Kustosz

Mirek Długosz

Sep 16, 2022

CONTENTS:

1 Containers 1

2 Heroku 7

3 Manual installation 11

4 VPS (Raspberry Pi, DigitalOcean Droplet) 19

5 Trying it out 23

6 Production deployments 25

7 Quick overview for software developers 27

8 Initial setup 29

9 Basic usage 31

10 Searching 37

11 Filters 41

12 Configuration 43

13 Development 55

14 Kustosz documentation 63

CHAPTER

ONE

CONTAINERS

There is one image, available at quay.io/kustosz/app. Image is OCI/runc-compatible, meaning it works with Docker and Podman.

1.1 Single container

Kustosz requires background processes, and running multiple processes in single container is generally discouraged. However, it has the benefit of being extremely easy to do.

If you decide to run your main instance in single container, then you should at least use volume or bind mount to store `/opt/kustosz/web/db/` directory content on local host. This way Kustosz database will survive container restart.

Container image will bind Kustosz to port 8000 inside the container. When starting it, you need to bind it to some port on the host machine. Otherwise, Kustosz will not be available.

docker

```
docker run -p 127.0.0.1:8000:8000 -v kustosz_db:/opt/kustosz/web/db/ quay.io/kustosz/app
```

podman

```
podman run -p 127.0.0.1:8000:8000 -v kustosz_db:/opt/kustosz/web/db/ quay.io/kustosz/app
```

Kustosz web UI will be available at localhost:8000/ui/.

By default, container will create new user `admin` with randomly generated password. Password will be printed to container log after line saying "You can log in with following credentials". If you want to specify your own password, see *Container-specific configuration options* below.

If you have OPML file that you want to import, specify `KUSTOSZ_IMPORT_CHANNELS_DIR` variable when starting a container. Alternatively, you can use command line tool described in *initial setup*. You can use `docker exec` to execute a command in a context of running container.

1.2 docker-compose

docker-compose is default container orchestrating tool for Docker. It allows you to manage multiple containers as a single unit.

First, download docker-compose.yaml file. Then review it and modify to your liking. You should at least change Postgres password (make sure that you changed all references to old password in the file). You probably also want to specify `KUSTOSZ_USERNAME` and `KUSTOSZ_PASSWORD` variables in `kustosz_api` container. Finally, run it:

```
docker-compose -p kustosz -f ./docker-compose.yaml up
```

Kustosz web UI will be available at localhost/ui/.

In default configuration, docker-compose will listen to port 80 on the host machine. If this port is already taken - which is very likely if you have multiple web-based services running - change definition of `ports` in `kustosz_api` container.

docker-compose will start containers for Postgres (database) and Redis (Celery transport). They are commonly used by modern applications and it's possible that you already have them running, or have different images for them. In that case, feel free to change image references or remove them from your `docker-compose.yaml` file. Make sure that applications running in containers can access these external services.

You can switch Redis to any message broker supported by Celery. RabbitMQ is popular and well-tested option, but it requires more resources.

When using different database or Celery transport, make sure that you also change environment variables telling Kustosz how to connect with these services. These variables start with `DYNACONF_DATABASES__` and `DYNACONF_CELERY_`. Make sure that you change their values in all containers running from `kustosz` image.

1.3 podman pods

podman pods allow multiple containers to share single namespace. This way they can easily access each other, and you can manage them as a single unit.

Pods has seen very active development in Podman 2.x line. We recommend that you use at least Podman 3.0, released in February 2021.

First, download kustosz_pod.yaml file. Then review it and modify to your liking. You should at least change Postgres password (make sure that you changed all references to old password in the file). You probably also want to specify `KUSTOSZ_USERNAME` and `KUSTOSZ_PASSWORD` variables in `kustosz_api` container. Finally, run it:

```
podman play kube ./kustosz_pod.yaml
```

Kustosz web UI will be available at localhost/ui/.

In default configuration, podman will listen to port 80 on the host machine. Usually user processes can't bind to this port. If you want to run rootless podman, you need to change the port in pod definition or change the configuration of host machine. If port 80 is already taken - which is very likely if you have multiple web-based services running - change definition of `ports` in `kustosz_api` container.

podman will start containers for Postgres (database) and Redis (Celery transport). They are commonly used by modern applications and it's possible that you already have them running, or have different images for them. In that case, feel free to change image references or remove them from your `kustosz_pod.yaml` file. Make sure that applications running in containers can access these external services.

You can switch Redis to any message broker supported by Celery. RabbitMQ is popular and well-tested option, but it requires more resources.

When using different database or Celery transport, make sure that you also change environment variables telling Kustosz how to connect with these services. These variables start with `DYNACONF_DATABASES__` and `DYNACONF_CELERY_`. Make sure that you change their values in all containers running from `kustosz` image.

1.4 Changing Kustosz configuration

All Kustosz configuration options are documented on *backend configuration page*.

The recommended way of changing configuration of Kustosz running in container is by environment variables. Remember that environment variables must be prefixed by `DYNACONF_` - e.g. to change value of setting `KUSTOSZ_READING_SPEED_WPM`, your environment variable should be named `DYNACONF_KUSTOSZ_READING_SPEED_WPM`.

You can set variable using `-e` / `--env` flag. If you want to set multiple variables, consider storing them in file and using `--env-file` flag.

docker

```
docker run -e KUSTOSZ_SKIP_MIGRATE=1 quay.io/kustosz/app
```

```
echo 'KUSTOSZ_SKIP_MIGRATE=1' > production.env
docker run --env-file production.env quay.io/kustosz/app
```

podman

```
podman run -e KUSTOSZ_SKIP_MIGRATE=1 quay.io/kustosz/app
```

```
echo 'KUSTOSZ_SKIP_MIGRATE=1' > production.env
podman run --env-file production.env quay.io/kustosz/app
```

Alternatively, you can put `settings.yaml` file in `/opt/kustosz/web/settings` directory inside the container. It is recommended that you store `settings.yaml` file on your host machine and use volume or bind mount to make it accessible inside the container. As this is the only directory that Kustosz will read while searching for setting files, it's best to start with copy of `settings.yaml` and `settings.local.yaml` files from Kustosz backend source code repository.

1.5 Container-specific configuration options

There are few environmental variables recognized by container image entry point script, listed below.

None of them is set by default.

Most variables act as flags - script only checks if they have been set, ignoring their value. In these cases, setting variable to any non-empty value will cause effect described below variable name. Note that this includes values often considered "falsy", such as number 0, string "false", or string "off". Set variable if you want effect described below, omit it completely to retain default behavior.

See *Changing Kustosz configuration* section above for quick overview of passing environment variables to containers.

1.5.1 KUSTOSZ_USERNAME

Username that you will use to log in to Kustosz in your browser. If user doesn't exist, it will be created automatically. Password will be set to value of `KUSTOSZ_PASSWORD` or generated randomly.

If omitted, default value of `admin` will be assumed. Both username and password will be printed in container log, if user was created.

When running multiple containers through docker-compose or podman pods, this variable should be set **only** in `kustosz_api` container.

1.5.2 KUSTOSZ_PASSWORD

Password that you will use to log in to Kustosz in your browser.

If omitted, random password will be generated. Both username and password will be printed in container log, if user was created.

When running multiple containers through docker-compose or podman pods, this variable should be set **only** in `kustosz_api` container.

1.5.3 KUSTOSZ_SKIP_CREATE_USER

You need user account to access the application. By default, container entry script will create user for you, using `KUSTOSZ_USERNAME` and `KUSTOSZ_PASSWORD` variables when available. You can skip automatic user creation by setting this variable.

Note that user is created only if it doesn't already exist.

1.5.4 KUSTOSZ_IMPORT_CHANNELS_DIR

Most feed readers have an option to export list of subscribed channels into OPML file. If you have such file, you can import it in Kustosz when starting container with help of this variable.

This variable should point to directory where OPML files are. All files in that directory are assumed to be OPML files. File extension doesn't matter.

Importing feeds is generally done only once, but Kustosz ignores feeds that were added previously, so it's safe to always specify this variable. However, this will add some time to container startup.

When specifying this variable, you most likely also want to mount host directory with OPML file into container.

When running multiple containers through docker-compose or podman pods, this variable should be set **only** in `kustosz_api` container.

1.5.5 KUSTOSZ_SKIP_MIGRATE

Migrations change database structure. Kustosz will automatically detect migrations that were run previously on connected database to ensure they are not run again. In other words, it's safe to run migrations multiple times on the same database. However, running them will add some time to container startup.

If you decide to skip migrations in day-to-day operations, there are still two situations when you are required to run them:

- when starting application with new database for the first time, so database can be populated with initial tables structure

- after upgrading Kustosz to newer version, when specific version is run for the first time.

1.5.6 KUSTOSZ_SKIP_CREATECACHETABLE

By default, Kustosz will store cache in main database. In order to do that, additional table must be created. Kustosz will automatically detect when required table is available, so it's safe to call this function multiple times. However, running it will add some time to container startup.

This option is only required when starting application with new database for the first time.

Kustosz supports all cache backends supported by Django, except local-memory. If you decide to use memcached as cache backend, there's no need to create cache table, and setting this variable is safe.

1.5.7 KUSTOSZ_SKIP_COLLECTSTATIC

Django, which Kustosz is built upon, requires us to run `collectstatic` command during deployment. It will populate special local directory (`/opt/kustosz/web/static/`) with copy of all static files. That copy is considered disposable and can be re-populated every time application is started. However, this re-population will add some time to container startup.

You can skip running `collectstatic` if you decide to store `/opt/kustosz/web/static/` outside of container, so this directory content may survive container restart. When you do that, there are still two situations when you are required to run `collectstatic`:

- when starting application with empty `/opt/kustosz/web/static/` directory for the first time
- after upgrading Kustosz to newer version, when specific version is run for the first time.

CHAPTER

TWO

HEROKU

Deploying on Heroku is as easy as clicking the button below:

Fill in the form, click “Deploy app” button, wait about a minute and click “View” button. Use username and password provided in Heroku deployment form to log in.

2.1 Deploying manually

If you want to modify your Kustosz instance before deployment, or don’t like clicking buttons, you can deploy it manually using Heroku CLI.

First, clone kustosz-heroku repository:

```
git clone https://github.com/KustoszApp/kustosz-heroku
```

Perform any modifications to repository files and commit them. Don’t push your changes.

If you would like to import OPML file during deployment, create `kustosz/opml` directory and put XML file there.

