PolicyKit

PolicyKit

Dec 06, 2021

GETTING STARTED

Consider the platforms you use for online communities today. These platforms only offer governance options that are top-down, autocratic, and punitive, involving admins and mods. But what if platforms could provide other types of governance, such as more democratic ones? What if communities could *build for themselves* the governance that suits their needs and values?

PolicyKit empowers online community members to **concisely author a wide range of governance procedures and automatically carry out those procedures** on their home platforms. Inspired by Nobel economist Elinor Ostrom, we've developed a framework that describes governance as a series of *actions* and *policies*, written in short programming scripts. We're now building out an editor, software libraries, and connectors to platforms like Slack, Reddit, and Discord for communities to author actions and policies.

See our ACM UIST 2020 paper and video to learn more. PolicyKit is still in development and will be released soon. Visit our Github to contribute. To receive updates on PolicyKit, sign up for our mailing list here.

CHAPTER

ONE

INSTALLATION AND GETTING STARTED

On this page, we will take you through the process of setting up PolicyKit, both for local development and on an Ubuntu server.

1.1 Getting Started

PolicyKit requires Python 3. Before you install, we recommend that you activate a Python 3+ virtual environment.

To begin, clone the PolicyKit GitHub repository (or your fork) and navigate to the python project root:

```
git clone https://github.com/amyxzhang/policykit.git
cd policykit/policykit
```

From here, run the following commands to install PolicyKit's dependencies:

```
pip install --upgrade pip
pip install -r requirements.txt
```

Next, run the following command to create a file to store your settings and secrets:

```
cp policykit
cp .env.example .env
```

To run PolicyKit in production, you'll need to change some values in the `.env` file such as the `DJANGO_SECRET_KEY` and `SERVER_URL`. For local development, all you need to do is set `DEBUG=true`.

To verify that you have set the PolicyKit server up correctly, run the following command:

```
python manage.py runserver
```

By default, PolicyKit will create a sqlite3 database in the root directory. If you want to use another database, you can edit the `DATABASES` field in `settings.py`.

Run the following command to create and set up the database:

```
python manage.py migrate
```

Open PolicyKit in the browser at http://localhost:8000/main

1.2 Running PolicyKit on a Server

Thus far, we have been run in Ubuntu 18.04 and Ubuntu 20.04, and the below instructions should work for both.

1. Add PolicyKit to the server by uploading the codebase or using `git clone`.
2. Follow this guide to install Python3 and to create a virtual environment for PolicyKit.
3. Install the requirements to the virtual environment with `pip install -r requirements.txt`.
4. Finish the earlier guide to setting up PolicyKit.
5. Make the following additional changes to `.env`:
    - Set the `DJANGO_SECRET_KEY` field. Generate a key with this command:

      ```
      python manage.py shell -c 'from django.core.management import utils;␣
      ↪print(utils.get_random_secret_key())'
      ```

    - Set the `SERVER_URL` field.
    - Set the `ALLOWED_HOSTS` field to point to your host.
    - Make sure `DEBUG` is empty or set to false.
    - You can leave the platform integration API keys/secrets empty for now. Follow the instructions under "Set up Integrations" to set up each integration.
6. If you want to use a database other than dbsqlite3, or if you want to change the database path, update the `DATABASES` object in `settings.py`.
7. Next, run the following command to collect static files into a `static/` folder:

```
python manage.py collectstatic
```

1.2.1 Deploy with Apache web server

Now that you have PolicyKit installed on your server, you can deploy it on Apache web server. Make sure you have a domain dedicated to Policykit that is pointing to your server's IP address.

Note: In the remaining examples, make sure to substitute the following values:

`$POLICYKIT_REPO` is the path to your policykit repository root. (`/policykit`)

`$POLICYKIT_ENV` is the path to your policykit virtual environment. (`/environments/policykit_env`)

`$SERVER_NAME` is your server name. (`policykit.mysite.com`)

---

1. Install apache2

```
sudo apt-get install apache2 libapache2-mod-wsgi-py3
```

2. Create a new apache2 config file:

```
cd /etc/apache2/sites-available
# replace SERVER_NAME (ie policykit.mysite.com.conf)
cp default-ssl.conf SERVER_NAME.conf
```

3. Edit the config file to look like this:

```
<IfModule mod_ssl.c>
        <VirtualHost _default_:443>
                ServerName $SERVER_NAME
                ServerAdmin webmaster@localhost
                Alias /static $POLICYKIT_REPO/policykit/static

                <Directory $POLICYKIT_REPO/policykit/static>
                        Require all granted
                </Directory>

                # Grant access to wsgi.py file. This is the Django server.
                <Directory $POLICYKIT_REPO/policykit/policykit>
                        <Files wsgi.py>
                                Require all granted
                        </Files>
                </Directory>

                WSGIDaemonProcess policykit python-home=$POLICYKIT_ENV python-
↪path=$POLICYKIT_REPO/policykit
                WSGIProcessGroup policykit
                WSGIScriptAlias / $POLICYKIT_REPO/policykit/policykit/wsgi.py
                # .. REST ELIDED
        </VirtualHost>
</IfModule>
```

4. Test your config with `apache2ctl configtest`. You should get a "Syntax OK" as a response.

5. Enable your site:

```
# activate your config
a2ensite /etc/apache2/sites-available/$SERVER_NAME.conf
```

(continues on next page)

(continued from previous page)

```
# disable the default config
sudo a2dissite 000-default-le-ssl.conf
```

6. Get an SSL certificate and set it up to auto-renew using LetsEncrypt:

```
sudo apt install certbot python3-certbot-apache
sudo certbot --apache
```

7. Add the certificates to your `$SERVER_NAME.conf` file:

```
SSLCertificateFile /etc/letsencrypt/live/$SERVER_NAME/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/$SERVER_NAME/privkey.pem
```

8. Reload the config:

```
systemctl reload apache2
```

9. Give the Apache2 user access to the database directory and the logging directory (update paths as needed):

```
sudo chown -R www-data:www-data /var/log/django
sudo chown -R www-data:www-data /var/databases/policykit
```

10. Load your site in the browser and navigate to `/login`. You should see a site titled "Django adminstration" with options to connect to Slack, Reddit, Discourse, and Discord. Before you can install PolicyKit into any of these platforms, you'll need to set the necessary client IDs and client in `private.py`. Follow the setup instructions for each integration in *Integrations*.

    Check for errors at `/var/log/apache2/error.log` and `/var/log/django/debug.log` (or whatever logging path you set in `.env`).

11. Any time you update the code, you'll need to run `systemctl reload apache2` to reload the server.

1.2.2 Set up Celery

PolicyKit uses Celery to run scheduled tasks. Follow these instructions to run a celery daemon on your Ubuntu machine using `systemd`. For more information about configuration options, see the Celery Daemonization.

