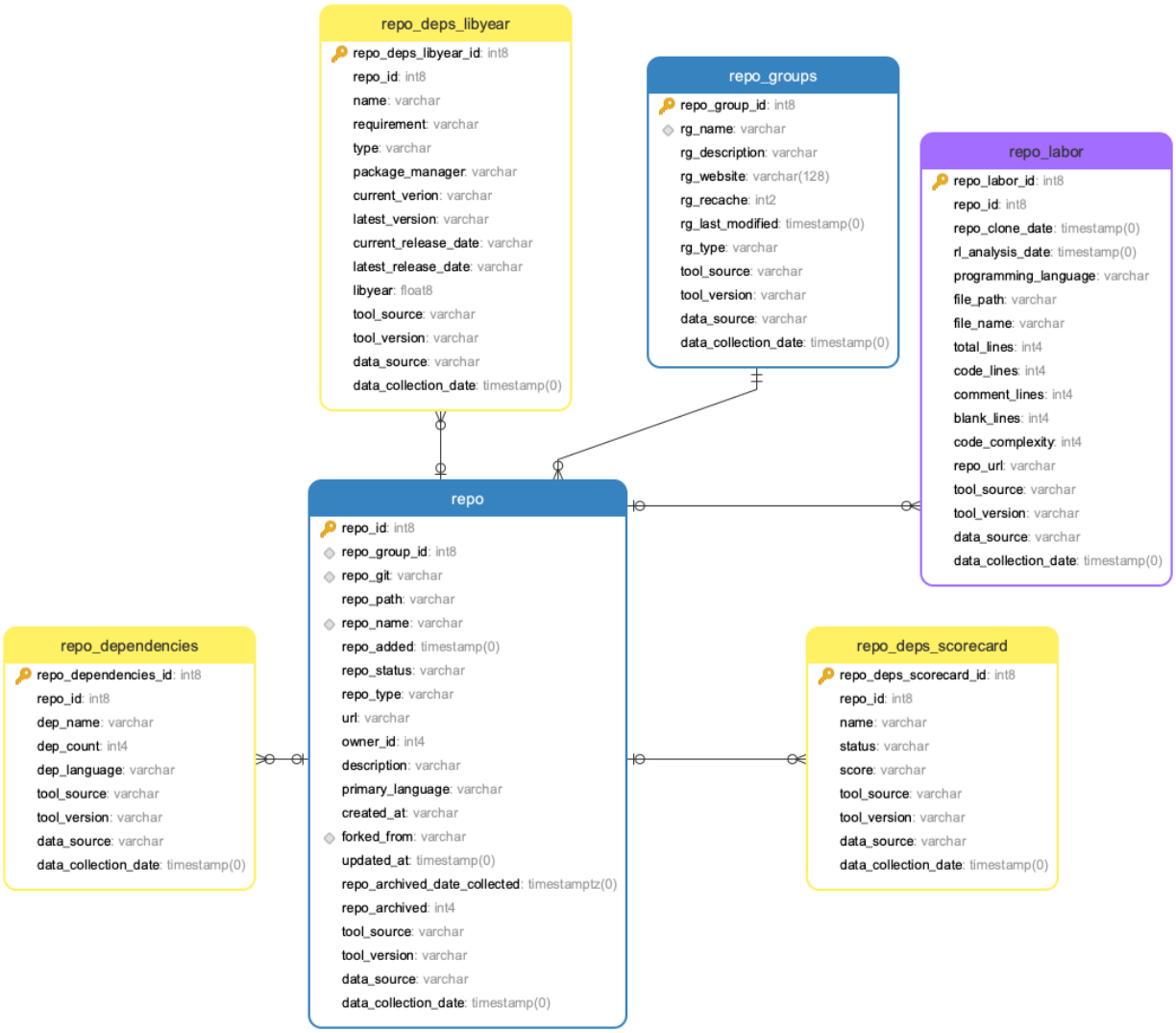


1.8.4 Focus: Pull Requests



1.8.5 Focus: Contributors

1.8.6 Focus: Dependencies



1.9 Augur OAuth Flow

Augur implements the Oauth 2.0 specification, and each Augur instance is capable of acting as an authorization server for external applications.

