
Healthchecks



Healthchecks is a cron job monitoring service. It listens for HTTP requests and email messages (“pings”) from your cron jobs and scheduled tasks (“checks”). When a ping does not arrive on time, Healthchecks sends out alerts.

Healthchecks comes with a web dashboard, API, 25+ integrations for delivering notifications, monthly email reports, WebAuthn 2FA support, team management features: projects, team members, read-only access.

The building blocks are:

- Python 3.10+
- Django 5
- PostgreSQL or MySQL

Healthchecks is licensed under the BSD 3-clause license.

Healthchecks is available as a hosted service at <https://healthchecks.io/>.

A Dockerfile and pre-built Docker images are available.

Screenshots:

The “My Checks” screen. Shows the status of all your cron jobs in a live-updating dashboard.

Demo Project

CHECKS

INTEGRATIONS

BADGES

SETTINGS

Pricing

Docs

Blog

About

Account

db-backups

prod

sandbox

worker

Filter by check name...

Name	UUID <small>uuid slug</small>	Integrations	Period Grace	Last Ping Last Duration	
<div><div></div><div>/opt backups</div><div>sandbox</div></div>	ae69168c-2d1a-4d4a-babe-cf0f150320c9	<div><div></div></div>	12 hours 15 minutes	2 months, 1 week ago	...
<div><div></div><div>Product Sync</div><div>prod</div></div>	ffc143c9-6aea-44fa-b299-599739d8cb6d	<div><div></div></div>	5 minutes 20 minutes	5 minutes ago 22 sec	...
<div><div></div><div>DB Backups</div><div>prod</div><div>db-backups</div></div>	7c934798-b3f5-4fc4-a373-418ae184c899	<div><div></div></div>	1 day 1 hour	25 minutes ago	...
<div><div></div><div>Weekly Reports</div><div>prod</div><div>worker</div></div>	fe68e83b-aa2a-43a2-97d5-afbe607fa485	<div><div></div></div>	1 week 30 minutes	25 minutes ago	...
<div><div></div><div>Clean Uploads</div><div>sandbox</div></div>	1bed143e-5d06-4ec1-ab52-55e2621310b5	<div><div></div></div>	1 day 1 hour	7 months, 2 weeks ago	...

Full Screen

Each check has configurable Period and Grace Time parameters. Period is the expected time between pings. Grace Time specifies how long to wait before sending out alerts when a job is running late.

Demo Project

db-backups

prod

Name

/opt backu

sandbox

Product Sy

prod

DB Backu

prod

db-backups

Weekly Reports

prod

work

Clean Uploads

sandbox

Period

7

days

Grace Time

30

minutes

Period – The expected time between pings.

Grace Time – When a check is late, or has received a "start" signal, how long to wait to send an alert.

Simple

Cron

OnCalendar

Cancel

Save

Alternatively, you can define the expected schedules using a cron expressions. Healthchecks uses the

cronsim library to parse and evaluate cron expressions.

Demo Project

Cron Expression: 15 4 * * *
m h dom mon dow (cheatsheet)

Server's Time Zone: UTC

Grace Time: 30 minutes

"At 04:15 every day"

Expected Ping Dates

Date	Time	From Now	Timestamp
Dec 14, 04:15	19 hours from now	2023-12-14T04:15:00+00:00	
Dec 15, 04:15	1 day, 19 hours from now	2023-12-15T04:15:00+00:00	
Dec 16, 04:15	2 days, 19 hours from now	2023-12-16T04:15:00+00:00	
Dec 17, 04:15	3 days, 19 hours from now	2023-12-17T04:15:00+00:00	
Dec 18, 04:15	4 days, 19 hours from now	2023-12-18T04:15:00+00:00	
Dec 19, 04:15	5 days, 19 hours from now	2023-12-19T04:15:00+00:00	

Simple Cron OnCalendar Cancel Save

Check details page, with a live-updating event log.

Demo Project CHECKS INTEGRATIONS BADGES SETTINGS Pricing Docs Blog About Account

Product Sync (edit...)

prod

Description

Fetch product catalog from supplier, parse and import into our database. If this fails, the VM has likely ran out of memory.

Contact alice@example.org with any questions.

Edit Description...