Create RabbitMQ virtual host

Install RabbitMQ and create a virtual host:

```
sudo apt-get install rabbitmq-server

sudo rabbitmqctl add_user 'username' 'password'
sudo rabbitmqctl add_vhost 'policykit-vhost'
sudo rabbitmqctl set_permissions -p 'policykit-vhost' 'username' '.*' '.*' '.*'
```

In `policykit/settings.py`, set the `CELERY_BROKER_URL` as follows, substituting values for your RabbitMQ username, password, and virtual host:

```
CELERY_BROKER_URL = "amqp://USERNAME:PASSWORD@localhost:5672/CUSTOMVIRTUALHOST"
```

Create celery user

If you don't already have a `celery` user, create one:

```
sudo useradd celery -d /home/celery -b /bin/bash
```

Give the `celery` user access to necessary pid and log folders:

```
sudo useradd celery -d /home/celery -b /bin/bash
sudo mkdir /var/log/celery
sudo chown -R celery:celery /var/log/celery
sudo chmod -R 755 /var/log/celery

sudo mkdir /var/run/celery
sudo chown -R celery:celery /var/run/celery
sudo chmod -R 755 /var/run/celery
```

The `celery` user will also need write access to the Django log file and the database. To give `celery` access, create a group that contains both `www-data` (the apache2 user) and `celery`. For example, if your Django logs are in `/var/log/django` and your database is in `/var/databases`:

```
sudo groupadd www-and-celery
sudo usermod -a -G www-and-celery celery
sudo usermod -a -G www-and-celery www-data

# give the group read-write access to logs
sudo chgrp -R www-and-celery /var/log/django
sudo chmod -R 775 /var/log/django

# give the group read-write access to database
sudo chgrp -R www-and-celery /var/databases
sudo chmod -R 775 /var/databases
```

Create Celery configuration files

Next, you'll need to create three Celery configuration files for PolicyKit:

/etc/conf.d/celery-policykit

```
CELERYD_NODES="w1"

# Absolute or relative path to the 'celery' command:
CELERY_BIN="$POLICYKIT_ENV/bin/celery"

# App instance to use
CELERY_APP="policykit"

# How to call manage.py
CELERYD_MULTI="multi"

# Extra command-line arguments to the worker
CELERYD_OPTS="--time-limit=300 --concurrency=8"

# - %n will be replaced with the first part of the nodename.
```

(continues on next page)

(continued from previous page)

```
# - %I will be replaced with the current child process index
#   and is important when using the prefork pool to avoid race conditions.
CELERYD_PID_FILE="/var/run/celery/%n.pid"
CELERYD_LOG_FILE="/var/log/celery/%n%I.log"
CELERYD_LOG_LEVEL="INFO"

# you may wish to add these options for Celery Beat
CELERYBEAT_PID_FILE="/var/run/celery/policykit_beat.pid"
CELERYBEAT_LOG_FILE="/var/log/celery/policykit_beat.log"
```

/etc/systemd/system/celery-policykit.service

```
[Unit]
Description=Celery Service
After=network.target

[Service]
Type=forking
User=celery
Group=celery
EnvironmentFile=/etc/conf.d/celery-policykit
WorkingDirectory=$POLICYKIT_REPO/policykit
ExecStart=/bin/sh -c '${CELERY_BIN} multi start ${CELERYD_NODES} \
-A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
--logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERYD_OPTS}'
ExecStop=/bin/sh -c '${CELERY_BIN} multi stopwait ${CELERYD_NODES} \
--pidfile=${CELERYD_PID_FILE}'
ExecReload=/bin/sh -c '${CELERY_BIN} multi restart ${CELERYD_NODES} \
-A ${CELERY_APP} --pidfile=${CELERYD_PID_FILE} \
--logfile=${CELERYD_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} ${CELERYD_OPTS}'

[Install]
WantedBy=multi-user.target
```

/etc/systemd/system/celerybeat-policykit.service

```
[Unit]
Description=Celery Beat Service
After=network.target

[Service]
Type=simple
User=celery
Group=celery
EnvironmentFile=/etc/conf.d/celery-policykit
WorkingDirectory=$POLICYKIT_REPO/policykit
ExecStart=/bin/sh -c '${CELERY_BIN} -A ${CELERY_APP}  \
beat --pidfile=${CELERYBEAT_PID_FILE} \
--logfile=${CELERYBEAT_LOG_FILE} --loglevel=${CELERYD_LOG_LEVEL} \
--schedule=/var/run/celery/celerybeat-policykit-schedule'

[Install]
WantedBy=multi-user.target
```

After creating the files (and after any time you change them) run the following command:

```
sudo systemctl daemon-reload
```

Finally, run the following commands to start the celery daemon:

```
sudo service rabbitmq-server start
sudo systemctl start celery-policykit celerybeat-policykit
```

Verify that there are no errors with celery and celerybeat by running these commands:

```
sudo systemctl status celery-policykit
sudo systemctl status celerybeat-policykit
```

Troubleshooting

If celery or celerybeat fail to start up as a service, try running celery directly to see if there are errors in your code:

```
celery -A policykit worker -l info --uid celery
celery -A policykit beat -l info --uid celery --schedule=/var/run/celery/celerybeat-
↪policykit-schedule
```

If celerybeat experiences errors starting up, check the logs at `/var/log/celery/policykit_beat.log`.

1.2.3 Interactive Django Shell

The interactive Django shell can be useful when developing and debugging PolicyKit. Access the Django shell with `python manage.py shell_plus`. Some useful shell commands for development:

```
# List all communities
Community.objects.all()

# List CommunityPlatforms for a specific community
community = Community.objects.first()
CommunityPlatform.objects.filter(community=community)

# Get all pending proposals
Proposal.objects.filter(status="proposed")

# Manually run the policy checking task that is executed on a schedule by Celery
from policyengine.tasks import consider_proposed_actions
consider_proposed_actions()

###### Advanced Commands for debugging Metagov ######
```

(continues on next page)

(continued from previous page)

```
# Access the Metagov Community model
from metagov.core.models import Community as MetagovCommunity
MetagovCommunity.objects.all()
MetagovCommunity.objects.get(slug=community.metagov_slug)

# Access the Metagov Plugin models (1:1 with CommunityPlatform)
Plugin.objects.all()
Slack.objects.all()
Plugin.objects.filter(community__slug=community.metagov_slug)

# Get pending Metagov GovernanceProcesses
GovernanceProcess.objects.filter(status='pending')
GovernanceProcess.objects.filter(plugin__community=metagov_community)
SlackEmojiVote.objects.filter(status='pending', plugin__community__slug="my-slug")
```

1.2.4 Set up Integrations

Before your instance of PolicyKit can be installed onto external platforms, you'll need to go through setup steps for each *integration* that you want to support:

Slack

The Slack integration occurs through Metagov. Follow the setup instructions for the Metagov Slack Plugin to create a new Slack App to use with PolicyKit.

Discord

1. Go to https://discord.com/developers/applications
2. Click "New Application" to create your PolicyKit application
3. Under OAuth2, add the redirect URL `[POLICYKIT_URL]/discord/oauth`
4. Add a new Bot and enable these options:
    - Public Bot
    - Requires OAuth2 Code Grant
    - Presence Intent
    - Server Members Intent
5. Copy the bot token into `DISCORD_BOT_TOKEN` in `private.py` file on your PolicyKit server.
6. On the OAuth2 page, get the Client ID and Client Secret and copy them into `private.py`.
7. Reload apache2: `systemctl reload apache2`
8. To test it out, open `[POLICYKIT_URL]/main` and click "Install PolicyKit to Discord."
9. Now, you should be able to use "Sign in with Discord" to access the PolicyKit dashboard for the community you just installed PolicyKit to.