1.9.1 Prerequisites

If your Augur instance is running behind Nginx or Apache, make sure this parameter (or its Apache equivalent) is set in your `sites-enabled` configuration:

```
proxy_set_header X-Forwarded-Proto $scheme;
```

Registering a user account on the desired Augur instance is a requirement for creating a Client Application. The developer of the application must follow the below steps:

1. Navigate to the home page of the desired Augur instance.
2. Click "Login" on the navigation bar.
3. Click "Register" and fill out the account details.

Once you have registered an account, follow the below steps to create a new Client Application:

1. Click your username in the navigation bar.
2. Click "Profile".
3. Click "Applications"
4. **In the create application form, fill out the application name and redirect URL**
    - The redirect URL is relative to the user-agent (i.e. the user's browser), and **must** be accessible to the user-agent.
    - If you are testing an application locally, you may use `http://127.0.0.1/` or `http://localho.st` as the host for the redirect URL. The authorization server will *not* prevent redirection if the redirect url is unreachable.

Once the application has been created, note the Application ID and Client Secret, as you will need them for application authentication requests.

1.9.2 Authorization Flow

The auth flow **must** be initiated by a user intent. Your application **must not** request initial authorization on the user's behalf, and **must not** automatically redirect the user to the authorization server.

Initial Request

The authorization flow is initiated when a user clicks a link or button which redirects the user-agent (browser) to the authorization server. This request URL must be of the following format:

```
https://augur.example.com/user/authorize?
    client_id=[your application ID]
    &response_type="code"
    &state=[optional value that you define]
```

Authorization Response

Upon redirection, the user is presented with a verification page on the authorization server which indicates the requesting application, what information will be shared, and where the user will be redirected after authorizing. Should the user choose to approve authorization, the authorization server will redirect the user-agent to the redirect URL provided during creationg of the Client Application. The redirect request from the authorization server will be of the following format:

```
https://example.com/your/redirect/url?
    code=[temporary authorization code]
    &state=[the same state (if) provided in the previous request]
```

The temporary authorization code provided in this response is only valid for a number of seconds, and should be considered volatile (it is one-time use, and only exists for the duration required to complete the transaction).

The Client Application **must not** store the temporary authorization code for any longer than is necessary to exchange the code for a User Session Token. The following request must be made immediately upon receipt of the temporary authorization code.

Temporary Code Exchange

The temporary authorization code provided from the initial authorization response must be exchanged for a User Session Token before user authentication through the Client Application can take place.

The Client Application must make the following request in order to facilitate this exchange:

```
URL: https://augur.example.com/api/unstable/user/session/generate
arguments:
    code: [the temporary authorization code]
    grant_type: "code"
headers:
    Authorization: Client [your client secret]
```

The authorization server will respond with the following on success:

```
{
    "status": "Validated",
    "username": "the username associated with this request",
    "access_token": "the new Bearer token",
    "refresh_token": "the new refresh token",
    "token_type": "Bearer",
    "expires": [integer: seconds until this access_token expires]
}
```

Success!

Now that the temporary code exchange is complete, your application has the authorization required to make requests on behalf of the logged-in user.

Refreshing Sessions

When a User Session Token expires, the Client Application has two options for reauthorization. The application may ask the user to manually reauthenticate by presenting a link or button which restarts the authentication flow.

The application may also attempt automatic reauthorization using the previously provided refresh token. Refreshing a User Session Token can be done with the following request:

```
URL: https://augur.example.com/api/unstable/user/session/refresh
arguments:
    refresh_token: [the previously provided refresh token]
    grant_type: "refresh_token"
headers:
    Authorization: Client [your client secret]
```

The authorization server will respond with the following on success:

```
{
    "status": "Validated",
    "access_token": "the new Bearer token",
    "refresh_token": "the new refresh token",
    "token_type": "Bearer",
    "expires": [integer: seconds until this access_token expires]
}
```

The new User Session and refresh tokens should replace the existing tokens, as they may not be the same.