Create new Heroku app. You can pass `--region=<region_id>` to deploy to specific region. Use `heroku regions` for list of available regions.

```
APP_NAME="my-kustosz-instance-name"
heroku create --addons=heroku-postgresql:hobby-dev "$APP_NAME"
```

Set required environment variables:

```
heroku config:set ENV_FOR_DYNACONF=production
heroku config:set DYNACONF_ALLOWED_HOSTS="[\"$APP_NAME.herokuapp.com\"]"
```

Set username and password. This step is optional and if you don’t set these variables, random password will be generated automatically during initial deployment. It will be printed to deployment log, but this is the only time you will see it.

```
heroku config:set KUSTOSZ_USERNAME=<username of your choice>
heroku config:set KUSTOSZ_PASSWORD=<password of your choice>
```

Add buildpacks explicitly:

```
heroku buildpacks:add heroku/nodejs
heroku buildpacks:add heroku/python
```

Push current branch to Heroku. This will build new version and deploy it:

```
git push heroku main
```

Finally, set new Django SECRET_KEY. This can't be done automatically during deployment, because resetting this variable will log out all users.

```
heroku config:set DYNACONF_SECRET_KEY="$(heroku run kustosz-manager generate_secret_key)"
```

That's it! Your Kustosz is now available under $APP_NAME.herokuapp.com/ui/

2.2 Importing OPML file

Most feed readers have an option to export list of subscribed channels into OPML file. If you have such file, you can import it during Kustosz deployment. Just create `kustosz/opml` directory and put XML file there:

```
mkdir kustosz/opml
cp <path/to/file.xml> kustosz/opml/
git add kustosz/opml/*.xml
git commit -m 'Adding OPML file'
```

Importing OPML file can take some time, depending on number of channels and amount of content they have published. There's no reason to import the same file during every subsequent deployment (e.g. when you update Kustosz to newest version), so once file has been imported, you can tell deployment script to skip this step:

```
heroku config:set KUSTOSZ_SKIP_IMPORT_CHANNELS=1
```

2.3 Considerations when running Kustosz on paid plans

kustosz-heroku repository is designed to use **single dyno** on **free plan**. Kustosz deployed from this repository won't be able to scale horizontally.

The first limit you are likely to encounter is Postgres row limit. Free tier provides only 10 000 rows, which can be quickly exhausted with just 100 feeds. Hobby Basic tier costs 9 USD per month and offers 10 000 000 row limit, which is virtually unlimited as far as Kustosz is concerned.

You also need Redis to allow multiple dynos to coordinate their work. Free Redis tier will not allow enough connections.

```
heroku addons:create heroku-redis:premium-1
```

kustosz-heroku repository comes with sample Procfile for scalable deployments, `Procfile.scalable`. You should use it instead of default `Procfile`:

```
mv Procfile.scalable Procfile
git add Procfile
git commit -m 'Use scalable Procfile'
```

Once application is deployed, you need to start dynos for additional process types:

```
heroku ps:scale worker=1 clock=1 feedfetcher=1
```

You should **never** have more than single `clock` and `feedfetcher` processes - the first one is lightweight process responsible for starting recurring background tasks, and second is responsible for processing sequential queues. However, you are free to start multiple `worker` and `web` processes:

```
heroku ps:scale web=2 worker=6
```

2.4 Heroku-specific configuration options

There are few environmental variables recognized by Heroku deployment script, listed below.

None of them is set by default.

Most variables act as flags - script only checks if they have been set, ignoring their value. In these cases, setting variable to any non-empty value will cause effect described below variable name. Note that this includes values often considered "falsy", such as number 0, string "false", or string "off". Set variable if you want effect described below, omit it completely to retain default behavior.

You can set these variables in your application settings in Heroku web dashboard or with Heroku CLI:

```
heroku config:set VARIABLE_NAME=<value>
```

2.4.1 KUSTOSZ_USERNAME

Username that you will use to log in to Kustosz in your browser. If user doesn't exist, it will be created automatically. Password will be set to value of `KUSTOSZ_PASSWORD`, which must also be set.

If omitted, default value of `admin` will be assumed, unless `KUSTOSZ_SKIP_PASSWORD_GENERATION` is also set.

2.4.2 KUSTOSZ_PASSWORD

Password that you will use to log in to Kustosz in your browser.

If omitted, random password will be generated, unless `KUSTOSZ_SKIP_PASSWORD_GENERATION` is also set. Password value will be displayed in container log.

2.4.3 KUSTOSZ_SKIP_PASSWORD_GENERATION

By default, Heroku deployment script will generate random password for `admin` user and print it in log. This password is only used if `admin` user doesn't already exist. Since this is very fast operation, it is generally safe to run it every time you deploy your application.

If `KUSTOSZ_USERNAME` and `KUSTOSZ_PASSWORD` are set, they will be used instead and password will not be generated or displayed in log.

If you set this variable and **don't** set `KUSTOSZ_USERNAME` and `KUSTOSZ_PASSWORD`, no user will be created. If this is your first time deploying Kustosz, you won't be able to log in until you create new user manually.

2.4.4 KUSTOSZ_SKIP_IMPORT_CHANNELS

If you have added OPML file (see *Importing OPML file*) and deployed your application, there's no reason to import the same file during each subsequent deployment (e.g. when you update Kustosz to newest version). You can tell deployment script to skip this step by setting this variable.

2.4.5 KUSTOSZ_SKIP_MIGRATE

Migrations change database structure. Kustosz will automatically detect migrations that were run previously on connected database to ensure they are not run again. In other words, it's safe to run migrations multiple times on the same database. However, running them will make deployment a little slower.

If you decide to skip migrations in day-to-day operations, there are still two situations when you are required to run them:

- when starting application with new database for the first time, so database can be populated with initial tables structure
- after upgrading Kustosz to newer version, when specific version is run for the first time.

2.4.6 KUSTOSZ_SKIP_CREATECACHETABLE

By default, Kustosz will store cache in main database. In order to do that, additional table must be created. Kustosz will automatically detect when required table is available, so it's safe to call this function multiple times. However, running them will make deployment a little slower.

This option is only required when deploying Kustosz for the first time.

Kustosz supports all cache backends supported by Django, except local-memory. If you decide to use memcached as cache backend, there's no need to create cache table, and setting this variable is safe.

CHAPTER

THREE

MANUAL INSTALLATION

User attention is required

Kustosz is flexible and supports deployments to various environments. This guide limits a number of choices covered to lower cognitive load and provide instructions that are easier to follow. However, it still **expects you to understand what is happening and adjust specific steps to your particular environment**. It is recommended that you familiarize yourself with entire document before taking any action, including *Additional setup instructions* section.

This page covers manual installation of Kustosz. It's intended for users who require maximum flexibility or want deeper understanding of Kustosz deployments. Majority of users should be able to deploy Kustosz using *automatic installer*.

3.1 Prerequisites

Kustosz is web application. You probably want it available under a dedicated (sub)domain that supports secure HTTPS connection. Configuration of domain, HTTPS and access control is outside of scope of this guide.

There are few dependencies that you absolutely must meet:

- Python 3.9 or newer
- Ability to execute arbitrary commands in the context of server
- Ability to start your own long-running processes (needed by Celery worker)
- At least 256 MiB of memory

While not required, we also recommend:

- PostgreSQL database server
- Redis as Celery transport

3.2 Prepare Kustosz server environment

Create Kustosz base directory. This is where you will put settings, and where Kustosz will store SQLite database file and certain cache files.

```
mkdir ~/kustosz_home
```

Create environment variables required by Kustosz. You need to ensure they are set for all processes that involve Kustosz, e.g. Celery or cron job. You might want to add them to `~/.bash_profile`.

```
export ENV_FOR_DYNACONF="production"
export DJANGO_SETTINGS_MODULE=kustosz.settings
export KUSTOSZ_BASE_DIR="$HOME/kustosz_home"
```

At this stage, do all the other setup that your Kustosz installation might require - create (sub)domain, create Postgres user and database, install Redis server and ensure it starts automatically etc. The exact actions to take depend on your operating system and preferred configuration.

3.3 Install Kustosz server

The very first step is creating and activating virtual environment. It must exists as long as you run Kustosz, so you must create it in a path that won't disappear automatically.

You might need to re-create virtual environment once your system Python is upgraded.

It's fine to remove virtual environment and create new one when you upgrade Kustosz to newer version.

```
python3 -m venv ~/.virtualenvs/kustosz/
source ~/.virtualenvs/kustosz/bin/activate
```

Activating virtual environment with environment variables

Instead of sourcing activation script, you may also set two environment variables:

```
export VIRTUAL_ENV="$HOME/.virtualenvs/kustosz"
export PATH="$VIRTUAL_ENV/bin:$PATH"
```

Once virtual environment is created and activated, you can install Kustosz package:

```
pip install --upgrade pip wheel
pip install kustosz
```

3.4 (Optional) Install additional Python packages

Depending on your preferred configuration, you might need to install additional Python modules, such as:

- `redis` is required to use Redis for cache or as Celery transport
- `psycopg2` is required to connect to PostgreSQL database (psycopg2 requires additional system packages to build)
- `pylibmc` or `pymemcache` is required to use Memcached for cache
- `whitenoise` may be used to *serve static files directly from WSGI server*

```
pip install redis psycopg2
```

3.5 (Optional) Install kustosz-node-readability

While syndication formats exists so users can read content in their reader applications, many authors try to force readers to visit the website. Kustosz can automatically extract full article content from source page. This extraction is done by "readability" algorithm.

If you have Node.js on your server, you can install kustosz-node-readability. It uses Readability.js, algorithm implementation used for Firefox Reader View and by Pocket. It's probably the best maintained readability implementation around.

```
npm install -g kustosz-node-readability
```

3.6 Configure Kustosz

The main configuration file is `$KUSTOSZ_BASE_DIR/settings/settings.yaml`. You can download sample file from git repository.

```
mkdir -p $KUSTOSZ_BASE_DIR/settings/
curl https://raw.githubusercontent.com/KustoszApp/server/main/settings/settings.yaml -o
↪$KUSTOSZ_BASE_DIR/settings/settings.yaml
```

You can modify sample file, or put site-specific modifications in `$KUSTOSZ_BASE_DIR/settings/settings.local.yaml`. Second option is preferred, as it allows you to overwrite `settings.yaml` file during upgrade, while keeping site-specific settings safely separated.

The exact changes to make are highly dependant on your preferred configuration. Make sure to also read *backend configuration page*.

Below is non-exhaustive list of settings you might want to change. Consult *Kustosz*, Django and Celery settings documentation for meaning of specific options.