Discourse

There is no admin setup required for Discourse. Each Discourse community that installs PolicyKit needs to register the PolicyKit auth redirect separately.

Reddit

1. Create a new app at https://www.reddit.com/prefs/apps
2. Set the `REDDIT_CLIENT_SECRET` in `private.py`.
3. Reload apache2: `systemctl reload apache2`

CHAPTER

TWO

DESIGN OVERVIEW

2.1 Data Model

- The top-level Community represents a group of users that may exist on 1 or more platforms.
- Each Community has at least 2 CommunityPlatforms: the initial platform (eg SlackCommunity) and the policykit platform (ConstitutionCommunity). All actions proposed on the PolicyKit platform that are "constitutional" (eg changing a policy, creating a role, etc) are linked to the community's ConstitutionCommunity.
- Each Policy in a community is tied to 1 or more Action Types. For Platform Policies and Constitution Policies, the action type indicates which action type the policy **governs**. For Trigger Policies, the action type indicates which action types trigger the policy to evaluate. When a Policy is evaluated against an Action, a `Proposal` record is created to store any data relevant to the policy evaluation.
- `CommunityRole` is tied to `Community` (not `CommunityPlatform`) because it may determine the ability to propose/execute governable actions, which may occur on any platform. A given `CommunityRole` may be assigned to CommunityUsers on multiple platforms (which in some cases may be the same person, eg "alice123" on Discourse and "alice" on Slack).

2.2 Policy Engine Evaluation Loop

This diagram shows what happens when a new Action is created, and how the Policy engine decides whether to perform a policy evaluation for it. This diagram does not include the policy evaluation checker task, which periodically re-evaluates all Proposals that are in "PROPOSED" state.

CHAPTER

THREE

WRITING POLICIES

Once any new action is proposed by a user, either by invoking it on the PolicyKit website or on a particular community platform, it passes through the Policy engine evaluation loop. It may be helpful to review the *Policy Engine Evaluation Loop* to understand how policy evaluations are triggered.

Policy code is divided into 5 discrete blocks: `filter`, `initialize`, `check`, `notify`, `pass`, and `fail`. Scope is not shared across blocks. See the *Evaluation Context* reference for what's in scope to the policy author in each block.

See *Policy Examples* for example policies that can be downloaded and uploaded into your PolicyKit community.

3.1 Action Types

This field specifies the scope of the policy. For Platform Policies and Constitution Policies, the action type indicates which action type the policy governs. For Trigger Policies, the action type indicates which action types trigger the policy to evaluate. When a Policy is evaluated against an Action, a `Proposal` record is created to store any data relevant to the policy evaluation.

Each community is set up with a **Starter Kit** which defines the **Base Policies** for governing platform and constitution actions. Base policies act as a "fallback policy" and applies to ALL action types. Base policies are overridden by creating more specific Platform and Constitution policies.

3.2 Filter

This function allows the policy author to further filter down the scope of the policy. The function returns `True` if the policy governs the action object passed in as an argument. For instance, if the policy is meant to cover only one type of action, the function can check the `action_type` field of the action object. The policy could also filter on the initiator of the action or even the time of day, if, for example, the community has decided that Friday evenings are a free-for-all in a particular channel.

3.3 Initialize

If `filter` returns true, the action in question is considered in scope for this policy, and we move on to `initialize`. Within this function, the author can specify any code that must be completed once at the start of the policy to set it up.

3.4 Check

This block should return `PROPOSED`, `PASSED`, or `FAILED`.

The `check` function specifies the conditions that need to be met so that an action has passed or failed. For example, it may test whether a vote has reached a quorum, or whether an elected individual has responded to the proposal. When created, all actions have a status of `PROPOSED`. New actions first encounter `check` immediately after `initialize`; this is so that in case the policy can already pass or fail, we can exit the workflow early. For instance, if there was a policy that holds messages containing profanity for review by a moderator, the policy would automatically pass actions that do not contain profanity.

As long as an action is still `PROPOSED`, `check` will run periodically until it returns `PASSED` or `FAILED`. If `check` does not return anything, `PROPOSED` is presumed. For instance, if a policy calls for a vote from users, it may take time for the required number of votes to come in. The policy's `check` function could also specify a maximum amount of time, at which point the action fails.

3.5 Notify

If the policy involves reaching out to one or more community members for input, then the code for notifying members occurs in this function. While policy authors can send messages to users in any function, this function is specifically for notifications soliciting user input. Authors may use the method `initiate_vote` to start a vote on whether or not to pass the action. This function is only run once, after a new action does not return `PASSED` or `FAILED` from the first `check`, so as to not unnecessarily notify users.

3.6 Pass

This function runs if an action is passed (e.g. `check` returned `PASSED`). Code that could go here includes: post-action clean-up tasks such as announcing the outcome to the voters or to the community.

When a Governable Action is passed, the engine automatically executes it, if applicable (see evaluation loop diagram).

3.7 Fail

This function runs if the action fails to pass the policy (e.g. `check` returned `FAILED`). Code that could go here includes: invoking fall-back actions due to failure, or share the outcome privately with the proposer alongside an explanation of why the action failed.

When a Governable Action is failed, the engine automatically executes it, if applicable (see evaluation loop diagram).

CHAPTER

FOUR

PLATFORM INTEGRATIONS

PolicyKit is an application that sits on its own server. However, it would be prohibitive if users of a social platform needed to go to PolicyKit for every governance task, such as proposing an action or voting on a proposal. In addition, as PolicyKit needs to enforce policies, it must have a way of stopping and allowing actions that are carried out on the platform itself, since the platform already has an existing governance that PolicyKit must supersede. These capabilities are defined in platform integration libraries that can be developed for any platform to connect with PolicyKit. Once a single developer has created an integration using a platform's web API, any community on that platform can use PolicyKit.

In order to install PolicyKit to a community, there must be an **authentication mechanism**, such as OAuth, for at least one admin or mod account to give access to PolicyKit so that it may govern a broad set of actions, including privileged ones. In addition, each platform integration supports **one or more** of these capabilities:

- **Actions** are API requests to perform some action on a platform, such as sending a message to users. Some platforms don't require authentication (like SourceCred), others require API keys to be uploaded (Loomio, Open Collective) and others need to be authenticated via an OAuth flow (Slack, GitHub).
- **Trigger Actions** are platform events that can be used as policy triggers. Typically these are received via webhooks, so they may require registering a PolicyKit webhook URL for your community on the external platform. For platforms that don't support webhooks, some integrations have a polling mechanism to fetch data periodically and create triggers from new events.
- **Governable Actions** are a PolicyKit construct that combines "actions" and "trigger actions." A Governable Action can be reverted and re-executed, which allows PolicyKit to "govern" that capability on the platform. Policies that govern platform actions are called **Platform Policies** (see *Policy Examples*). This capability may require an admin account to give access to PolicyKit so that it may govern a broad set of actions, including privileged ones. All Governable Actions can also be used as triggers for Trigger Policies.
- **Voting** is the ability to perform a vote on an external platform and capture the result.

Once you have installed PolicyKit to one platform, you can enable additional integrations on the Settings page. Only users with the `Integration Admin` role are able to add and remove integrations.

Note: Some integrations require one-time setup process by a PolicyKit server admin. If you deploying your own instance of PolicyKit, see *Installation and Getting Started* for instructions. Platforms like Slack and Discord require an initial setup process to create bots/apps and store their Client IDs and secrets on the PolicyKit server.