How To Ping uuid slug

Keep this check up by making HTTP requests to this URL:

`https://hc-ping.com/ffc143c9-6aea-44fa-b299-599739d8cb6d`

Ping by sending email:

`ffc143c9-6aea-44fa-b299-599739d8cb6d@hc-ping.com`

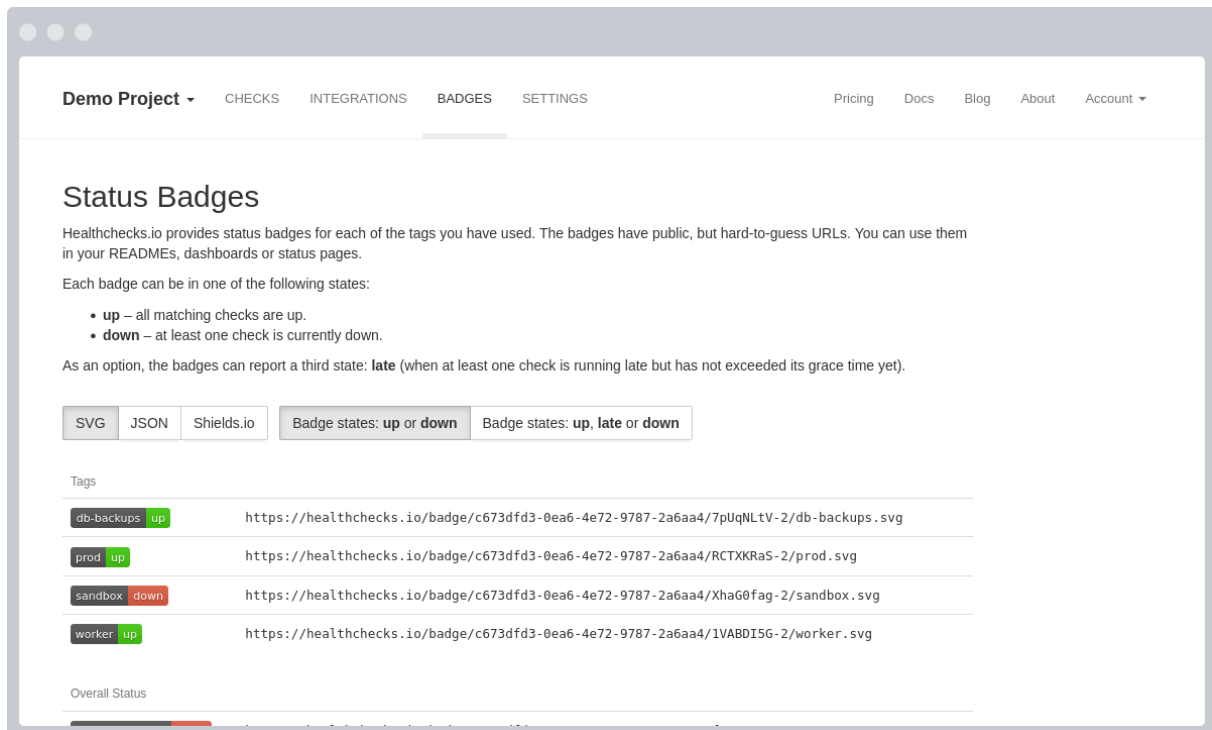
You can also explicitly [signal a failure](#) and [measure job execution time](#).

Filtering Rules... Usage Examples Copy URL

Events Click on individual items for details UTC Browser's time zone

ID	Date	Time	Status	Event
#608	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#607	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#606	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#605	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#604	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#603	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#602	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#601	Dec 13	10:35	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#600	Dec 13	10:34	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#599	Dec 13	10:34	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#598	Dec 13	10:34	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0
#597	Dec 13	10:34	OK	HTTPS GET from 2a03:b0c0:1:e0:53:a001 - curl/7.68.0

Healthchecks provides status badges with public but hard-to-guess URLs. You can use them in your READMEs, dashboards, or status pages.



Setting Up for Development

To set up Healthchecks development environment:

- Install dependencies (Debian/Ubuntu):

```
1 sudo apt update
2 sudo apt install -y gcc python3-dev python3-venv libpq-dev
  libcurl4-openssl-dev libssl-dev
```

- Prepare directory for project code and virtualenv. Feel free to use a different location:

```
1 mkdir -p ~/webapps
2 cd ~/webapps
```

- Prepare virtual environment (with virtualenv you get pip, we'll use it soon to install requirements):

```
1 python3 -m venv hc-venv
2 source hc-venv/bin/activate
3 pip3 install wheel # make sure wheel is installed in the venv
```

-
- Check out project code:

```
1 git clone https://github.com/healthchecks/healthchecks.git
```

- Install requirements (Django, ...) into virtualenv:

```
1 pip install -r healthchecks/requirements.txt
```