- `SECRET_KEY` should be set to new value, obtained by running `kustosz-manager generate_secret_key`
- `ALLOWED_HOSTS` should contain fully qualified domain name where Kustosz is running
- `DATABASES`, especially if you want to use PostgreSQL
- `CACHES`, if you want to use Memcached or Redis for cache
- `CORS_ALLOWED_ORIGINS`, if you decide to run frontend and backend on separate domains, this should contain origin of frontend page
- `CELERY_BROKER_URL`, if you decide to use Redis as Celery broker
- `KUSTOSZ_READABILITY_NODE_ENABLED`, if you have installed kustosz-node-readability
- `STATIC_ROOT`, if you decide to serve static files using gunicorn; this is path to directory where files will be copied to, it should be inside `$KUSTOSZ_BASE_DIR`
- `STATICFILES_DIRS`, if you decide to serve static files using gunicorn; this is list of paths of static files to copy, it should contain directory where you *extract frontend files*

Example configuration

Settings below assume that you serve Kustosz from `kustosz.example.com`, use PostgreSQL as database and Redis as Celery transport.

```
production:
  DATABASES:
    default:
      ENGINE: 'django.db.backends.postgresql_psycopg2'
      NAME: "kustoszdb"
      USER: "postgres"
      PASSWORD: "postgres_user_password"
      HOST: "localhost"
      PORT: "5432"
  CELERY_BROKER_URL: "redis://localhost:6379/"
  ALLOWED_HOSTS:
    - 'kustosz.example.com'
  STATIC_ROOT: '/home/USER/kustosz_home/frontend_assets'
  STATICFILES_DIRS:
    - '/home/USER/kustosz_frontend'
```

3.7 Run migrations

Migrations change database structure. You have to run migrations before you first start Kustosz, and each time after upgrading to new version.

```
kustosz-manager migrate
```

3.8 Create cache tables

In default configuration, Kustosz uses main database for cache. Database must first be prepared for that. You need to run this only once, before you first start Kustosz. You don't have to run it if you use Memcached or Redis for cache.

```
kustosz-manager createcachetable
```

3.9 Start main Celery process

Kustosz requires main Celery process to be running. It is used to handle all background tasks, such as downloading files or running filters.

Command to start Celery process is:

```
celery -A kustosz worker -l INFO -Q feed_fetcher,celery
```

Due to way Kustosz is designed, some background tasks should run serially. The best way of ensuring that is by running separate Celery process with single worker for `feed_fetcher` queue. The downside is that it requires more system resources.

```
celery -A kustosz worker -l INFO -Q celery
celery -A kustosz worker -l INFO -Q feed_fetcher --concurrency 1
```

It is recommended that you use process supervisor to run commands above. Most Linux systems come with systemd, which may be used for that purpose - see *Use systemd to ensure background processes are running*. Another option is Supervisor - see *Use supervisor to ensure background processes are running*.

3.10 Run gunicorn

It's finally time to start main Kustosz server. We install and run gunicorn, but you can use any WSGI-compatible web server. uWSGI is another popular option. If you use Apache web server, you may want to use mod_wsgi.

```
pip install gunicorn
gunicorn kustosz.wsgi --bind 0.0.0.0:8000
```

3.11 Download frontend files

Kustosz server provides REST API only. To make use of it, you also need a client.

Create new directory on your server. Open Kustosz web UI releases page in your browser and find `kustosz.tar.xz` file provided by the newest release. Download that file into created directory and unpack it. You can remove archive afterwards.

```
mkdir ~/kustosz_frontend
cd ~/kustosz_frontend
curl -L https://github.com/KustoszApp/web-ui/releases/download/VERSION_NUMBER/kustosz.
↪tar.xz -o kustosz.tar.xz  # change VERSION_NUMBER to latest version number
tar xf kustosz.tar.xz
rm kustosz.tar.xz
```

3.12 Set up HTTP proxy in front of gunicorn

Most of dedicated WSGI web servers consider serving static files as out of their scope. Usually it is recommended that you set up HTTP proxy in front of WSGI server. NGINX is popular web server that is commonly used as such proxy. Most Linux distributions have NGINX package that you can install.

After installing NGINX, create `/etc/nginx/sites-available/kustosz` file with following content:

Listing 1: /etc/nginx/sites-available/kustosz

```
upstream kustosz {
    # port number here must be the same as port number in gunicorn --bind command
    server localhost:8000;
}

server {

    listen 80;
    # domain name that Kustosz will be available at
    server_name localhost;

    location / {
```

(continues on next page)

(continued from previous page)

```
        proxy_pass http://kustosz;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect off;
        client_max_body_size 10M;
    }

    location /ui/ {
        # this must be absolute path to directory where you extracted frontend files
        alias /home/USER/kustosz_front/;
    }

}
```

Make configuration available to NGINX server and restart it so changes can be applied:

```
ln -s /etc/nginx/sites-available/kustosz /etc/nginx/sites-enabled/
systemctl restart nginx
```

NGINX file permissions

NGINX often runs as an user with limited permissions, such as `www`, `www-data`, `nginx` or `nobody`. You need to ensure that this user has permissions to read extracted frontend files. It also needs executable bit on all directories in the path.

Congratulations! You should see Kustosz running at http://your-domain/ui/.

3.13 Additional setup instructions

Following section documents configuration changes that do not apply to all deployments, but are still common. Some of these instructions extend basic installation described above, while some override parts of it.

3.13.1 Set up periodic channels update with Celery

As noted in *automatic channels update* section, Kustosz requires channels update process to run periodically. The preferred way of ensuring this process is run is by using Celery beat. Celery beat must run in the background all the time in addition to main Celery process.

```
celery -A kustosz beat -l INFO
```

3.13.2 Set up periodic channel update with cron

As noted in *automatic channels update* section, Kustosz requires channels update process to run periodically. If you can't run Celery beat, you may use system scheduler, such as cron.

The command that you want to run is:

```
kustosz-manager fetch_new_content --wait
```

To run it with cron, use `crontab -e` command and add the following line:

```
*/5 * * * *  kustosz-manager fetch_new_content --wait
```

cron jobs usually run in special environment that differs from normal shell. Remember that Kustosz command line tools require *few environment variables* and *active virtual environment*. You might want to create simple wrapper script that sets up these variables before running the command. Example of such wrapper is available in source code repository in `etc/cron/kustosz-fetch-content.sh`.

3.13.3 Use systemd to ensure background processes are running

Kustosz requires some background processes to run all the time - main Celery process, Celery beat process (optionally), gunicorn WSGI server (optionally). One way to ensure they are running is systemd. Unlike supervisor, systemd can start processes automatically during system boot.

Systemd is default init in most of Linux distributions.

Sample systemd unit file is available in source code repository in `/etc/systemd/system/kustosz@.service`. This is unit template that calls dispatcher script to start the actual process. Dispatcher script is responsible for setting *required environment variables* and *activating virtual environment*. Sample dispatcher script is provided in `/etc/systemd/bin/kustosz-service-dispatcher`.

Put dispatcher script anywhere on your file system and unit file in `/etc/systemd/system/`. Start and enable all background processes:

```
systemctl enable --now kustosz@worker.service
systemctl enable --now kustosz@feedfetcher.service
systemctl enable --now kustosz@clock.service
systemctl enable --now kustosz@web.service
```

3.13.4 Use supervisor to ensure background processes are running

Kustosz requires some background processes to run all the time - main Celery process, Celery beat process (optionally), gunicorn WSGI server (optionally). One way to ensure they are running is supervisor.

You can install supervisor in your virtual environment with `pip install supervisor`.

Sample supervisor config is available in source code repository in `etc/supervisor/supervisord.conf`. Feel free to adjust it to your liking and to better fit your environment. Sample config file requires `run` and `logs` directories to exist in the same directory as supervisord.conf file and does not start gunicorn process automatically.

Finally, start supervisor:

```
supervisord -c path/to/supervisord.conf
```

supervisor can ensure that certain processes are running, but it has one drawback - you need to start it manually. supervisor documentation has section on starting supervisord on system startup. Some Linux distributions, like Debian, offer supervisor package with init daemon script that can be used to start it automatically. If you go down this path, remember that Kustosz command line tools require *few environment variables* and *active virtual environment*. You might also need to ensure correct file permissions, as supervisor will be running as privileged user, while Kustosz was installed for standard user.

3.13.5 Use gunicorn to serve static files

WhiteNoise is project dedicated to making WSGI-compatible web servers better at serving static files. Serving static content is main reason this guide uses NGINX, so with WhiteNoise you might not need NGINX anymore. The second reason is security, so you probably still want *some* kind of HTTP proxy in front of WhiteNoise-enabled WSGI server.

First, install WhiteNoise:

```
pip install whitenoise
```

Configure Kustosz:

- `STATICFILES_DIRS` should contain directory where you *extracted frontend files*; this setting is list of paths
- `STATIC_ROOT` should be a path to directory where static files will be copied to; it should be inside `$KUSTOSZ_BASE_DIR`
- `STATICFILES_STORAGE` should be set to `"whitenoise.storage.CompressedManifestStaticFilesStorage"`
- `MIDDLEWARE` should contain `"whitenoise.middleware.WhiteNoiseMiddleware"` as third entry on the list, **below** `"django.middleware.security.SecurityMiddleware"` and `"corsheaders.middleware.CorsMiddleware"`; see container `settings.local.yaml` in source code repository
- `STATIC_URL` must be set

Finally, run `collectstatic` command. This command will walk through all directories specified in `STATICFILES_DIRS` and copy files into `STATIC_ROOT`, where they will be processed (i.e. compressed).

```
kustosz-manager collectstatic
```

You have to run above command before you first start Kustosz, and each time after upgrading to new version.

CHAPTER

FOUR

VPS (RASPBERRY PI, DIGITALOCEAN DROPLET)

Automatic installer is designed to enable fully-operational Kustosz instance on machine where you have **superuser** (root) permissions. Read *Impact of the installer on the system* section below for overview of changes that installer does to the system.

If you require greater flexibility than installer can provide, we have separate page discussing *manual installation*.

Installer installs latest released version of Kustosz. You can re-run the installer to upgrade existing Kustosz instance to newer version.

4.1 Prerequisites

Kustosz is web application. You probably want to access it from any network using dedicated (sub)domain. Configuration of domain and access control is outside of scope of this guide.

Kustosz needs at least 256 MiB of memory.

Installer must be run on Linux machine and it needs to connect to the machine where Kustosz will be installed. You can run the installer on the machine where Kustosz will be running.