Here is an overview of the capabilities supported by each platform integration:

4.1 Slack

The authentication mechanism for the Slack Integration is **OAuth**. The installing user must be an admin on Slack in order to install PolicyKit.

Actions		Write policies that perform actions on Slack, such as posting messages.
Trigger Actions		Write Trigger Policies that are triggered by events that occurred on Slack (e.g. "when a Slack channel is renamed, update the generated welcome post")
Governable Actions		Write Platform Policies that govern Slack actions (e.g. "only users with X role can rename Slack channels")
Voting		Write policies that perform boolean- or single-choice voting in Slack channels or DMs.

4.2 Open Collective

The authentication mechanism for the Open Collective Integration is an **API Key** for a user with admin access to the collective. It also requires registering a webhook in Open Collective. Follow instructions on the setup page.

Actions		Write policies that perform actions on Open Collective, such as processing expenses or posting comments.
Trigger Actions		Write Trigger Policies that are triggered by events that occurred on the Open Collective platform (e.g. "when an expense is created, start a vote on Slack")
Governable Actions		
Voting		

4.3 Loomio

The authentication mechanism for the Loomio Integration is an **API Key**. It also requires registering a webhook in Loomio. Follow instructions on the setup page.

Actions		
Trigger Actions		
Governable Actions		
Voting		Write policies that perform votes on Loomio.

4.4 GitHub

The authentication mechanism for the GitHub Integration is **OAuth**.

Actions		
Trigger Actions		Write Trigger Policies that are triggered by events that occurred on GitHub (e.g. "when a new issue is created, post about it on Slack if certain conditions are met").
Governable Actions		
Voting		Write policies that perform votes on Github.

4.5 SourceCred

There is no authentication mechanism for the SourceCred Integration. The SourceCred server must be public. The only thing this integration supports is fetching cred and grain values.

Actions		Write policies that fetch SourceCred values from the configured SourceCred instance.
Trigger Actions		
Governable Actions		
Voting		

4.6 Reddit

Actions		Write policies that perform actions on Reddit, such as posting messages.
Trigger Actions		Write Trigger Policies that are triggered by events that occurred on Reddit.
Governable Actions		Write Platform Policies that govern Reddit posting
Voting		Write policies that perform boolean voting on a Reddit thread

4.7 Discord

The authentication mechanism for the Discord Integration is **OAuth**. The installing user must be an admin on Discord in order to install PolicyKit.

Actions		Write policies that perform actions on Discord, such as posting messages.
Trigger Actions		Write Trigger Policies that are triggered by events that occurred on Discord (e.g. "when a user posts a message in a certain channel, do something)
Governable Actions		Write Platform Policies that govern Discord actions (e.g. "only users with X role can post in this Discord channel")
Voting		Write policies that perform boolean voting in a Discord channel.

4.8 Discourse

The authentication mechanism for the Discourse Integration is **OAuth**. This integration requires a Discourse admin to do some setup steps in Discourse before PolicyKit can be installed.

Actions		Write policies that create posts and topics on Discourse.
Trigger Actions		Write Trigger Policies that are triggered by events that occurred on Discourse (e.g. "when a user posts a new topic in a certain category, do something)
Governable Actions		Write Platform Policies that govern Discourse actions (e.g. "only users with X amount of Cred can post on this Discourse topic")
Voting		

4.8.1 Setting up your Discourse community

You can set up a Discourse community either by running a server that hosts a community locally or by creating a community hosted remotely by Discourse.org. To host a community remotely, you can press "Start Trial" on this page and follow the instructions to set up a community. Discourse.org offers free 14 day trials, which can be extended by contacting support.

Once the site is up and running, you need to configure a few settings to enable PolicyKit to interact with your site. On the site homepage, log in as your admin account and enter the Settings menu (located on the top right of the homepage). On the left sidebar, select the User API page. On this page, you should set / verify the following settings:

- **allow user api keys**: `checked`
- **allow user api key scopes**: Select the scopes you want to enable here. Possible scopes: `read`, `write`, `message_bus`, `push`, `notifications`, `session_info`, `one_time_password`. Recommend allowing all the scopes for full usability of PolicyKit.
- **min user level for user api key**: `0`
- **allowed user api auth redirects**: Add an entry: `[POLICYKIT_URL]/discourse/auth`. (example: `https://policykit.org/discourse/auth`)

4.8.2 Installing PolicyKit to your Discourse community

On the login page, select "Install PolicyKit to Discourse". On the Configure screen that appears, enter the full URL of your Discourse community (example: `https://policykit.trydiscourse.com`). On the next screen that appears, you must approve PolicyKit's authorization to access your Discourse community. On the third and final screen, you must select a Starter Kit system of governance, which will initialize your community with the selected system of governance.

For testing purposes, we recommend trying out the Testing Starter Kit, which will give all members in the community complete access to PolicyKit action. For more experienced PolicyKit users who are hoping to use PolicyKit with an existing community, we recommend trying out one of the other more restrictive Starter Kits.

Once you have selected a Starter Kit, you will be redirected back to the login page. If PolicyKit was installed correctly, you should see a text message near the top saying "Successfully added PolicyKit!". If you see this success message, you are all set to sign in to your Discourse community's dashboard.

4.8.3 Signing in to your PolicyKit dashboard

On the login page, select "Sign in with Discourse". This will display a screen asking "Which Discourse community would you like to sign into?" In the text box, enter the full URL of your Discourse community (example: `https://policykit.trydiscourse.com`) and press Continue. Once again, you must approve PolicyKit's authorization to access your Discourse community. After approving the request, you should be in! You should now be able to see your PolicyKit dashboard and use all the features of PolicyKit with your Discourse community.

4.9 Metagov (experimental)

PolicyKit uses Metagov to integrate with and and govern a range of external platforms. PolicyKit exposes some generic tools to interact with any available Metagov plugins. These are experimental and typically would only be used if you are developing a new Metagov Plugin that doesn't yet have a PolicyKit integration.

4.9.1 Webhook Trigger Action

All events received from Metagov generate a generic trigger action. To write a policy triggered by this generic action, select `Webhook Trigger Action` in the action types dropdown. Use the `filter` block to choose which event type your policy is triggered by. The event type is stored at `action.event_type`, and any additional data is stored as a dict at `action.data`.

```
# "filter" block

return action.event_type == 'opencollective.expense_created'
```

4.9.2 Performing actions

Platform policies have access to a `metagov` client that can be used to perform actions that are defined on a Metagov Plugin. Policy authors can only use actions that are defined in Plugins that are currently enabled in their community.

```
# "check" block

parameters = {"low": 0, "high": 10}
response = metagov.perform_action("randomness.random-int", parameters)
if response and response.get('value') >  5:
    return PASSED
else:
    return FAILED
```

4.9.3 Performing governance processes

Platform policies can use the `metagov` client to perform asynchronous governance processes. Here's a partial example of a policy that uses the `loomio.poll` process to perform a vote.

```
# "notify" block kicks off the process

import datetime

closing_at = (action.proposal.proposal_time + datetime.timedelta(days=3)).strftime("
↪%Y-%m-%d")
result = metagov.start_process("loomio.poll", {
    "title": "Agree or disagree?",
    "options": ["agree", "disagree"],
    "closing_at": closing_at
})
poll_url = result.get('poll_url')
```

```
# "check" block polls for the process outcome

```

(continues on next page)

(continued from previous page)

```
result = metagov.get_process()
if result.status != "completed":
    return # still processing
if result.errors:
    return FAILED
if result.outcome:
    agree_count = result.outcome.get("agree")
    disagree_count = result.outcome.get("disagree")
    return PASSED if agree_count > disagree_count else FAILED
return FAILED
```

CHAPTER

FIVE

POLICY EXAMPLES

This is a library of example policies to get started.