- macOS only - pycurl needs to be reinstalled using the following method (assumes OpenSSL was installed using brew):

```
1 export PYCURL_VERSION=`cat requirements.txt | grep pycurl | cut -d  
  '=' -f3`  
2 export OPENSSL_LOCATION=`brew --prefix openssl`  
3 export PYCURL_SSL_LIBRARY=openssl  
4 export LDFLAGS=-L$OPENSSL_LOCATION/lib  
5 export CPPFLAGS=-I$OPENSSL_LOCATION/include  
6 pip uninstall -y pycurl  
7 pip install pycurl==$PYCURL_VERSION --compile --no-cache-dir
```

- Create database tables and a superuser account:

```
1 cd ~/webapps/healthchecks  
2 ./manage.py migrate  
3 ./manage.py createsuperuser
```

With the default configuration, Healthchecks stores data in a SQLite file `hc.sqlite` in the checkout directory (`~/webapps/healthchecks`).

- Run tests:

```
1 ./manage.py test
```

- Run development server:

```
1 ./manage.py runserver
```

The site should now be running at <http://localhost:8000>. To access Django administration site, log in as a superuser, then visit <http://localhost:8000/admin/>

Configuration

Healthchecks reads configuration from environment variables.

Full list of configuration parameters.

Accessing Administration Panel

Healthchecks comes with Django's administration panel where you can manually view and modify user accounts, projects, checks, integrations etc. To access it,

- if you haven't already, create a superuser account: `./manage.py createsuperuser`
- log into the site using superuser credentials
- in the top navigation, "Account" dropdown, select "Site Administration"

Sending Emails

Healthchecks must be able to send email messages, so it can send out login links and alerts to users. Specify your SMTP credentials using the following environment variables:

- Implicit TLS (*recommended*): `python DEFAULT_FROM_EMAIL = "valid-sender-address@example.org"EMAIL_HOST = "your-smtp-server-here.com"EMAIL_PORT = 465 EMAIL_HOST_USER = "smtp-username"EMAIL_HOST_PASSWORD = "smtp-password"EMAIL_USE_TLS = False EMAIL_USE_SSL = True`

Port 465 should be the preferred method according to RFC8314 Section 3.3: Implicit TLS for SMTP Submission. Be sure to use a TLS certificate and not an SSL one.

- Explicit TLS: `python DEFAULT_FROM_EMAIL = "valid-sender-address@example.org"EMAIL_HOST = "your-smtp-server-here.com"EMAIL_PORT = 587EMAIL_HOST_USER = "smtp-username"EMAIL_HOST_PASSWORD = "smtp-password"EMAIL_USE_TLS = True`

For more information, have a look at Django documentation, Sending Email section.

Receiving Emails

Healthchecks comes with a `smtpd` management command, which starts up a SMTP listener service. With the command running, you can ping your checks by sending email messages to `your-uuid-here@my-monitoring-project.com` email addresses.

Start the SMTP listener on port 2525:

```
1 ./manage.py smtpd --port 2525
```

Send a test email:

```
1 curl --url 'smtp://127.0.0.1:2525' \  
2     --mail-from 'foo@example.org' \  
3     --mail-rcpt '11111111-1111-1111-1111-111111111111@my-monitoring-  
4         project.com' \  
5     -F '='
```

Sending Alerts and Reports

Healthchecks comes with a `sendalerts` management command, which continuously polls database for any checks changing state, and sends out notifications as needed. Within an activated virtualenv, you can manually run the `sendalerts` command like so:

```
1 ./manage.py sendalerts
```

In a production setup, you will want to run this command from a process manager like systemd or supervisor.

Healthchecks also comes with a `sendreports` management command which sends out monthly reports, weekly reports, and the daily or hourly reminders.

Run `sendreports` without arguments to run any due reports and reminders and then exit:

```
1 ./manage.py sendreports
```

Run it with the `--loop` argument to make it run continuously:

```
1 ./manage.py sendreports --loop
```

Database Cleanup

Healthchecks deletes old entries from `api_ping` and `api_notification` tables automatically. By default, Healthchecks keeps the 100 most recent pings for every check. You can set the limit higher to keep a longer history: go to the Administration Panel, look up user's **Profile** and modify its "Ping log limit" field.

For each check, Healthchecks removes notifications that are older than the oldest stored ping for same check.

Healthchecks also provides management commands for cleaning up `auth_user`, `api_tokenbucket` and `api_flip` tables.