You need superuser (root) access to machine where Kustosz is installed. You can configure installer to skip steps that require elevated permissions.

4.2 Preparing installation environment

Installer is implemented as Ansible collection with a playbook. You need Ansible 2.9 or newer to run it. Ansible must be installed on Linux, and you can install it on the machine where Kustosz will be running.

Many Linux distributions provide Ansible package. We recommend that you install Ansible in a disposable virtual environment:

```
python3 -m venv /tmp/ansible-venv/
. /tmp/ansible-venv/bin/activate
pip install --upgrade pip
pip install ansible
```

Now you can install the installer itself:

```
ansible-galaxy collection install kustosz.install
```

You also need an inventory file. It tells Ansible where your host is and allows you to configure the installer (see *Installer configuration* page).

Listing 1: inventory.yml

```yaml
all:
  children:
    kustosz:
      hosts:
        kustosz_server_ip_or_domain_name OR localhost
      vars:
        ansible_port: 22
        ansible_user: 'name_of_user_that_can_ssh_to_host'
        ansible_ssh_private_key_file: '/path/to/file/id_rsa'
        ansible_connection: local  # only if hosts is localhost

        web_user_name: kustosz
        web_user_password: kustosz
        settings_local_path: /tmp/kustosz/settings/settings.local.yml
        opml_local_path: /tmp/kustosz/opmls/
        kustosz_nginx_server_name: "kustosz.example.com"
        run_certbot: true
        certbot_extra_args: >
          --agree-tos
          --email somename@example.com
```

`vars` section is optional, but recommended. Apart from Ansible connection variables, you should consider setting following variables:

- `web_user_name` - user name that you will use to log in to Kustosz
- `web_user_password` - password that you will use to log in to Kustosz
- `settings_local_path` - **local** path to `settings.local.yaml` file (see *backend configuration* page)
- `opml_local_path` - **local** path to directory with OPML files that should be imported during installation
- `kustosz_nginx_server_name` - domain name of your Kustosz instance (technically, value of `server_name` directive in NGINX configuration file)
- `run_certbot` - whether `certbot` should be called to configure Let's Encrypt TLS certificate
- `certbot_extra_args` - additional `certbot` command-line options

Full list of variables recognized by the installer is available on *installer configuration* page.

4.3 Running the installer

```
ansible-playbook kustosz.install.playbook -i ./inventory.yml
```

When installer finishes, open KUSTOSZ_SERVER_NAME/ui/ in your web browser to access Kustosz.

If installer fails for any reason (e.g. due to intermittent problem with access to external resources), you can run above command again. Installer will automatically pick up where it left.

Running above command on server with Kustosz already installed is also safe, as installer will not make any changes that are not required. You can use the same command to upgrade Kustosz to the newest version.

4.4 Impact of the installer on the system

Automatic installer is primarily designed with new systems in mind. The idea is that you spin up virtual machine in the cloud, assign it domain, run installer and then you have fully operational Kustosz instance. To achieve this goal, installer needs to change global configuration of the system.

Specifically, installer will:

- install Redis, NGINX and some packages needed to compile Python
- create new system user
- create NGINX virtual server configuration file, systemd unit file, and systemd unit Exec script
- enable systemd services for background tasks.

You can skip each of these steps, but Kustosz might not work correctly until you perform their equivalent on your system.

Everything else is done within the scope of separate system user and will not impact other services running on the system.

CHAPTER

FIVE

TRYING IT OUT

There are many ways to install Kustosz, depending on your environment and hosting provider. If you just want to try it out and see if Kustosz is for you, the easiest way is by using containers. Just run:

docker

```
docker run -p 127.0.0.1:8000:8000 quay.io/kustosz/app
```

podman

```
podman run -p 127.0.0.1:8000:8000 quay.io/kustosz/app
```

Open localhost:8000/ui/ in your web browser to access Kustosz. To log in, use credentials printed in container log after "Generated random login credentials" line.

CHAPTER

SIX

PRODUCTION DEPLOYMENTS

While you can use above command to start and run your main instance of Kustosz, in production environments we recommend using container orchestration tool, such as docker-compose. We have installation instruction pages for few common deployment targets:

- *Containers* - docker and podman, including docker-compose and podman pods
- *Heroku*
- *Your own server* - including Raspberry Pi, DigitalOcean Droplets, VPS, virtual machines and shared hosting accounts
  - *Using installer*
  - *Manual installation*

CHAPTER

SEVEN

QUICK OVERVIEW FOR SOFTWARE DEVELOPERS

If you are software developer and you don't want to plow through detailed instructions, here's a quick summary.

Kustosz is Django application. It requires Python 3.9 or newer and works on Django 3.2 and newer. Data is stored in database engine supported by Django ORM. Default setup uses filesystem-backed SQLite, but we recommend PostgreSQL.

Kustosz requires Celery process to handle background tasks. Celery requires transport and backend. We use file system as default transport, but we recommend Redis. We use Django database as default backend.

You have to trigger channel update process periodically. We provide ready-to-use Celery Beat configuration, but it requires starting and maintaining additional Celery process. If you don't like this, you can use OS scheduler, such as cron or systemd.

If you are unsure about any of the above, you can find detailed instructions on *manual installation* page.

CHAPTER

EIGHT

INITIAL SETUP

Kustosz is running and accessible, but it doesn't have any data. You need to create new account to log in the web interface. We recommend that you also import feed data from OPML file and set new value of `SECRET_KEY` setting.

Instructions provided below should be done only once, after initial installation of Kustosz. You don't need to repeat them after upgrading to newer version.

Some installation methods perform some or all of actions listed below automatically, in which case you can skip them.

8.1 Creating new user

Kustosz supports only single user, and you need to create it to access the application. This can be done with following command:

```
kustosz-manager createsuperuser --email doesnt@matter.invalid
```

Program will ask for desired username and password. Username can be any string, but you will need to provide it every time you log in, so you might opt in to something that is easy to type.

Each user has to have email address - this is requirement imposed on us by Django, one of Kustosz dependencies. Kustosz never uses your email address and doesn't mind if you provide non-existing address.

8.2 Importing OPML file

Most feed readers have an option to export list of subscribed channels into OPML file. If you have such file, you can import it in Kustosz with following command:

```
kustosz-manager import_channels --file <path/to/file.xml> opml
```

Importing OPML file can take some time, depending on number of channels and amount of content they have published.

You can use above command at any time after installing Kustosz. If OPML file contains reference to feed that you have added earlier, Kustosz will automatically skip it.

8.3 Setting new value of SECRET_KEY

SECRET_KEY is Django setting used for various security-related purposes. Your Kustosz installation should have unique SECRET_KEY value.

You can set new value any time you want, but you will have to log in again on all your devices.

Obtain new value by running:

```
kustosz-manager generate_secret_key
```

Then, depending on your environment, put it in your `settings.yaml` file or provide as environment variable named DYNACONF_SECRET_KEY.

CHAPTER

NINE

BASIC USAGE

Kustosz user interface follows familiar pattern used by many other news reading applications: navigation pane on the left and central list of items on the right. Article below is not exhaustive and focuses primarily on features that are specific to Kustosz or might be hard to discover. Two prominent features are documented on separate pages: *Searching* and *Filters*.

9.1 Adding new channel

Channel is source of content displayed in Kustosz. Currently RSS, Atom and JSON Feed are supported.

To add a new channel, click "Add content" button on top of navigation pane. In window that appears, paste URL of website you would like to subscribe to. Kustosz will automatically detect channels provided by the website. Click on channel name to get preview of few latest entries. Click "Add channel" to subscribe to channel.

Alternatively, you can click down arrow right to "Add content" and click "Add channel". In window that appears, you need to provide full URL of XML file. You can also specify channel title and tags.

9.2 Keyboard navigation

You can use your keyboard to quickly move through articles:

- `j` - move to next article and open it
- `k` - move to previous article and open it
- `n` - move to next article without opening it (currently opened article will remain open)
- `b`, `p` - move to previous article without opening it (currently opened article will remain open)
- `o`, `Space`, `Enter` - open currently selected article, or close if it's already opened
- `m` - mark currently selected article as read, or mark it back as unread

9.3 Adding web page manually

You can add any web page to Kustosz. This is especially useful for older articles that no longer appear in website feed, for articles published by websites that don't publish feed, or when you don't want to subscribe to feed.

9.3.1 From Kustosz

Click "Add content" button on top of navigation pane. In window that appears, paste URL of entry you would like to add. Kustosz will also detect channels provided by website, if any. Click "Add article" to add specified web page.

Alternatively, you can click down arrow right to "Add content" and click "Add article". In window that appears, you need to provide URL of article. You can also specify tags.

9.3.2 From web browser

Bookmarklets are special bookmarks that execute JavaScript in the context of currently opened page. They are supported by all major web browsers.

Kustosz automatically creates ready to use bookmarklet for you. Click "Settings" at the bottom of navigation panel and scroll down to "Add to Kustosz" link. Copy link target and paste it in "URL" field of a new bookmark; or drag-and-drop the link to bookmarks bar (be wary that many browsers hide bookmarks bar by default).

Whenever you open a page you would like to add to Kustosz, click a bookmark you have added. Instead of opening bookmarked page, browser will send HTTP request to Kustosz, and Kustosz will add visited page.

On mobile device, you need to tap on address bar and start typing your bookmark name. Opening bookmark from separate Bookmarks screen will not work.

9.3.3 From iOS device

This method requires token. See *How to obtain the token?* below.

iOS comes with "Shortcuts" app. You can use it to automate certain tasks on your mobile device.

Open "Add to Kustosz" shortcut page on your iPhone or iPad and confirm that you want to open Shortcuts app. App will guide you through adding new shortcut, including setting up Kustosz instance address and token.

After shortcut has been added, there will be "Add to Kustosz" action available in website sharing menu.

9.3.4 From terminal

This method requires token. See *How to obtain the token?* below.

You can use curl, which should be available on most Linux machines, to send HTTP request from your terminal:

```
curl -X POST 'http://KUSTOSZ_URL/api/v1/entries/manual_add' \
    -H 'Content-Type: application/json' \
    -H 'Authorization: Token KUSTOSZ_TOKEN' \
    -d '{"link": "http://URL_OF_WEB_PAGE"}'
```

9.3.5 From the machine where Kustosz is installed

This method doesn't require any setup, but can only be exercised on a machine where Kustosz is installed:

```
kustosz-manager add_entry --link 'http://URL_OF_WEB_PAGE'
```

9.4 Exporting channels to OPML file

You can export Kustosz channels to OPML file, making it easy to migrate away from Kustosz to another feed reader.