5.1 Vote on renaming Slack channels

When a channel is renamed in Slack, revert the change and perform a vote. If the vote passes, execute the rename.

Download Policy

Policy Kind: `Platform`

Action Types: `slackrenameconversation`

Filter:

```
return True
```

Initialize:

```
pass
```

Check:

```
yes_votes = proposal.get_yes_votes().count()
no_votes = proposal.get_no_votes().count()
logger.debug(f"{yes_votes} for, {no_votes} against")
if yes_votes >= 1:
  return PASSED
elif no_votes >= 1:
  return FAILED

logger.debug("No votes yet....")
return PROPOSED
```

Notify:

```
message = f"Should this channel be renamed to #{action.name}? Vote with :thumbsup: or␣
↪:thumbsdown: on this post."
slack.initiate_vote(proposal, template=message)
```

Pass:

```
text = f"Proposal to rename this channel to #{action.name} passed."
slack.post_message(text=text, channel=action.channel, thread_ts=proposal.community_
↪post)
action.execute()
```

Fail:

```
text = f"Proposal to rename this channel to #{action.name} failed."
slack.post_message(text=text, channel=action.channel, thread_ts=proposal.community_
↪post)
```

5.2 Vote on OpenCollective expenses in Slack with restricted voting

Posts OpenCollective expenses to Slack channel to be voted on. If expense has three or more yes votes and one or less no votes after three hours, approves, otherwise waits until three days has passed and approves if there are any yes votes and no no votes, otherwise rejects. Once the vote is resolved, posts to both Slack thread and the OpenCollective expense thread with vote results. Restricts voting on OpenCollective expenses to eligible voters.

Download Policy

Policy Kind: `Trigger`

Action Types: `expensecreated`

Filter:

```
return True
```

Initialize:

```
pass
```

Check:

```
if not proposal.community_post:
  return None #the vote hasn't started yet

# Check the status of the vote and evaluate a closing condition.

yes_votes = proposal.get_yes_votes().count()
no_votes = proposal.get_no_votes().count()
#logger.debug(f"{yes_votes} for, {no_votes} against")

import datetime
time_elapsed = proposal.get_time_elapsed()

# we never close a vote before three hours
if time_elapsed < datetime.timedelta(hours=3):
  return None

# if there are three or more votes for, and one or less against, it passes
if yes_votes >= 2 and no_votes <= 1:
  return PASSED

# if there's more than one vote against, and less than three yes votes, it fails
if no_votes > 0:
```

(continues on next page)

(continued from previous page)

```
  return FAILED

# if it's been three days, and there are any yeses and no nos, pass, otherwise fail
if time_elapsed > datetime.timedelta(days=3):
  logger.info("Ending based on time")
  reached_threshold = yes_votes > 0 and no_votes == 0
  return PASSED if reached_threshold else FAILED

return PROPOSED # still pending
```

Notify:

```
# Start a vote on Slack. Only 2 users are eligible to vote.
discussion_channel = "C0177HZTXXX"

eligible_voters = ["UABC123", "UABC999"]  # Slack member IDs for Alice and Bob
message = f"Vote on whether to approve <{action.url}|this request> for funds: {action.
↪description}"
slack.initiate_vote(proposal, template=message, channel=discussion_channel,␣
↪users=eligible_voters)

# Start a discussion thread on the voting message
slack.post_message(text="Discuss here! :meow-wave:", channel=discussion_channel,␣
↪thread_ts=proposal.community_post)

# Add a comment to the expense on Open Collective with a link to the Slack vote
import json
link = f"<a href='{json.loads(proposal.governance_process_json)['outcome']['url']}'>
↪on Slack</a>"
text = f"Thank you for submitting a request! A vote has been started {link}."
opencollective.post_message(text=text, expense_id=action.expense_id)
```

Pass:

```
# approve the expense
opencollective.process_expense(action="APPROVE", expense_id=action.expense_id)

yes_votes = proposal.get_yes_votes().count()
no_votes = proposal.get_no_votes().count()
message = f"Expense approved. The vote passed with {yes_votes} for and {no_votes}␣
↪against."

# comment on the expense
opencollective.post_message(text=message, expense_id=action.expense_id)

# update the Slack thread
discussion_channel = "C0177HZTXXX"
slack.post_message(text=message, channel=discussion_channel, thread_ts=proposal.
↪community_post)
```

Fail:

```
# reject the expense
opencollective.process_expense(action="REJECT", expense_id=action.expense_id)

yes_votes = proposal.get_yes_votes().count()
no_votes = proposal.get_no_votes().count()
```

(continues on next page)

(continued from previous page)

```
message = f"Expense rejected. The vote failed with {yes_votes} for and {no_votes}␣
↪against."

# comment on the expense
opencollective.post_message(text=message, expense_id=action.expense_id)

# update the Slack thread
discussion_channel = "C0177HZTXXX"
slack.post_message(text=message, channel=discussion_channel, thread_ts=proposal.
↪community_post)
```

5.3 Vote on OpenCollective expenses in Loomio

When an expense is submitted in Open Collective, start a proposal on Loomio. The vote closes when 3 days have passed. Approve or reject the OC expense based on the vote results, and post a comment on the OC expense with the vote count.

Download Policy

Policy Kind: `Trigger`

Action Types: `expensecreated`

Filter:

```
return True
```

Initialize:

```
pass
```

Check:

```
if not proposal.is_vote_closed:
  # Proposal is still pending in Looomio
  return PROPOSED

consent_count = proposal.get_choice_votes(value="consent").count()
objection_count = proposal.get_choice_votes(value="objection").count()

oc_action = None
oc_message = None

if objection_count > 0:
  # If 1 or more people objected, reject the expense.
  oc_action = "REJECT"
  oc_message = f"Expense rejected. Loomio proposal had {objection_count} objection{'s
↪' if objection_count > 1 else ''}."
elif consent_count > 2:
  # If enough people consented, approve the expense.
  oc_action = "APPROVE"
  oc_message = f"Expense approved. {consent_count} members consented on Loomio."
else:
  # If nobody objected, but not enough people consented, leave the expense open.
  oc_message = f"Consensus was not reached on Loomio. {consent_count} members␣
↪consented and {objection_count} members objected."
```

(continues on next page)

(continued from previous page)

```
# Process the expense (if applicable)
if oc_action:
  opencollective.process_expense(action=oc_action, expense_id=action.expense_id)

# Comment on the expense
opencollective.post_message(text=oc_message, expense_id=action.expense_id)

return PASSED
```

Notify:

```
import datetime
closing_at = proposal.proposal_time + datetime.timedelta(days=3)

# Start vote on Loomio
loomio.initiate_vote(
  proposal,
  title=f"Expense '{action.description}'",
  poll_type="proposal",
  details=f"Submitted on Open Collective: {action.url}",
  options=["consent", "objection"],
  closing_at=closing_at,
)

loomio_poll_url = proposal.community_post
logger.debug(f"Vote started at {loomio_poll_url}")


# Record start of vote on Open Collective
link = f"<a href='{loomio_poll_url}'>on Loomio</a>"
text = f"Thank you for submitting a request! A vote has been started {link}."
opencollective.post_message(text=text, expense_id=action.expense_id)
```

Pass:

```
pass
```

Fail:

```
pass
```

5.4 Minimum SourceCred value required for creating Discourse Topic

Only users with at least 300 cred can create new topics in a certain Discourse category. If a user does not pass the threshold, the topic gets deleted (aka the 'governed action' is reverted).