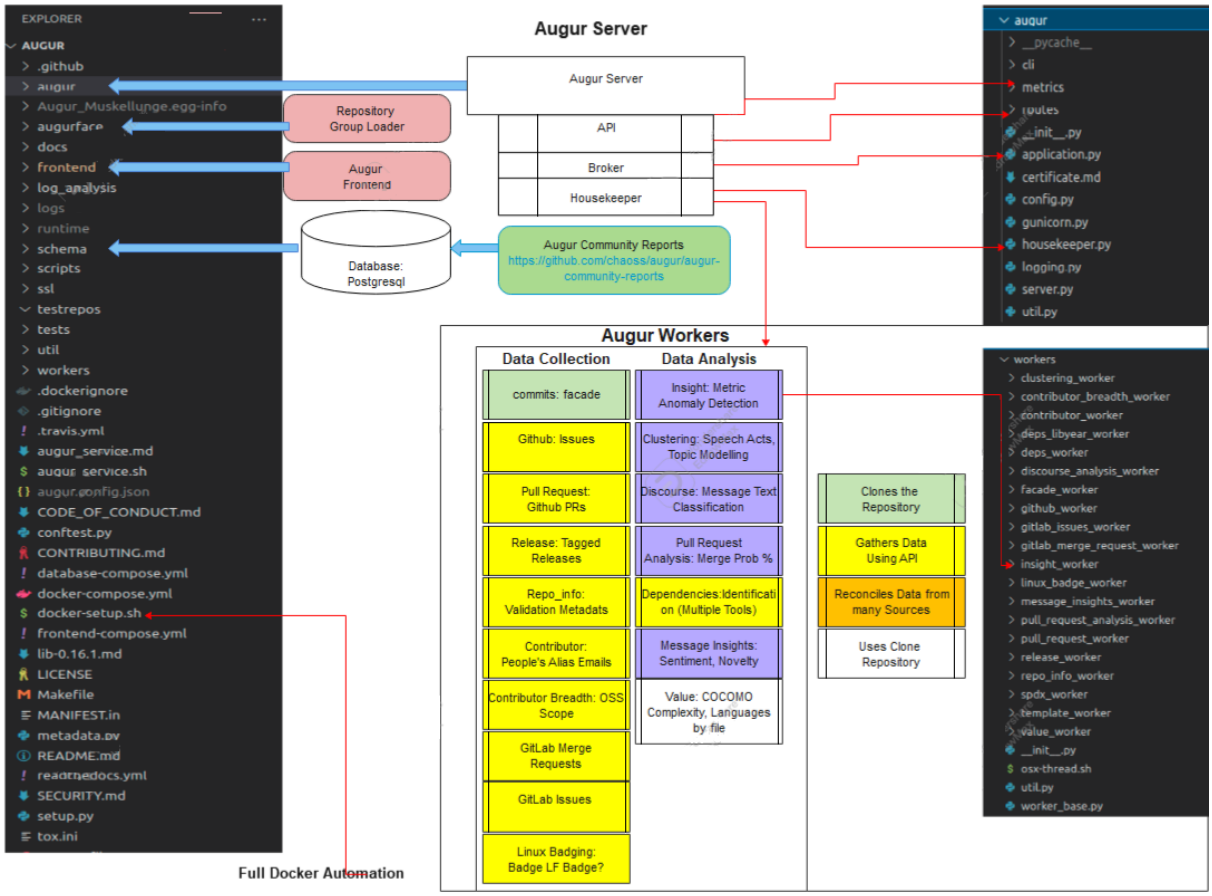
See the rest API documentation for more specific details about these login endpoints.

1.9.3 Making Authenticated Requests

Once the User Session Token has been acquired, authenticated requests must be made using both the Client Secret and the Bearer Token. Authentication credentials must be provided in the `Authorization` header as such:

```
Authorization: Client [Client Secret], Bearer [User Session Token]
```

Please note that both the Client Secret and the User Sesson Token must be included in the Authorization header for authenticated requests