Click "Settings" at the bottom of navigation panel and scroll down to "Export data" section. Then, click "Export channels as OPML" button.

Alternatively, use following command on a machine where Kustosz is installed:

```
kustosz-manager export_channels --file /path/to/file.xml
```

9.5 Automatic channels update frequency

When editing a channel, you can set automatic update frequency - how often Kustosz should check for new content. Default value is every one hour. If channel is lower priority for you or is known to publish seldom, you can check it less often. If you want to ensure that you learn about new content quickly enough and channel publishes a lot of content, you can check it more frequently.

Every time you check for new content, website server uses a little bit of resources. Some websites might be behind firewall or CDN that will detect too frequent connections and block them, thinking this might be some sort of attack. That's why you should not check for new content *too often*. Every 30 minutes is reasonable lower limit.

Kustosz relies on external scheduler to start channels update procedure - you might have set it up during installation, depending on your deployment environment. We recommend that channels update process is run every 5 minutes. This value puts effective limit on channels update frequency. If you have set up external scheduler to check for updates every hour, and set update frequency of specific channel to 5 minutes, that channel will still be updated every hour.

9.6 Using local file as channel source

Kustosz can read content from any XML file that follows RSS or Atom specification. While most of these files are delivered through web, you can use local file as well. This way Kustosz can read any content source it does not handle natively - as long as you can transform it into RSS or Atom file.

First, save XML files inside the `$KUSTOSZ_BASE_DIR/feeds` directory. You can create directories inside to organize your files. Then, run following command:

```
kustosz-manager import_channels --wait autodiscover
```

If you add new files in the future, just re-run the command.

9.7 Deactivating a channel

When modifying a channel, you can toggle "active" flag. When "active" flag is disabled (channel is not active), it will not be checked for new content during channels update.

Deactivating a channel is intermediate step before removing it. Channel is not checked for new content, but all past content is still available in the application. This is especially useful for channels that disappeared from the web, stopped being updated or are very active and you want to silent them for a while.

9.8 Deduplication

Deduplication automatically marks some entries as read, so they don't appear on the default list of entries when opening the application.

Deduplication is turned on by default. You can opt-out specific channels by turning off "is deduplicated" flag. When flag is disabled, entries from that channel are never marked as read by deduplication algorithm. You can also turn off deduplication altogether by setting `KUSTOSZ_DEDUPLICATE_DAYS` setting to `0`.

Deduplication works only across channels, i.e. entries from one channel are never considered a duplicates. That's because some authors use the same title for all of their posts.

Deduplication algorithm looks into entry GID, normalized link and author-title pair. If any of these values is the same as for one of other entries, latter entry is considered a duplicate. By default, deduplication algorithm looks into all entries added in last 2 days.

9.9 Maintenance section

Maintenance section provides views to quickly (de)activate multiple channels at once.

9.9.1 Stale channels

Channels become stale when Kustosz determines it's no longer possible to access them.

In general, it's normal for resources to become temporarily unavailable - websites are upgraded to new versions, are unable to cope with amount of traffic after going viral on social media, people forget to pay hosting bills. That's why Kustosz will mark channels as stale only if they could not be accessed for extended period of time: during last 10 tries (determined by update frequency) or in last three days, whichever is longer.

It's worth noting that stale channels might have been moved to new URL without leaving proper redirection in place. It's up to you as an user to determine if that is the case.

9.9.2 Channels without new entries

These channels are online and can be accessed, but have not produced a new entry in number of days (30 by default).

9.9.3 Inactive channels

These channels have "active" flag turned off. You can activate them back or delete them.

Deleting channel automatically deletes all channel entries. However, if any entry is tagged, Kustosz will ask if you want to delete tagged entries or keep them. If you decide to keep them, they will be moved from deleted channel to special "Manually added" channel with id 1.

9.10 How to obtain the token?

Authorization token is like password - it allows Kustosz to decide if incoming HTTP request should be allowed to perform an action. You should not share authorization token with other people.

Token is included in all requests that Kustosz UI sends to server. You can open web browser development tools (F12), click Network tab and select any request of XHR type. In Request Headers section, there will be `Authorization` header with value `Token 0123456789abcdef0123456789abcdef01234567`. This long string of random letters and numbers is a token.

Another way is running following command:

```
kustosz-manager drf_create_token KUSTOSZ_USER
```

CHAPTER

TEN

SEARCHING

Searching allows to change content displayed on main articles list.

10.1 Basic search

Basic search is displayed by default and allows to use graphical user interface to perform basic queries. Basic search interface is intentionally limitied to few most often used options.

- Unread only button - when this button is selected (which it is by default), archived entries will not be displayed. When it is pressed off, both archived and unarchived entries are displayed.
- Date selector - allows to see only entries published before / after yesterday, 2 days ago, a week ago or 2 weeks ago.
- Tag selector - allows to see entries tagged with specific tag. Multiple tags are alternated (joined with conjunction OR), i.e. if entry is tagged with any of the provided tags, it will be on the results list.

10.2 Advanced search

Advanced search allow to perform much wider range of queries, but they require you to write queries by hand. At the moment, there is no autocompletion support.

Table below contains list of fields and possible lookups. Fields and lookups are separated by double underscore: `field__lookup`. Default lookup method, `exact`, can be skipped. Value for lookup is specified after equals sign: `field=value`. Some lookup methods can take in multiple arguments - they are separated by comma. Multiple query parameters are joined with ampersand: `field1=value&field2=value`. See *Examples* below.

Lookup semantics are generally the same as in Django. See Django documentation on field lookups for details. Special lookup method `not` is logical negation - it will match everything **except** specified value.

10.2.1 Fields and lookups

Field	Lookup methods	Notes
`archived`	• `exact`	Boolean field that takes `true` and `false` as values.
`link`	• `exact` • `iexact` • `contains` • `icontains` • `startswith` • `istartswith` • `endswith` • `iendswith`	
`title`	• `exact` • `iexact` • `contains` • `icontains` • `startswith` • `istartswith` • `endswith` • `iendswith`	
`reading_time`	• `exact` • `lt` • `gt` • `lte` • `gte`	Estimated time needed to read article, in minutes. Seconds are expressed as fraction of minute, e.g. 1 minute 30 seconds is `1.5`.
`reader_position`	• `exact` • `lt` • `gt` • `lte` • `gte`	Part of article that you have read, as fraction between 0 and 1. Half of article is `0.5`.
`published_time`	• `exact` • `lt` • `gt` • `lte` • `gte`	Publication time as specified by source feed. It will fall back automatically to last updated time if publication time is not specified by source feed.
`published_time_upstream`	• `exact` • `lt` • `gt` • `lte` • `gte`	Publication time as specified by source feed. Many feeds skip this field, and some broken feeds also change it when entry is updated. You almost certainly want to use `published_time` instead.
`updated_time_upstream`	• `exact` • `lt` • `gt` • `lte` • `gte`	Last updated time as specified by source feed. Most content is never updated, so it is often the same as `published_time_upstream`. You almost certainly want to use `published_time` instead.
`added_time`	• `exact` • `lt` • `gt`	Time when entry was added to Kustosz database. When you add new channel, all entries will have the same `added_time`. You almost certainly

10.2.2 Working with tags

Under the hood, tags have names (user-visible strings) and slugs (URL-friendly strings). Slugs are automatically derived from names using `django.utils.text.slugify()` function (click the link for description of exact transformations applied).

As an user, you are mostly dealing with names. However, advanced search expects you to use tag **slugs**. That begs the question: how can you obtain the tag slug?

The easiest way is opening browser development tools and refreshing the page. In "Network" tab, you will see requests to `/api/v1/tags/entry` and `/api/v1/tags/channel` endpoints. Responses of these requests contain the list of all entry and channel tags, respectively. Each item on the list will have both `name` and `slug` property.

Alternatively, you can connect to the host where Kustosz is running and run following command to open development shell:

```
kustosz-manager shell
```

Then, run following commands to get the list of all tags:

```
from kustosz.models import Channel, Entry
[f"{i.name}: {i.slug}" for i in Channel.tags.all()]
[f"{i.name}: {i.slug}" for i in Entry.tags.all()]
```

10.2.3 Examples

Entries published in a year 2022:

```
published_time__gte=2022-01-01T00:00:00&published_time__lte=2022-12-31T23:59:59
```

Entries with title that contain the word "test", case insensitive:

```
title__icontains=test
```

Entries published in a year 2022 with title that contains the word "test", case insensitive:

```
published_time__gte=2022-01-01T00:00:00&published_time__lte=2022-12-31T23:59:59&title__
↪icontains=test
```

Entries with title that starts with the phrase "The quick brown fox", case sensitive (note space is not escaped or put in quotes):

```
title__startswith=The quick brown fox
```

All entries, except these tagged with tag "slug" or "snail" (this includes entries that are not tagged at all):

```
entry_tags__not=slug,snail
```

Entries coming from three specific channels:

```
channel=5,10,15
```

Entries coming from channels that are not tagged (entries themselves may be tagged):

```
channel_has_tags=false
```

CHAPTER

ELEVEN

FILTERS

Filters do something with entries matching criteria. Filters act on entries as they come in.

11.1 Filtering criteria

At the moment, filtering criteria must be input manually. Criteria field works exactly the same as *Advanced searching* field. You can use the same fields and lookup methods, and you can compose them into advanced queries using ampersand.

11.2 Actions

Each action works on a single matching entry. If there are multiple matching entries, action will be run this many times.

11.2.1 Mark as read

Automatically marks matched entry as read. It will be stored in Kustosz, but you won't see it in main entries list by default.

This is especially useful for feed aggregators (planets) and larger sites that have multiple authors (perhaps you don't fancy some of them). If you want to hide all entries from specific channel, consider *deactivating the channel*.

11.2.2 Assign tag

Add a **single** tag to matching entry. This action requires an argument, which is a tag name. Tag will be created when assigned for the first time, if it does not exist.

11.2.3 Run external application

Run external application (script) on matching entry. This action requires an argument, which is a path to executable. We recommend using absolute paths.

Application will be run only once. If you want to handle expected failure conditions, do so in your application.

Kustosz expects that external application will finish within 30 seconds. If your action might need more time, your application should be lightweight wrapper that starts the long-running script outside of the main process and returns. In shell, consider using `nohup` or `disown`.