Download Policy

Policy Kind: `Platform`

Action Types: `discoursecreatetopic`

Filter:

```
return action.category == 8
```

Initialize:

```
pass
```

Check:

```
value = sourcecred.get_cred(username=action.initiator.username)
return PASSED if value > 300 else FAILED
```

Notify:

```
pass
```

Pass:

```
pass
```

Fail:

```
pass
```

5.5 Don't allow posts in Slack channel

This could be extended to add any logic to determine who can post in a given channel. Posts in the channel are deleted, and the user is notified about why it happened.

Download Policy

Policy Kind: `Platform`

Action Types: `slackpostmessage`

Filter:

```
return action.channel == "C025LDZ76R3"
```

Initialize:

```
pass
```

Check:

```
return FAILED
```

Notify:

```
pass
```

Pass:

```
pass
```

Fail:

```
# create an ephemeral post that is only visible to the poster
message = f"Post was deleted because of policy '{policy.name}'"
slack.post_message(
  channel=action.channel,
  users=[action.initiator],
  post_type="ephemeral",
  text=message
)
```

5.6 Consensus vote in Slack with restricted voting + reminder

Allows users to trigger a vote in the #core-group slack channel. Only core group members are able to cast votes. Following community policy, all votes must pass via consensus. One day before a policy is about to be resolved by default, reminds core-group members that the vote is still open.

Download Policy

Policy Kind: `Trigger`

Action Types: `slackpostmessage`

Filter:

```
return action.text.startswith("core-group-vote")
```

Initialize:

```
pass
```

Check:

```
# CONSTANTS
CHANNEL = "C0ABC123"
LENGTH_OF_VOTE_IN_DAYS = 5
MINIMUM_YES_VOTES = 2
MAXIMUM_NO_VOTES = 0
ELIGIBLE_VOTERS = ["U01", "U02", "U03", "U04", "U05", "U06"]

# save for later
proposal.data.set("eligible_voters", ELIGIBLE_VOTERS)
proposal.data.set("discussion_channel", CHANNEL)

if not proposal.community_post:
  return PROPOSED #the vote hasn't started yet

# vote info
consent_votes = proposal.get_choice_votes(value="consent")
object_votes = proposal.get_choice_votes(value="object")
abstain_votes = proposal.get_choice_votes(value="abstain")
logger.debug(f"{consent_votes} for, {object_votes} against, {abstain_votes} abstain")

# if everyone has consented or abstained, the vote passes
if len(consent_votes) + len(abstain_votes) == len(ELIGIBLE_VOTERS):
  return PASSED

import datetime
```

(continues on next page)

(continued from previous page)

```
# if voting is almost over, remind people who haven't voted
reminder_sent = proposal.data.get('reminder_sent')
if not reminder_sent and proposal.get_time_elapsed() > datetime.timedelta(days=LENGTH_
↪OF_VOTE_IN_DAYS-1):
  already_voted = proposal.get_choice_votes().values_list("user__username")
  already_voted_list = [voter[0] for voter in already_voted]
  nonvoter_list = [f"<@{uid}>" for uid in ELIGIBLE_VOTERS if uid not in already_voted_
↪list]
  nonvoter_string = ", ".join(nonvoter_list)
  logger.debug(f"Eligible voters: {ELIGIBLE_VOTERS}; Already voted: {already_voted};␣
↪Nonvoter list: {nonvoter_list}")
  reminder_msg = f"There is one day left to vote on this proposal. {nonvoter_string}␣
↪have not voted yet."
  slack.post_message(text=reminder_msg, channel=action.channel, thread_ts=proposal.
↪community_post)
  proposal.data.set("reminder_sent", True)

# if time's up, resolve
if proposal.get_time_elapsed() > datetime.timedelta(days=LENGTH_OF_VOTE_IN_DAYS):

  if object_votes.count() > MAXIMUM_NO_VOTES:
    return FAILED
  if consent_votes.count() < MINIMUM_YES_VOTES:
    return FAILED
  return PASSED

return PROPOSED # still pending
```

Notify:

```
# Start a vote on Slack
discussion_channel = proposal.data.get('discussion_channel')
eligible_voters = proposal.data.get('eligible_voters')
proposal_text = action.text.replace("core-group-vote ", "").strip()
message = f"<@channel>, please respond to this proposal: {proposal_text} If all core␣
↪group members respond by consenting or abstaining, or if after five days there are␣
↪no objections and at least two consents, the proposal passes."
slack.initiate_vote(proposal, template=message, channel=discussion_channel,
                    users=eligible_voters, options=["consent", "object", "abstain"])
slack.post_message(text="Discuss here! :meow-wave:", channel=discussion_channel,␣
↪thread_ts=proposal.community_post)
```

Pass:

```
# Announce result on thread
text = f"<@{action.initiator.username}>'s proposal passed!"
slack.post_message(text=text, channel=action.channel, thread_ts=proposal.community_
↪post)
```

Fail:

```
# Announce result on thread
text = f"<@{action.initiator.username}>'s proposal failed."
slack.post_message(text=text, channel=action.channel, thread_ts=proposal.community_
↪post)
```

5.7 All Constitution Actions need 1 approval in Slack

All constitution actions trigger a Slack vote. In order to pass, the proposal must get at least 1 approval within 3 hours. If the proposal gets any downvotes, it fails.

Download Policy

Policy Kind: `Constitution`

Filter:

```
return True
```

Initialize:

```
pass
```

Check:

```
if proposal.get_yes_votes().count() > 0:
  return PASSED

if proposal.get_no_votes().count() > 0:
  return FAILED

import datetime
if proposal.get_time_elapsed() > datetime.timedelta(hours=3):
  logger.debug(f"Failing proposed constitution change, no approvals in 3 hours")
  return FAILED

return PROPOSED
```

Notify:

```
message = f"'{action}' proposed by {action.initiator.readable_name}"
governance_channel = "C0001"
slack.initiate_vote(proposal, template=message, channel=governance_channel)
```

Pass:

```
message = f"Constitutional change accepted!"
governance_channel = "C0001"
slack.post_message(text=message, channel=governance_channel, thread_ts=proposal.
↪community_post)
```

Fail:

```
message = f"Constitutional change rejected."
governance_channel = "C0001"
slack.post_message(text=message, channel=governance_channel, thread_ts=proposal.
↪community_post)
```

5.8 All Constitution Actions Pass and get posted in Slack

All constitution actions pass automatically and get posted to a governance channel in Slack. Constitution actions are changes to policies, roles, and documents.