- Remove user accounts that match either of these conditions:

-
- Account was created more than 6 months ago, and user has never logged in. These can happen when user enters invalid email address when signing up.
 - Last login was more than 6 months ago, and the account has no checks. Assume the user doesn't intend to use the account any more and would probably *want* it removed.

```
1 ./manage.py pruneusers
```

- Remove old records from the `api_tokenbucket` table. The TokenBucket model is used for rate-limiting login attempts and similar operations. Any records older than one day can be safely removed.

```
1 ./manage.py prunetokenbucket
```

- Remove old records from the `api_flip` table. The Flip objects are used to track status changes of checks, and to calculate downtime statistics month by month. Flip objects from more than 3 months ago are not used and can be safely removed.

```
1 ./manage.py pruneflips
```

- Remove old objects from external object storage. When an user removes a check, removes a project, or closes their account, Healthchecks does not remove the associated objects from the external object storage on the fly. Instead, you should run `pruneobjects` occasionally (for example, once a month). This command first takes an inventory of all checks in the database, and then iterates over top-level keys in the object storage bucket, and deletes any that don't also exist in the database.

```
1 ./manage.py pruneobjects
```

When you first try these commands on your data, it is a good idea to test them on a copy of your database, not on the live database right away. In a production setup, you should also have regular, automated database backups set up.

Two-factor Authentication

Healthchecks optionally supports two-factor authentication using the WebAuthn standard. To enable WebAuthn support, set the `RP_ID` (relying party identifier) setting to a non-null value. Set its value to your site's domain without scheme and without port. For example, if your site runs on `https://my-hc.example.org`, set `RP_ID` to `my-hc.example.org`.

Note that WebAuthn requires HTTPS, even if running on localhost. To test WebAuthn locally with a self-signed certificate, you can use the `runsslserver` command from the `django-sslserver` package.

External Authentication

Healthchecks supports external authentication by means of HTTP headers set by reverse proxies or the WSGI server. This allows you to integrate it into your existing authentication system (e.g., LDAP or OAuth) via an authenticating proxy. When this option is enabled, **healthchecks will trust the header's value implicitly**, so it is **very important** to ensure that attackers cannot set the value themselves (and thus impersonate any user). How to do this varies by your chosen proxy, but generally involves configuring it to strip out headers that normalize to the same name as the chosen identity header.

To enable this feature, set the `REMOTE_USER_HEADER` value to a header you wish to authenticate with. HTTP headers will be prefixed with `HTTP_` and have any dashes converted to underscores. Headers without that prefix can be set by the WSGI server itself only, which is more secure.

When `REMOTE_USER_HEADER` is set, Healthchecks will: - assume the header contains user's email address - look up and automatically log in the user with a matching email address - automatically create an user account if it does not exist - disable the default authentication methods (login link to email, password)

External Object Storage

Healthchecks can optionally store large ping bodies in S3-compatible object storage. To enable this feature, you will need to:

- ensure you have the MinIO Python library installed:

```
1 pip install minio
```

- configure the credentials for accessing object storage: `S3_ACCESS_KEY`, `S3_SECRET_KEY`, `S3_ENDPOINT`, `S3_REGION` and `S3_BUCKET`.

Healthchecks will use external object storage for storing any request bodies that exceed 100 bytes. If the size of a request body is 100 bytes or below, Healthchecks will still store it in the database.

Healthchecks automatically removes old stored ping bodies from object storage while uploading new data. However, Healthchecks does not automatically clean up data when you delete checks, projects or entire user accounts. Use the `pruneobjects` management command to remove data for checks that don't exist any more.

When external object storage is not enabled (the credentials for accessing object storage are not set), Healthchecks stores all ping bodies in the database. If you enable external object storage, Healthchecks will still be able to access the ping bodies already stored in the database. You don't

need to migrate them to the object storage. On the other hand, if you later decide to disable external object storage, Healthchecks will not have access to the externally stored ping bodies any more. And there is currently no script or management command for migrating ping bodies from external object storage back to the database.

Integrations

Slack

Healthchecks supports two Slack integration setup flows: legacy and app-based.

The legacy flow does not require additional configuration and is used by default. In this flow the user creates an incoming webhook URL on the Slack side, and pastes the webhook URL in a form on the Healthchecks side.