Kustosz will pass a number of information through environmental variables. The full list is below. Note that all environmental variable values are strings - you might need to cast them to integers, booleans or dates in your application.

- KUSTOSZ_ID - internal id of entry. Can be used if you want to communicate back with Kustosz using API or `kustosz-manager shell`
- KUSTOSZ_GID - id of entry, as claimed by source feed
- KUSTOSZ_ARCHIVED - boolean value specifying whether entry is archived (read) or not
- KUSTOSZ_LINK - URL of entry, as claimed by source feed
- KUSTOSZ_TITLE - title of entry
- KUSTOSZ_AUTHOR - author field of entry, as claimed by source feed
- KUSTOSZ_PUBLISHED_TIME_UPSTREAM - timestamp when entry was published, as claimed by source feed; many feeds skip this field, and some broken feeds also change it when entry is updated
- KUSTOSZ_UPDATED_TIME_UPSTREAM - timestamp when entry was last updated, as claimed by source feed; most content is never updated, so it is often the same as KUSTOSZ_PUBLISHED_TIME_UPSTREAM
- KUSTOSZ_ADDED_TIME - timestamp when entry was added to Kustosz database; you almost certainly want to use KUSTOSZ_PUBLISHED_TIME_UPSTREAM
- KUSTOSZ_UPDATED_TIME - timestamp when entry was last updated in Kustosz database; you almost certainly want to use KUSTOSZ_UPDATED_TIME_UPSTREAM
- KUSTOSZ_TAGS - comma-separated list of entry tags slugs
- KUSTOSZ_CHANNEL_ID - internal id of channel
- KUSTOSZ_CHANNEL_URL - URL of channel
- KUSTOSZ_CHANNEL_CHANNEL_TYPE - internal value specifying channel type; see `ChannelTypesEnum` in server source
- KUSTOSZ_CHANNEL_DISPLAYED_TITLE - title of channel, as visible in user interface; KUSTOSZ_CHANNEL_TITLE when provided, with automatic fallback to KUSTOSZ_CHANNEL_TITLE_UPSTREAM
- KUSTOSZ_CHANNEL_TITLE - title of channel, as provided by the user; may be empty and you almost certainly want to use KUSTOSZ_CHANNEL_DISPLAYED_TITLE
- KUSTOSZ_CHANNEL_TITLE_UPSTREAM - title of channel, as claimed by source feed; you almost certainly want to use KUSTOSZ_CHANNEL_DISPLAYED_TITLE
- KUSTOSZ_CHANNEL_LINK - URL of channel, as claimed by source feed
- KUSTOSZ_CHANNEL_LAST_CHECK_TIME - timestamp when channel was last checked for new content
- KUSTOSZ_CHANNEL_LAST_SUCCESSFUL_CHECK_TIME - timestamp when last check for new content was successful
- KUSTOSZ_CHANNEL_ADDED_TIME - timestamp when channel was added to Kustosz database
- KUSTOSZ_CHANNEL_ACTIVE - boolean value specifying whether channel is active or not
- KUSTOSZ_CHANNEL_UPDATE_FREQUENCY - how often channel is checked for new content, in seconds
- KUSTOSZ_CHANNEL_DEDUPLICATION_ENABLED - boolean value specifying whether deduplication is enabled for channel or not
- KUSTOSZ_CHANNEL_TAGS - comma-separated list of channel tags slugs

CHAPTER

TWELVE

CONFIGURATION

12.1 Installer

Installer is implemented as Ansible collection and is configured the same way other Ansible roles are - by putting variable definitions in place where Ansible can read them.

Ansible supports multiple levels of variable precendence rules. Unless you want to incorporate installer in your own custom playbook, it's best to define variables in inventory file - just as it was shown on *installer* page.

Default values of variables listed below can be found in `roles/set_variables/defaults/main.yml` file inside installer repository.

12.1.1 `system_user_name`

Username of system user that will run Kustosz. User is created automatically, unless `run_system_requirements_root` is set to `false`.

12.1.2 `system_user_home`

Path to home directory of system user that will run Kustosz. Directory is created automatically when creating user. Most of paths mentioned below are by default constructed relative to this directory.

12.1.3 `system_user_shell`

Shell of system user that will run Kustosz. Shell is set automatically when creating user. Should be POSIX-compliant (i.e. **not** fish). There's little reason to log in as Kustosz user, apart from occasional maintenance task.

12.1.4 `run_system_requirements_root`

When `false`, during first phase installer will not run steps that require superuser permissions. These steps are: installing system packages, starting and enabling NGINX and Redis system services, creating system user, ensuring `system_user_home` directory is globally readable.

Set to `false` only when you don't have root permissions on the machine.

12.1.5 use_postgres

When `true`, installer will install system packages required to build psycopg2 (Python PostgreSQL driver), as well as psycopg2 itself in Kustosz virtual environment. This prepares your system to use Kustosz with PostgreSQL database.

Note that installer will **not** install Postgres itself, will not create Postgres database and will not configure Kustosz to connect with Postgres. Postgres configuration must be done manually.

12.1.6 use_system_nodejs

When `true`, installer will use system-provided Node.JS instead of NVM-provided one. Node.JS is needed by kustosz-node-readability. While kustosz-node-readability should work on all Node.JS LTS versions, it is only tested with Node.JS 16.

Note that installer will **not** check if Node.JS is installed at all.

12.1.7 nodejs_version

Node.JS version string passed to `nvm install`.

12.1.8 nvm_path

Directory where NVM itself and NVM-managed Node.JS will be installed. This is called `NVM_DIR` by NVM.

12.1.9 use_system_python

When `true`, installer will use system-provided Python instead of pyenv-provided one. Kustosz requires Python 3.9 or newer.

Note that installer will check Python version and will emit appropriate failure message if system-provided Python version is too old.

12.1.10 python_path

Path to Python binary. Binary names will be searched for in `$PATH`.

12.1.11 pyenv_path

Directory where pyenv itself and pyenv-managed Python will be installed. This is called `PYENV_ROOT` by pyenv.

12.1.12 pyenv_python_version

Version of Python to install with pyenv. This is passed to `pyenv install` and used while checking if required Python version is already installed.

12.1.13 kustosz_frontend_version

Version of Kustosz web-ui to install. Special string "latest" evaluates to latest released version.

12.1.14 kustosz_backend_version

Version of Kustosz server to install. Special string "latest" evaluates to latest released version.

12.1.15 venv_path

Directory where Kustosz virtual environment will be created.

12.1.16 force_install_backend

By default, installer tries to *not* install server package - it will do so only if it was unable to determine installed version, or when requested version is different than installed version. By setting this to `true`, installer will always create new virtual environment and install Kustosz backend server.

12.1.17 force_install_frontend

By default, installer tries to *not* install frontend files - it will do so only if it was unable to determine installed version, or when requested version is different than installed version. By setting this to `true`, installer will always remove existing frontend files and download fresh copy from GitHub.

12.1.18 extra_python_packages

Any extra packages you want to install in Kustosz virtual environment. This variable is passed to `python -m pip install`. You may use it to install memcached driver or force specific version of Django.

12.1.19 kustosz_base_dir

Value of `$KUSTOSZ_BASE_DIR` environment variable. Kustosz will read settings from directory created here.

12.1.20 db_path

Directory where default SQLite database file will be created.

12.1.21 settings_path

Directory where `settings.yaml` and `settings.local.yaml` files will be put. Kustosz server requires this to be `$KUSTOSZ_BASE_DIR/settings`.

12.1.22 frontend_path

Directory where frontend files will be downloaded. This is referenced in NGINX virtual host configuration file, so NGINX server user needs to be able to read it. If you decide to use WhiteNoise, that path should be referenced in `STATICFILES_DIRS`.

12.1.23 kustosz_settings_allowed_hosts

A list of strings representing the host/domain names that this Django site can serve. This is the same as Django's `ALLOWED_HOSTS` setting.

When this is set to empty list, installer will construct new list from `kustosz_nginx_server_name` value. If you want Kustosz to be available under single domain name, it's recommended to set only `kustosz_nginx_server_name`.

12.1.24 settings_local_path

A **local** path to file that will become `settings.local.yaml` (see *Local files section on backend configuration page*). This file is input value to `ansible.builtin.template`, which means that you can use jinja2 and all the variables documented on this page.

This file will always overwrite existing `setting.local.yaml`. If you have made any changes to your `setting.local.yaml` after installation, make sure this variable is unset during next installer run.

When running installer on machine that will run Kustosz, this path should still be different than `setting.local.yaml`.

12.1.25 configure_nginx_server

When `false`, during post-installation phase installer will not run steps related to NGINX server configuration. These steps are: configuring SELinux variables to allow NGINX to make HTTP connections, creating Kustosz virtual host configuration file in `/etc/nginx/` and running certbot, assuming `run_certbot` is `true`.

Set to `false` when you don't have root permissions on the machine or when you don't use NGINX as HTTP server.

12.1.26 configure_system_services

When `false`, during post-installation phase installer will not run steps related to system services configuration. These steps are: putting special dispatcher script in `systemd_dispatcher_path`, creating template service file under `/etc/systemd/system/`, and starting and enabling four background tasks for Kustosz (gunicorn server, two Celery workers, and Celery beat).

Set to `false` when you don't have root permissions on the machine or when you don't use systemd as init.

12.1.27 nginx_template_path

A **local** path to file that will become NGINX virtual host configuration file. This file is input value to `ansible.builtin.template`, which means that you can use jinja2 and all the variables documented on this page.

12.1.28 kustosz_internal_port

Port to which gunicorn (web server platform that runs Kustosz) process should bind to. Most of userspace web servers bind to port 8000, 8080, or similar, and depending on number of services you are running, the default port might be already taken.

12.1.29 kustosz_nginx_listen

External port on which Kustosz should be accessible. NGINX must listen on this port. Usually it's 80 or 443.

This value is passed as argument to NGINX `listen` directive.

12.1.30 kustosz_nginx_server_name

Domain name of Kustosz instance. In standard NGINX setup, multiple sites listen on the same port, and NGINX matches request's `Host` header against `server_name` directive to decide which virtual host should handle specific request.

This value is passed as argument to NGINX `server_name` directive.

12.1.31 kustosz_nginx_extra_config

Anything you want to include in NGINX virtual host configuration file for Kustosz.

12.1.32 run_certbot

When `true`, installer will run `certbot` to configure Let's Encrypt TLS certificate on virtual host.

Installer will **not** check if certbot is available or configured correctly - it blindly tries to run the binary. Set this to `true` only if you actually use certbot and have configured it on the machine.