Download Policy

Policy Kind: `Constitution`

Filter:

```
return True
```

Initialize:

```
pass
```

Check:

```
return PASSED
```

Notify:

```
pass
```

Pass:

```
message = f"Constitution change '{action}' proposed by {action.initiator.readable_
↪name} passed."
governance_channel = "C000111"
slack.post_message(text=message, channel=governance_channel)
```

Fail:

```
pass
```

5.9 Discord Message Filter

In this tutorial, we will introduce policy creation by creating a policy that filters messages for a set of banned words.

```
Note: In this tutorial, and the following tutorials, we will make use of the␣
↪DiscordIntegration.
If you are new to PolicyKit, we recommend following along in the DiscordIntegration␣
↪so as not to
become lost. However, it shouldn't pose too much of a challenge to emulate the steps␣
↪in this
tutorial in another integration, if you are up to the task.
```

To begin, we must log into the PolicyKit dashboard. You can use either our test server at https://policykit.org/main/ or your own custom PolicyKit server. To set up PolicyKit with your local Discord guild, please see our tutorial on setting up PolicyKit with Discord. Once you have finished setting up PolicyKit with Discord, you should install PolicyKit to your Discord server. For practice purposes, you should use the Testing starter kit, as it will allow you to instantly pass any policy you propose. When you have installed PolicyKit to your Discord server, you can sign in with Discord to view the PolicyKit dashboard.

From there, you should click the Propose Action button on the top right of the dashboard. On the following Actions screen, you should click the Platform Policies menu to drop down the list of platform policy actions. Select the Add Platform Policy option to view the Policy Editor.

Finally, you will be on the Policy Editor page, and we can begin creating our policy! First, choose a name and description for your policy. You can leave the description blank if you wish.

In PolicyKit, incoming actions are checked against the Filter block of each active policy. Each policy is only executed on the action if the policy's Filter block returns True. The Filter block returns False by default.

We only want our Message Filter policy to run on actions which are messages posted to the Discord channel we are monitoring. To check if the action is a posted message, we can check a property of the `action` object called `action_type`. The codename for posting a message on Discord is `"discordpostmessage"`. Thus, our Filter block is:

```
if action.action_type == "discordpostmessage":
  return True
```

We want to check all posted messages to see if they contain any blacklisted words. For example, suppose we want to ban the words "minecraft", "amazon", and "facebook" (due to repeated spam). In the Check block of the policy, we can check the `text` property of the `action` object and see if a substring of the text is a banned word. If so, the policy will fail the action (`return FAILED`). Otherwise, it will pass the action (`return PASSED`). If we don't return anything, `PROPOSED` will be returned by default, representing an intermediate state. Our Check block is:

```
for banned_word in ["minecraft", "amazon", "facebook"]:
  if banned_word in action.text:
    return FAILED
return PASSED
```

All other fields can be left as their defaults; there is no need to modify them. Once you have finished typing this code into the po

- Check `action.text` against Google's Perspective API (which checks for spam, hate speech, etc.).
- Instead of removing posts which violate the Message Filter, allow the community to vote on whether the post should be shown. Or wait for moderator approval before displaying the post.

Great job! You have created your first policy.

5.10 Discord Dice Rolling

This will allow the user to roll a dice by typing the following command: !roll d[num_faces] +[num_modifier]

where num refers to a positive non-zero integer value. This command simulates rolling a dice with num_faces faces (e.g. d100 is a dice with 100 faces). The user can optionally add a modifier, which adds an integer value to the result of the dice roll. For example, +7 would add 7 to the result of the dice roll.

Filter:

```
if action.action_type != "DiscordPostMessage":
  return False
tokens = action.text.split()
if tokens[0] != "!roll":
  return False
if len(tokens) < 2 or len(tokens) > 3:
  discord.post_message(text='not right number of tokens: should be 2 or 3', channel =
↪"733209360549019688")
```

(continues on next page)

(continued from previous page)

```
  return False
return True
```

Initialize: `pass`

Check:

```
import random
tokens = action.text.split()
channel = 733209360549019691
if tokens[1][0] != "d":
  duscird.post_message(text='not have d', channel=channel)
  return FAILED
if tokens[1][1:].isnumeric() == False:
  duscird.post_message(text='not numeric num faces', channel=channel)
  return FAILED
num_faces = int(tokens[1][1:])
num_modifier = 0
if len(tokens) == 3:
  if tokens[2][0] != "+":
    duscird.post_message(text='not have +', channel=channel)
    return FAILED
  if tokens[2][1:].isnumeric() == False:
    duscird.post_message(text='not numeric num modifier', channel=channel)
    return FAILED
  num_modifier = int(tokens[2][1:])
roll_unmodified = random.randint(1, num_faces)
roll_modified = roll_unmodified + num_modifier
proposal.data.set('roll_unmodified', roll_unmodified)
proposal.data.set('roll_modified', roll_modified)
return PASSED
```

Notify: `pass`

Pass:

```
text = 'Roll: ' + str(proposal.data.get('roll_unmodified')) + " , Result: " +␣
↪str(proposal.data.get('roll_modified'))
discord.post_message(text=text, channel = "733209360549019688")
```

Fail:

```
text = 'Error: Make sure you format your dice roll command correctly!'
discord.post_message(text=text, channel = "733209360549019688")
```

5.11 Discord Lottery / Raffle

Allow users to vote on a "lottery" message, pick a random user as the lottery winner, and automatically notify the channel.

Filter:

```
if action.action_type != "DiscordPostMessage":
  return False
tokens = action.text.split(" ", 1)
```

(continues on next page)

(continued from previous page)

```
if tokens[0] != "!lottery":
  return False
if len(tokens) != 2:
  discord.post_message(text='need a lottery message', channel = "733209360549019688")
  return False
proposal.data.set('message', tokens[1])
return True
```

Initialize: `pass`

Notify:

```
message = proposal.data.get('message')
discord.initiate_vote(proposal, template=message, channel = "733209360549019688")
```

Check:

```
all_votes = proposal.get_yes_votes()
num_votes = len(all_votes)
if num_votes >= 3:
  return PASSED
```

Pass:

```
import random

all_votes = proposal.get_yes_votes()
num_votes = len(all_votes)
winner = random.randint(0, num_votes)
winner_name = all_votes[winner].user.readable_name
message = "Congratulations! " + winner_name + " has won the lottery!"
discord.post_message(text=message, channel = "733209360549019688")
```

Fail: `pass`

CHAPTER

SIX

EVALUATIONCONTEXT

class policyengine.engine.**EvaluationContext**(*proposal*)

Class to hold all variables available in a policy evaluation. All attributes on this class are in scope and can be used by the policy author.

proposal

The proposal representing this evaluation.

Type *Proposal*

action

The action that triggered this policy evaluation.

Type *BaseAction*

policy

The policy being evaluated.

Type *Policy*

slack

Type SlackCommunity

discord

Type DiscordCommunity

discourse

Type DiscourseCommunity

reddit

Type RedditCommunity

github

Type GithubCommunity

opencollective

Type OpencollectiveCommunity

loomio

Type LoomioCommunity

sourcecred

Type SourcecredCommunity

metagov
Metagov library for performing enabled actions and processes.

Type Metagov

logger
Logger that will log messages to the PolicyKit web interface.