EXPLORER
AUGUR
.github
augur
Augur_Muskellunge.egg-info
augurface
docs
frontend
log_analysis
logs
runtime
schema
scripts
ssl
testrepos
tests
util
workers
.dockerignore
.gitignore
.travis.yml
augur_service.md
augur_service.sh
augur.config.json
CODE_OF_CONDUCT.md
conftest.py
CONTRIBUTING.md
database-compose.yml
docker-compose.yml
docker-setup.sh
frontend-compose.yml
lib-0.16.1.md
LICENSE
Makefile
MANIFEST.in
metadata.py
README.md
readthedocs.yml
SECURITY.md
setup.py
tox.ini
Full Docker Automation
Augur Server
Augur Server
Repository Group Loader
Augur Frontend
API
Broker
Housekeeper
Database: Postgresql
Augur Community Reports
https://github.com/chaoss/augur/augur-community-reports
augur
__pycache__
cli
metrics
routes
__init__.py
application.py
certificate.md
config.py
gunicorn.py
housekeeper.py
logging.py
server.py
util.py
Augur Workers
Data Collection
Data Analysis
commits: facade
Github: Issues
Pull Request: Github PRs
Release: Tagged Releases
Repo_info: Validation Metadats
Contributor: People's Alias Emails
Contributor Breadth: OSS Scope
GitLab Merge Requests
GitLab Issues
Linux Badging: Badge LF Badge?
Insight: Metric Anomaly Detection
Clustering: Speech Acts, Topic Modelling
Discourse: Message Text Classification
Pull Request Analysis: Merge Prob %
Dependencies:Identificati on (Multiple Tools)
Message Insights: Sentiment, Novelty
Value: COCOMO Complexity, Languages by file
Clones the Repository
Gathers Data Using API
Reconciles Data from many Sources
Uses Clone Repository
workers
clustering_worker
contributor_breadth_worker
contributor_worker
deps_libyear_worker
deps_worker
discourse_analysis_worker
facade_worker
github_worker
gitlab_issues_worker
gitlab_merge_request_worker
insight_worker
linux_badge_worker
message_insights_worker
pull_request_analysis_worker
pull_request_worker
release_worker
repo_info_worker
spdx_worker
template_worker
value_worker
__init__.py
osx-thread.sh
util.py
worker_base.py

CHAPTER

TWO

WHAT IS AUGUR?

Augur is a software suite for collecting and measuring structured data about free and open-source software (FOSS) communities.

Augur's main focus is to measure the overall health and sustainability of open source projects, as these types of projects are system critical for nearly every software organization or company. We do this by gathering data about project repositories and normalizing that into our data model to provide useful metrics about your project's health. For example, one of our metrics is Burstiness. Burstiness – how are short timeframes of intense activity, followed by a corresponding return to a typical pattern of activity, observed in a project? This can paint a picture of a project's focus and gain insight into the potential stability of a project and how its typical cycle of updates occurs. There are many more useful metrics, and you can find a full list of them here.

Augur gathers trace data for a group of repositories, normalize it into our data model, and provide a variety of metrics about that data.

This software is developed as part of the CHAOSS (Community Health Analytics Open Source Software) project. Many of our metrics are implementations of the metrics defined by our community. You can find more information about how to get involved on the CHAOSS website.

If you want to see augur in action, you can view CHAOSS's augur instance here.

2.1 Current maintainers

- Derek Howard
- Andrew Brain
- Isaac Milarsky
- John McGinnes
- Sean P. Goggins

2.2 Former maintainers

- Carter Landis
- Gabe Heim
- Matt Snell
- Christian Cmehil-Warn
- Jonah Zukosky

- Carolyn Perniciaro
- Elita Nelson
- Michael Woodruff
- Max Balk

2.3 Contributors

- Dawn Foster
- Ivana Atanasova
- Georg J.P. Link

2.4 GSoC 2022 participants

- Kaxada
- Mabel F
- Priya Srivastava
- Ramya Kappagantu
- Yash Prakash

2.5 GSoC 2021 participants

- Dhruv Sachdev
- Rashmi K A
- Yash Prakash
- Anuj Lamoria
- Yeming Gu
- Ritik Malik

2.6 GSoC 2020 participants

- Akshara P
- Tianyi Zhou
- Pratik Mishra
- Sarit Adhikari
- Saicharan Reddy
- Abhinav Bajpai

2.7 GSoC 2019 participants

- Bingwen Ma
- Parth Sharma

2.8 GSoC 2018 participants

- Keanu Nichols

HTTP ROUTING TABLE

/complexity

GET /complexity/project_blank_lines, 100
GET /complexity/project_comment_lines, 101
GET /complexity/project_file_complexity, 100
GET /complexity/project_files, 101
GET /complexity/project_languages, 101
GET /complexity/project_lines, 100