12.1.33 certbot_extra_args

Additional command-line options passed to `certbot`. By default, installer includes `--non-interactive`, `--nginx` and `--domains $kustosz_nginx_server_name`.

You need to ensure that certbot is able to run without any human interaction. At the very least, you should include `--agree-tos` and `-m EMAIL@example.com`.

12.1.34 systemd_dispatcher_path

Path to background services dispatcher script used by systemd. Dispatcher script exists to ensure that background processes run with correct environment variables set.

12.1.35 opml_local_path

Local path to directory with OPML files. Directory must exist.

When defined, installer will import OPML files into Kustosz.

12.1.36 opml_path

Path on server where OPML files will be uploaded. When running installer on machine that will run Kustosz, this path should still be different than `opml_local_path`.

12.1.37 web_user_name

User name that you will use to log in to Kustosz web interface.

12.1.38 web_user_password

Password that you will use to log in to Kustosz web interface.

12.2 Backend

Kustosz uses central configuration storage to maintain configuration for both Kustosz itself and all of the dependencies (Django, Celery etc.). Such central configuration is possible thanks to excellent Dynaconf library. This page covers everything that you need to know to configure Kustosz, but you might want to take a look at Dynaconf documentation, too.

12.2.1 Configuration sources

Kustosz will read settings from configuration files and environment variables. Environment variables override settings from files.

Local files

The main configuration file is `$KUSTOSZ_BASE_DIR/settings/settings.yaml`. You have created this file during installation. This file is already present in container image as `/opt/kustosz/web/settings/settings.yaml`.

When `$KUSTOSZ_BASE_DIR` variable is not set, it will default to parent directory of `settings.py` file. In production deployments, that's usually `lib/python3.x/site-packages/` directory inside current virtual environment. In development environment, that's root of git tree.

Site-specific modifications can be put in `$KUSTOSZ_BASE_DIR/settings/settings.local.yaml` file. This file will be merged with main configuration file, i.e. any setting not specified in site-specific file will be read from main file. This allows you to copy newest Kustosz configuration file during upgrades without worrying about overwriting your local changes.

Main configuration file has two top-level keys: `default` and `production`. These are called "layered environments". You can switch between environments by setting `ENV_FOR_DYNACONF` variable prior to starting Kustosz. Selected environment is merged with `default` environment, i.e. any setting not specified in `production` environment will be read from `default`.

Site-specific modifications and layered environments

Layered environments are effectively applied **after** site-specific modifications. Consider these two files:

Listing 1: settings.yaml

```yaml
default:
    STATIC_ROOT: "./static/"

production:
    STATIC_ROOT: "/opt/kustosz/web/static/"
```

Listing 2: settings.local.yaml

```yaml
default:
    STATIC_ROOT: "/home/kustosz/website/static_files/"
```

Assuming `ENV_FOR_DYNACONF=production`, value of `STATIC_ROOT` setting will be **`/opt/kustosz/web/static/`**. That's because `settings.local.yaml` overrides `settings.yaml`, but `production` overrides `default`.

When using layered environments, it is recommended that site-specific configuration file work only on `production` environment.

Overriding single value in nested structure

Django uses complex (hierarchical, nested) structures for some of their settings. If you only want to change single value in hierarchy, double underscore can be used to express "one level down".

Assuming `ENV_FOR_DYNACONF=production`, following file:

```
default:
    DATABASES:
      default:
        ENGINE: 'django.db.backends.sqlite3'
        NAME: 'db.sqlite3'

production:
    DATABASES__default__NAME: "/tmp/kustosz.db"
```

would be the same as:

```
production:
    DATABASES:
      default:
        ENGINE: 'django.db.backends.sqlite3'
        NAME: '/tmp/kustosz.db'
```

You can use this method to add new keys to nested structure. This new key itself can also be complex, containing other nested items.

Environment variables

Environment variables should be the same as setting names, but prefixed with `DYNACONF_`. For example, if you want to change value of setting `KUSTOSZ_LOCK_EXPIRE`, then environment variable should be `DYNACONF_KUSTOSZ_LOCK_EXPIRE`.

Environment variables do not support site-specific settings or layered environments. They are simple source of data that overrides everything else.

It is possible to use environment variables to override only one value in complex (hierarchical, nested) structure with the help of double underscores. This works the same as in files.

Using environment variables to override single value in nested structure

Consider following `settings.yaml` file:

```
default:
    DATABASES:
      default:
        ENGINE: 'django.db.backends.sqlite3'
        NAME: 'db.sqlite3'
```

If you want to change path where database file will be stored (while still using sqlite3 engine), you can do so with following environment variable:

```
export DYNACONF_DATABASES__default__NAME="/home/kustosz/data/sqlite.db"
```

Environment variable values are assumed to be in TOML. It means that boolean, numbers and dates will be cast to their proper type by default. It's easy to specify lists or key-value pairs. You can also force specific type with `@type` syntax. Complex structures can be specified in JSON. See Dynaconf documentation on environment variables for some examples.

12.2.2 Verifying current configuration

With site-specific file overrides, layered environments and environment variables, it might be hard to determine what value will be used. You can use `kustosz-manager print_settings` and `dynaconf list` commands to see all settings and their values.

Before using the command, make sure that `DJANGO_SETTINGS_MODULE` and `ENV_FOR_DYNACONF` variables are set.

```
export DJANGO_SETTINGS_MODULE=kustosz.settings
export ENV_FOR_DYNACONF=production
kustosz-manager print_settings
dynaconf list
```

12.2.3 Available settings

KUSTOSZ_DEDUPLICATE_DAYS

Number of past days to consider when checking if new entry is a duplicate. Increasing this number will make deduplication slower, but makes it less likely that you see duplicates on your main list.

Setting this to `0` will disable deduplication altogether.

See also *Deduplication documentation*.

KUSTOSZ_READABILITY_NODE_ENABLED

Should Kustosz use kustosz-node-readability to obtain full content of new articles.

kustosz-node-readability uses Readability.js, which is used for Firefox Reader View and by Pocket. It's probably the best maintained readability implementation around, but it requires Node.js.

KUSTOSZ_READABILITY_NODE_EXECUTABLE

String with name of kustosz-node-readability executable.

This may be a list of strings, which might be useful if you don't have kustosz-node-readability in your `$PATH`, but you still meet all the other requirements to run it.

This is ignored if `KUSTOSZ_READABILITY_NODE_ENABLED` is False.

KUSTOSZ_READABILITY_PYTHON_ENABLED

Should Kustosz use python-readability to obtain full content of new articles.

python-readability is smaller project than Readability.js, and it may miss some content that JavaScript implementation can handle. But it's written in pure Python, so you can surely run it if you can run Kustosz.

KUSTOSZ_READING_SPEED_WPM

Number of words you read per minute. Used when calculating estimated reading time for entries.

Estimated reading time is calculated once and stored in database, so changing this setting will not affect existing articles.

KUSTOSZ_URL_FETCHER_EXTRA_HEADERS

Mapping of key-value pairs that represent requests_cache session headers. These headers are set after requests_cache session initialization, so they will be present in addition to default headers - and can overwrite them. There are two main use cases: setting `Accept-Language` to negotiate preferred content language, and setting fake `User-Agent` to avoid 403 Forbidden error that some servers send in response to unknown / programmatic user requests.

requests_cache is used when downloading HTML pages, which happens for entries added manually and content extracted with readability, assuming at least one of `KUSTOSZ_READABILITY_*_ENABLED` settings is set to True.

`User-Agent` set here will also be used as fallback during standard feed fetching, if first request failed with 403. Some misconfigured sites prevent programmatic access to RSS / Atom files, even though these files are meant to be read by computers.

KUSTOSZ_PERIODIC_FETCH_NEW_CONTENT_INTERVAL

How often should Kustosz check for new content of feeds, in minutes.

This value is used only once, during initial database migration. Changing this setting later doesn't affect anything, as value that is actually used is stored in database.

This value takes effect only if you maintain Celery beat process. It is ignored if you start periodic updates through operating system task scheduler.

KUSTOSZ_REQUESTS_CACHE_INIT_OPTIONS

Mapping of key-value pairs passed verbatim to `requests_cache.session.CachedSession()`. Main use case is specifying path to directory where cache will be stored.

requests_cache is used when downloading HTML pages, which happens for entries added manually and content extracted with readability, assuming at least one of `KUSTOSZ_READABILITY_*_ENABLED` settings is set to True.

KUSTOSZ_FEED_READER_WORKERS

Number of threads to use when getting the feeds. Passed verbatim to `reader.Reader.update_feeds()`.

KUSTOSZ_FETCH_CHANNELS_CHUNK_SIZE

Number of channels to update at once. You might want to lower this setting if your server has limited memory.

KUSTOSZ_FETCH_PAGE_MAX_RETRIES

Number of times to try to download HTML page before giving up. This affects entries added manually and content extracted with readability, assuming at least one of `KUSTOSZ_READABILITY_*_ENABLED` settings is set to True.

KUSTOSZ_LOCK_EXPIRE

When should internal locks expire. You might need to increase this number if you increased `KUSTOSZ_FETCH_CHANNELS_CHUNK_SIZE` or your server is particularly slow.

12.2.4 Django settings

Kustosz is built with Django, so all Kustosz settings are also Django settings. In fact, Kustosz settings are just special cases of general Django settings.

All Django settings are documented in Django documentation on Settings. They cover things like database connection, caching engine, local paths etc. You can use Django settings in `settings.yaml` as-is, with only exception being that Django uses pure Python and Kustosz uses YAML.

12.2.5 Celery settings

If you want to change Celery configuration, you need to make setting name all-uppercase and prefix it with `CELERY_`. For example, to change Celery setting `broker_url`, use Kustosz configuration variable `CELERY_BROKER_URL`.

All Celery settings are documented in Celery documentation on Settings.

Using environment variables to change Celery settings

Changing Celery settings by using environment variables is possible, but note about prefixing setting names still applies. To change Celery setting `broker_url`, set environment variable `DYNACONF_CELERY_BROKER_URL`.

12.3 Frontend

Frontend configuration can be changed in web UI, by accessing "Settings" at the bottom of navigation panel.

12.3.1 Theme

You can choose between Light and Dark themes.