Type logging.Logger

CHAPTER

SEVEN

COMMUNITY

7.1 Community

class policyengine.models.**Community**(**args*, ***kwargs*)
A Community represents a group of users. They may exist on one or more online platforms.

get_roles()
Returns a QuerySet of all roles in the community.

get_platform_policies(*is_active=True*)
Returns a QuerySet of all platform policies in the community.

get_constitution_policies(*is_active=True*)
Returns a QuerySet of all constitution policies in the community.

get_trigger_policies(*is_active=True*)
Returns a QuerySet of all trigger policies in the community.

get_documents(*is_active=True*)
Returns a QuerySet of all documents in the community.

7.2 CommunityPlatform

class policyengine.models.**CommunityPlatform**(**args*, ***kwargs*)
A CommunityPlatform represents a group of users on a single platform.

platform = None
The name of the platform ('Slack', 'Reddit', etc.).

community_name
The name of the community.

community
The `Community` that this CommunityPlatform belongs to.

initiate_vote(*proposal*, *users=None*)
Initiates a vote on whether to pass the action that is currently being evaluated.

Parameters

- **proposal** – The `Proposal` that is being run.
- **users** –
- **users who should be notified.**(*The*) –

get_platform_policies()
Returns a QuerySet of all platform policies for this `Community`.

get_roles()
Returns a QuerySet of all roles in the community.

get_users()
Returns a QuerySet of all users in the community on this platform.

CHAPTER

EIGHT

ACTIONS

8.1 BaseAction

class policyengine.models.**BaseAction**(**args*, ***kwargs*)

Base Action

community

The `CommunityPlatform` in which the action occurred (or was proposed). If proposed through the PolicyKit app, this is the community that the proposing user was authenticated with.

initiator

The `CommunityUser` who initiated the action. May not exist if initiated by PolicyKit.

is_bundled

True if the action is part of a bundle.

kind = None

Kind of action. One of 'platform' or 'constitution' or 'trigger'. Do not override.

property action_type

The type of action (such as 'slackpostmessage' or 'policykitaddcommunitydoc').

8.2 ConstitutionAction

Extends `BaseAction`.

8.3 PlatformAction

Extends `BaseAction`.

CHAPTER

NINE

POLICY

class policyengine.models.**Policy**(**args*, ***kwargs*)

kind
Kind of policy (platform, constitution, or trigger).

filter
The filter code of the policy.

initialize
The initialize code of the policy.

notify
The notify code of the policy.

success
The pass code of the policy.

fail
The fail code of the policy.

community
The community which the policy belongs to.

action_types
The action types that this policy applies to.

name
The name of the policy.

description
The description of the policy. May be empty.

is_active
True if the policy is active. Default is True.

modified_at
Datetime object representing the last time the policy was modified.

bundled_policies
Policies bundled inside this policy.

property is_bundled
True if the policy is part of a bundle

CHAPTER

TEN

PROPOSAL

class policyengine.models.**Proposal**(**args*, ***kwargs*)

The Proposal model represents the evaluation of a particular policy for a particular action. All data relevant to the evaluation, such as vote counts, is stored in this model.

proposal_time
Datetime object representing when the proposal was created.

status
Status of the proposal. One of PROPOSED, PASSED or FAILED.

policy
The policy that is being evaluated.

action
The action that triggered the proposal.

data
Datastore for persisting any additional data related to the proposal.

community_post
Identifier of the post that is being voted on, if any.

governance_process
The Metagov GovernanceProcess that is being used to make a decision about this Proposal, if any.

property is_vote_closed
Returns True if the vote is closed, False if the vote is still open.

get_time_elapsed()
Returns a datetime object representing the time elapsed since the first proposal.

get_all_boolean_votes(*users=None*)
For Boolean voting. Returns all boolean votes as a QuerySet. Can specify a subset of users to count votes of. If no subset is specified, then votes from all users will be counted.

get_yes_votes(*users=None*)
For Boolean voting. Returns the yes votes as a QuerySet. Can specify a subset of users to count votes of. If no subset is specified, then votes from all users will be counted.

get_no_votes(*users=None*)
For Boolean voting. Returns the no votes as a QuerySet. Can specify a subset of users to count votes of. If no subset is specified, then votes from all users will be counted.

get_all_number_votes(*users=None*)
For Number voting. Returns all number votes as a QuerySet. Can specify a subset of users to count votes of. If no subset is specified, then votes from all users will be counted.

get_one_number_votes(*value*, *users=None*)

For Number voting. Returns number votes for the specified value as a QuerySet. Can specify a subset of users to count votes of. If no subset is specified, then votes from all users will be counted.

CHAPTER

ELEVEN

DOCUMENTS

11.1 CommunityDoc

class policyengine.models.**CommunityDoc**(**args*, ***kwargs*)

name
The name of the document.

text
The text within the document.

community
The community which the document belongs to.

is_active
True if the document is active. Default is True.

CHAPTER

TWELVE

USER

12.1 CommunityUser

Extends `User`.

class policyengine.models.**CommunityUser**(**args*, ***kwargs*)

readable_name
The readable name of the user. May or may not exist.

community
The community which the user belongs to.

access_token
The access token which the user uses on login. May or may not exist.

is_community_admin
True if the user is an admin. Default is False.

avatar
The URL of the avatar image of the user. May or may not exist.

get_roles()
Returns a list of CommunityRoles containing all of the user's roles.

has_role(*name*)
Returns True if the user has a role with the specified role_name.

Parameters **name** – The name of the role to check for.

property constitution_community
The ConstitutionCommunity that this user belongs to.

CHAPTER

THIRTEEN

ROLE

13.1 CommunityRole

Extends `Group`.

class policyengine.models.**CommunityRole**(**args*, ***kwargs*)

community
The community which the role belongs to.

role_name
The readable name of the role.

description
The readable description of the role. May be empty.

is_base_role
Whether this is the default role in this community.

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

CHAPTER

FOURTEEN

VOTE

14.1 UserVote

class policyengine.models.**UserVote**(**args*, ***kwargs*)

user
The user who cast the vote.

proposal
The policy proposal that initiated the vote.

vote_time
Datetime object representing when the vote was cast.

get_time_elapsed()
Returns a datetime object representing the time elapsed since the vote was cast.

14.2 BooleanVote

Extends `UserVote`.

class policyengine.models.**BooleanVote**(**args*, ***kwargs*)

boolean_value
The value of the vote. Either True ('Yes') or False ('No').

14.3 NumberVote

Extends `UserVote`.

class policyengine.models.**NumberVote**(**args*, ***kwargs*)

number_value
The value of the vote. Must be an integer.

CHAPTER

FIFTEEN

DATASTORE

class policyengine.models.**DataStore**(**args*, ***kwargs*)

DataStore used for persisting serializable data on a Proposal.

get(*key*)

Returns the value associated with the given key.

Parameters **key** – The key associated with the value.

set(*key*, *value*)

Stores the given value, referenced by the given key.

Parameters

- **key** – The key to associate with the given value.
- **value** – The value to store.

remove(*key*)

Removes the value associated with the given key. Returns True if a value was found and removed. Returns False otherwise.

Parameters **key** – The key associated with the value to be removed.

PYTHON MODULE INDEX

p

INDEX

H

I

K

L

M

N

O

P

R

S

T

U

V