/contributor_reports

GET /contributor_reports/new_contributors_bar/, 96
GET /contributor_reports/new_contributors_stacked_bar/, 96
GET /contributor_reports/returning_contributors_pie_chart/, 97
GET /contributor_reports/returning_contributors_stacked_bar/, 97

/dei

POST /dei/repo/add, 102
POST /dei/report, 103

/metadata

GET /metadata/contributions_count, 73
GET /metadata/contributors_count, 74
GET /metadata/repo_info, 73

/owner

GET /owner/:owner/repo/:repo, 72

/pull_request_reports

GET /pull_request_reports/Average_PR_duration/, 100
GET /pull_request_reports/PR_counts_by_merged_status/, 98
GET /pull_request_reports/PR_time_to_first_response/, 99
GET /pull_request_reports/average_PR_events_for_closed_PRs/, 99
GET /pull_request_reports/average_comments_per_PR/, 98
GET /pull_request_reports/average_commits_per_PR/, 98
GET /pull_request_reports/mean_days_between_PR_comments/, 99
GET /pull_request_reports/mean_response_times_for_PR/, 98

/repo-groups

GET /repo-groups, 72
GET /repo-groups/:repo_group_id/abandoned-issues, 82
GET /repo-groups/:repo_group_id/aggregate-summary, 77
GET /repo-groups/:repo_group_id/annual-commit-count-ranked, 77
GET /repo-groups/:repo_group_id/annual-commit-count-ranked, 78
GET /repo-groups/:repo_group_id/annual-lines-of-code-count, 78
GET /repo-groups/:repo_group_id/annual-lines-of-code-count, 79
GET /repo-groups/:repo_group_id/average-issue-resolution-t, 74
GET /repo-groups/:repo_group_id/cii-best-practices-badge, 74
GET /repo-groups/:repo_group_id/closed-issues-count, 93
GET /repo-groups/:repo_group_id/code-changes, 83
GET /repo-groups/:repo_group_id/code-changes-lines, 83
GET /repo-groups/:repo_group_id/committers, 74
GET /repo-groups/:repo_group_id/contributors, 84
GET /repo-groups/:repo_group_id/contributors-new, 84
GET /repo-groups/:repo_group_id/fork-count, 75
GET /repo-groups/:repo_group_id/forks, 75
GET /repo-groups/:repo_group_id/issue-backlog, 85

GET /repo-groups/:repo_group_id/issue-comments-mean, 79
GET /repo-groups/:repo_group_id/issue-comments-mean-std, 80
GET /repo-groups/:repo_group_id/issue-duration, 93
GET /repo-groups/:repo_group_id/issue-participants, 85
GET /repo-groups/:repo_group_id/issue-throughput, 86
GET /repo-groups/:repo_group_id/issues-active, 86
GET /repo-groups/:repo_group_id/issues-closed, 87
GET /repo-groups/:repo_group_id/issues-closed-resolution-duration, 87
GET /repo-groups/:repo_group_id/issues-first-time-closed, 87
GET /repo-groups/:repo_group_id/issues-first-time-opened, 88
GET /repo-groups/:repo_group_id/issues-maintainer-response-duration, 89
GET /repo-groups/:repo_group_id/issues-new, 89
GET /repo-groups/:repo_group_id/issues-open-age, 90
GET /repo-groups/:repo_group_id/languages, 76
GET /repo-groups/:repo_group_id/license-count, 77
GET /repo-groups/:repo_group_id/license-coverage, 76
GET /repo-groups/:repo_group_id/license-declared, 76
GET /repo-groups/:repo_group_id/lines-changed-by-author, 82
GET /repo-groups/:repo_group_id/open-issues-count, 94
GET /repo-groups/:repo_group_id/pull-request-acceptance-rate, 80
GET /repo-groups/:repo_group_id/pull-requests-closed-no-merge, 81
GET /repo-groups/:repo_group_id/pull-requests-merge-contributor-new, 90
GET /repo-groups/:repo_group_id/pull-requests-new, 81
GET /repo-groups/:repo_group_id/repo-messages, 101
GET /repo-groups/:repo_group_id/repos, 73
GET /repo-groups/:repo_group_id/review-duration, 90
GET /repo-groups/:repo_group_id/reviews, 91
GET /repo-groups/:repo_group_id/reviews-accepted, 91
GET /repo-groups/:repo_group_id/reviews-declined, 92
GET /repo-groups/:repo_group_id/stars, 95
GET /repo-groups/:repo_group_id/stars-count, 95
GET /repo-groups/:repo_group_id/sub-projects, 92
GET /repo-groups/:repo_group_id/top-committers, 82
GET /repo-groups/:repo_group_id/top-insights, 73
GET /repo-groups/:repo_group_id/watchers, 94
GET /repo-groups/:repo_group_id/watchers-count, 95
GET /repo-groups/:repo_id/abandoned-issues, 80
GET /repo-groups/:repo_id/annual-commit-count-ranked-by-ne
78
GET /repo-groups/:repo_id/annual-commit-count-ranked-by-re
78
GET /repo-groups/:repo_id/annual-lines-of-code-count-ranke
79
GET /repo-groups/:repo_id/annual-lines-of-code-count-ranke
79
GET /repo-groups/:repo_id/issue-duration, 94
GET /repo-groups/:repo_id/languages, 76
GET /repo-groups/:repo_id/license-count, 77
GET /repo-groups/:repo_id/lines-changed-by-author, 83
GET /repo-groups/:repo_id/open-issues-count, 94
GET /repo-groups/:repo_id/pull-requests-new, 81