12.3.2 Automatically mark article as read

You can decide when article should be marked as read automatically, if ever. Available options:

- "Never" - disables automatic marking of articles.
- "Upon opening" - article will be marked as read immediately when you open it.
- "When opened for X seconds" - article will be marked as read when it is opened for specified number of seconds. If you close the article before timer runs out, it will not be marked as read. Timer starts from beginning when you open the same article again.
- "When X% has been read" - article will be marked as read once you scroll past the specified proportion of article content. 100% is special and means "as soon as the bottom of article is visible". Except 100%, values above 70% might be hard to trigger on short articles.

12.3.3 Behavior

Always keep opened entry on top of list

When this options is checked, article will scroll up to the top of the screen when it is opened. This only matters for articles at the top half of the screen, as articles at bottom half of screen will be always scrolled up.

CHAPTER

THIRTEEN

DEVELOPMENT

Kustosz is open source application, distributed under terms of European Union Public Licence. We welcome all kids of contributions, including code changes, reporting bugs, improving documentation and translating user-interface. Click one of the links below for more details and instructions.

13.1 Backend

Backend is written in Python. It requires Python 3.9 or newer. You will need Poetry version 1.2 or newer to install package and all the dependencies.

13.1.1 Preparing development environment

Clone Github repo:

```
git clone https://github.com/KustoszApp/server
```

Install dependencies:

```
poetry install --with test,dev -E container
```

13.1.2 Changing development version configuration

If you want to change the configuration of development version, the best way of doing so is through `settings.local.yaml` (inside `settings` directory). For example, following file would turn on debug-level logging, while ignoring log messages produced by some third-party libraries:

```
default:
  LOGGING__root__level: 'DEBUG'
  LOGGING__loggers__reader__handlers:
    - 'null'
  LOGGING__loggers__readability__handlers:
    - 'null'
  LOGGING__loggers__requests_cache:
    handlers:
      - 'null'
    propagate: False
  LOGGING__loggers__requests:
    handlers:
```

(continues on next page)

(continued from previous page)

```
    - 'null'
  propagate: False
```

13.1.3 Running current development version

Commands below assume that current virtual environment is active. You can spawn shell with activated virtual environment using:

```
poetry shell
```

Run migrations:

```
python manage.py migrate
```

Create cache tables:

```
python manage.py createcachetable
```

Create user:

```
python manage.py createsuperuser --username admin --email admin@example.invalid
```

Generate authentication token:

```
python manage.py drf_create_token admin
```

Run server:

```
python manage.py runserver
```

Kustosz API server will be available at http://127.0.0.1:8000/. You can use curl or httpx to communicate with it. See *Frontend* for instructions how to run development version of Kustosz frontend.

13.1.4 Running unit tests

Running unit tests using currently installed Django version and Python version used to create active virtual environment is as simple as:

```
pytest
```

If you want to run unit tests against the matrix of all supported Python and Django versions, then run:

```
nox --non-interactive --session "tests"
```

This will work only if you have binaries for all of the Python versions (e.g. `python3.9` and `python3.10`). You can use pyenv or asdf to install them.

13.2 Frontend

Frontend is written in JavaScript, using Vue.js and Node.js. It requires Node.js 14 or newer. We only support LTS releases of Node.js. npm package is currently build using Node.js 16.

13.2.1 Preparing development environment

Clone Github repo:

```
git clone https://github.com/KustoszApp/web-ui
```

Install dependencies:

```
npm install
```

13.2.2 Running current development version

Run server:

```
npm run serve
```

Kustosz UI will be available at http://localhost:8080.

It's not very useful to run frontend without *API server running*. By default, frontend will assume that backend is available at http://127.0.0.1:8000/api/v1. You can specify another URL by setting `VUE_APP_KUSTOSZ_BACKEND_URL` environment variable prior to running `serve` command. Note that variable needs to specify full URL, including path.

```
VUE_APP_KUSTOSZ_BACKEND_URL="http://127.0.0.1:9876/api/v2" npm run serve
```

13.3 Documentation

Documentation, available at docs.kustosz.org, is written in MyST, which is CommonMark with support for Sphinx directives. Basically, it's Markdown.

Documentation is built using Sphinx, which is Python package. It requires any Python 3 version supported by Sphinx.

13.3.1 Preparing development environment

Clone Github repo:

```
git clone https://github.com/KustoszApp/docs
```

Create new virtual environment:

```
python3 -m venv path/to/virtualenv
```

Activate virtual environment:

```
source path/to/virtualenv/bin/activate
```

Install dependencies:

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

13.3.2 Running current development version

Run server:

```
python3 run_livereload.py
```

Kustosz documentation will be available at http://127.0.0.1:5500/. Website refreshes automatically as you change documentation source files.

13.4 Website

Kustosz website, available at www.kustosz.org, is written in Markdown and built using Hugo.

Website requires Hugo 0.82.1 or later and Node.js runtime. Production version of website is currently build using Node.js 16.

13.4.1 Preparing development environment

Clone Github repo along with submodules:

```
git clone --recurse-submodules https://github.com/KustoszApp/kustoszapp.github.io.git
```

Install dependencies:

```
npm install
```

13.4.2 Running current development version

Run server:

```
hugo serve
```

Kustosz website will be available at http://localhost:1313/. Website refreshes automatically as you change documentation source files.

13.5 Installer

Installer is implemented as Ansible collection. You need Ansible 2.9 or newer to run it. We recommend users to create disposable virtual environment and install Ansible from PyPI, so it's generally fine to support only latest released version of Ansible.

13.5.1 Preparing development environment

Create virtual environment and install Ansible:

```
python3 -m venv ~/kustosz-ansible-venv/
. ~/kustosz-ansible-venv/bin/activate
pip install --upgrade pip
pip install ansible
```

You have to put installer files in a place where Ansible can find them. You can either clone directly to Ansible directory, or clone anywhere and bind-mount to Ansible directory:

```
# Option 1: Clone directly to Ansible directory
mkdir -p ~/.ansible/collections/ansible_collections/kustosz/install
git clone https://github.com/KustoszApp/kustosz-installer.git ~/.ansible/collections/
↪ansible_collections/kustosz/install/

# Option 2: Clone anywhere and bind-mount
mkdir -p ~/.ansible/collections/ansible_collections/kustosz/install
git clone https://github.com/KustoszApp/kustosz-installer.git
sudo mount --bind ./kustosz-installer/ ~/.ansible/collections/ansible_collections/
↪kustosz/install/
```

13.5.2 Setting up Vagrant

You need a machine you can test installer on. We suggest using Vagrant, which provides easy way to manage local virtual machines.

You need Vagrant and plugin to virtual machine provider. vagrant-libvirt is included in most distributions, making it one of the easiest to set up - just install vagrant and vagrant-libvirt packages, make sure libvirt system service is started, add your user to `libvirt` group and you are good to go. When in doubt, consult your distribution documentation.

In root directory of installer, create file named `Vagrantfile` with following content:

Listing 1: Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.define "centos8", autostart: false do |centos8|
    centos8.vm.box = "generic/centos8"
    centos8.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"
  end
  config.vm.define "ubuntu2004", autostart: false do |ubuntu2004|
    ubuntu2004.vm.box = "generic/ubuntu2004"
    ubuntu2004.vm.network "forwarded_port", guest: 80, host: 8081, host_ip: "127.0.0.1"
  end
  config.vm.define "ubuntu2204", autostart: true do |ubuntu2204|
    ubuntu2204.vm.box = "generic/ubuntu2204"
    ubuntu2204.vm.network "forwarded_port", guest: 80, host: 8082, host_ip: "127.0.0.1"
  end

  config.vm.provision "ansible" do |ansible|
```

(continues on next page)

(continued from previous page)

```
    ansible.playbook = "playbooks/playbook.yml"
    ansible.verbose = "v"
  end
end
```

13.5.3 Running current development version

Now you can use following commands:

- `vagrant up centos8` - create new CentOS machine, start it and run installer against it
- `vagrant provision centos8` - re-run installer against started virtual machine; useful for testing failure scenarios
- `vagrant ssh centos8` - ssh into started virtual machine
- `vagrant delete -f centos8` - stop and remove running virtual machine

Vagrant will automatically route network traffic from `host:` port to virtual machine. For example, you can access Kustosz running on Ubuntu 20.04 by opening localhost:8081/ui.

13.5.4 Changing development version configuration

If you want to test impact of various *configuration variables*, the best way is to create new file `playbooks/vagrant.yml` with following content:

Listing 2: playbooks/vagrant.yml

```
- name: "Install Kustosz on Vagrant"
  import_playbook: kustosz.install.playbook
  vars:
    use_system_python: true
    kustosz_nginx_server_name: "127.0.0.1 localhost"
    web_user_name: kustosz
    web_user_password: kustosz
    opml_local_path: /tmp/opmls
```

Change `Vagrantfile` to use new playbook during provisioning:

Listing 3: Vagrantfile

```
Vagrant.configure("2") do |config|
  # ...
  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "playbooks/vagrant.yml"
    ansible.verbose = "v"
  end
end
```

13.6 Release policy

Kustosz aims for monthly release schedule, with version numbers based on calendar.

We use `YY.MM.PATCH` version numbering scheme, where `YY` are last two digits of release year, `MM` are two digits for release month and `PATCH` is zero-based index of release in given month. "22.01.0" is first release in January 2022, "22.12.2" is third release in December 2022.

We release new version every time there are any changes in `main` branch at the time release decision is made, as compared to last release. If there are no changes, month is skipped.

`PATCH` releases are reserved for extraordinary situations that warrant a release outside of usual schedule. These could be packaging issue that make new version uninstallable or very serious regressions. Decision about PATCH releases is made on case-by-case basis.

There are no long-term support versions.

Frontend and backend are developed independently and their version numbers may be different.

Users are advised to always use the newest versions of frontend and backend code available at the time of installation or upgrade. Configurations that use older version of one of the components are not supported. However, developers should make changes in backwards-compatible manner where possible.

CHAPTER

FOURTEEN

KUSTOSZ DOCUMENTATION

Focus on the worthwhile content with Kustosz, open source self-hosted web application.

Kustosz is feed reader that supports RSS, Atom and JSON Feed. What makes it unique?

- Continue reading where you left, on any device
- Automatically download full article content from source web page
- *Add any web page manually* (like Instapaper or Pocket)
- *Automatically hide duplicated content*
- Flexible and powerful *filters system* that allows you to automatically act on selected entries
- Clean, elegant and responsive web frontend
- Quick, *keyboard-driven frontend navigation*
- *Self-hosted* - you have full control over your data
- Open source - study, modify and use for any purpose (subject to license terms)

Interested? Head on to *Installation* page. We have single container image you can use to try Kustosz in safely isolated environment. Don't forget to perform *Initial setup*.