/repos

GET /repos, 72
GET /repos/:id, 72
GET /repos/:repo_group_id/releases, 94
GET /repos/:repo_id/aggregate-summary, 77
GET /repos/:repo_id/average-issue-resolution-time, 74
GET /repos/:repo_id/cii-best-practices-badge, 74
GET /repos/:repo_id/closed-issues-count, 93
GET /repos/:repo_id/code-changes, 83
GET /repos/:repo_id/code-changes-lines, 84
GET /repos/:repo_id/committers, 75
GET /repos/:repo_id/contributors, 84
GET /repos/:repo_id/contributors-new, 85
GET /repos/:repo_id/fork-count, 75
GET /repos/:repo_id/forks, 75
GET /repos/:repo_id/issue-backlog, 85
GET /repos/:repo_id/issue-comments-mean, 80
GET /repos/:repo_id/issue-comments-mean-std, 80

GET /repos/:repo_id/issue-participants, 85
GET /repos/:repo_id/issue-throughput, 86
GET /repos/:repo_id/issues-active, 86
GET /repos/:repo_id/issues-closed, 87
GET /repos/:repo_id/issues-closed-resolution-duration,
87
GET /repos/:repo_id/issues-first-time-closed,
88
GET /repos/:repo_id/issues-first-time-opened,
88
GET /repos/:repo_id/issues-maintainer-response-duration,
89
GET /repos/:repo_id/issues-new, 89
GET /repos/:repo_id/issues-open-age, 90
GET /repos/:repo_id/license-coverage, 76
GET /repos/:repo_id/license-declared, 76
GET /repos/:repo_id/pull-request-acceptance-rate,
80
GET /repos/:repo_id/pull-requests-closed-no-merge,
81
GET /repos/:repo_id/pull-requests-merge-contributor-new,
90
GET /repos/:repo_id/releases, 94
GET /repos/:repo_id/repo-messages, 101
GET /repos/:repo_id/review-duration, 91
GET /repos/:repo_id/reviews, 91
GET /repos/:repo_id/reviews-accepted, 92
GET /repos/:repo_id/reviews-declined, 92
GET /repos/:repo_id/stars, 95
GET /repos/:repo_id/stars-count, 96
GET /repos/:repo_id/sub-projects, 93
GET /repos/:repo_id/top-committers, 82
GET /repos/:repo_id/watchers, 95
GET /repos/:repo_id/watchers-count, 95

/rg-name

GET /rg-name/:rg_name, 73
GET /rg-name/:rg_name/repo-name/:repo_name,
73

/user

POST /user/session/generate, 101
POST /user/session/refresh, 102