In the app-based flow the user clicks an “Add to Slack” button in Healthchecks, and gets transferred to a Slack-hosted dialog where they select the channel to post notifications to. This flow uses OAuth2 behind the scenes. To enable this flow, you will need to set up a Slack OAuth2 app:

- Create a new Slack app on <https://api.slack.com/apps/>
- Add at least one scope in the permissions section to be able to deploy the app in your workspace (By example `incoming-webhook` for the `Bot Token Scopes`).
- Add a *redirect url* in the format `SITE_ROOT/integrations/add_slack_btn/`. For example, if your `SITE_ROOT` is `https://my-hc.example.org` then the redirect URL would be `https://my-hc.example.org/integrations/add_slack_btn/`.
- Look up your Slack app for the Client ID and Client Secret. Put them in `SLACK_CLIENT_ID` and `SLACK_CLIENT_SECRET` environment variables. Once these variables are set, Healthchecks will switch from using the legacy flow to using the app-based flow.

The legacy and app-based flows only affect the user experience during the initial setup of Slack integrations. The contents of notifications posted to Slack are the same regardless of the setup flow used.

Discord

To enable Discord integration, you will need to:

- register a new application on <https://discord.com/developers/applications/me>

-
- add a redirect URI to your Discord application. The URI format is `SITE_ROOT/integrations/add_discord/`. For example, if you are running a development server on `localhost:8000` then the redirect URI would be `http://localhost:8000/integrations/add_discord/`
 - Look up your Discord app's Client ID and Client Secret. Put them in `DISCORD_CLIENT_ID` and `DISCORD_CLIENT_SECRET` environment variables.

Pushover

Pushover integration works by creating an application on Pushover.net which is then subscribed to by Healthchecks users. The registration workflow is as follows:

- On Healthchecks, the user adds a "Pushover" integration to a project
- Healthchecks redirects user's browser to a Pushover.net subscription page
- User approves adding the Healthchecks subscription to their Pushover account
- Pushover.net HTTP redirects back to Healthchecks with a subscription token
- Healthchecks saves the subscription token and uses it for sending Pushover notifications

To enable the Pushover integration, you will need to:

- Register a new application on Pushover via `https://pushover.net/apps/build`.
- Within the Pushover 'application' configuration, enable subscriptions. Make sure the subscription type is set to "URL". Also make sure the redirect URL is configured to point back to the root of the Healthchecks instance (e.g., `http://healthchecks.example.com/`).
- Put the Pushover application API Token and the Pushover subscription URL in `PUSHOVER_API_TOKEN` and `PUSHOVER_SUBSCRIPTION_URL` environment variables. The Pushover subscription URL should look similar to `https://pushover.net/subscribe/yourAppName-randomAlphaNumericData`.

Signal

Healthchecks uses signal-cli to send Signal notifications. Healthchecks interacts with signal-cli over UNIX or TCP socket. Healthchecks requires signal-cli version 0.11.2 or later.

To enable the Signal integration via UNIX socket:

- Set up and configure signal-cli to expose JSON RPC on an UNIX socket (instructions). Example: `signal-cli -a +xxxxxx daemon --socket /tmp/signal-cli-socket`
- Put the socket's location in the `SIGNAL_CLI_SOCKET` environment variable.

To enable the Signal integration via TCP socket:

- Set up and configure `signal-cli` to expose JSON RPC on a TCP socket. Example: `signal-cli -a +xxxxxx daemon --tcp 127.0.0.1:7583`
- Put the socket's hostname and port in the `SIGNAL_CLI_SOCKET` environment variable using "hostname:port" syntax, example: `127.0.0.1:7583`.

Telegram

- Create a Telegram bot by talking to the BotFather. Set the bot's name, description, user picture, and add a `/start` command. To avoid user confusion, please do not use the Healthchecks.io logo as your bot's user picture, use your own logo.
- After creating the bot you will have the bot's name and token. Put them in `TELEGRAM_BOT_NAME` and `TELEGRAM_TOKEN` environment variables.
- Run `settelegramwebhook` management command. This command tells Telegram where to forward channel messages by invoking Telegram's `setWebhook` API call:

```
1 ./manage.py settelegramwebhook
2 Done, Telegram's webhook set to: https://my-monitoring-project.com/integrations/telegram/bot/
```

For this to work, your `SITE_ROOT` must be correct and must use the `"https://"` scheme.

Apprise

To enable Apprise integration, you will need to:

- ensure you have `apprise` installed in your local environment:

```
1 pip install apprise
```

- enable the `apprise` functionality by setting the `APPRISE_ENABLED` environment variable.

Shell Commands

The "Shell Commands" integration runs user-defined local shell commands when checks go up or down. This integration is disabled by default, and can be enabled by setting the `SHELL_ENABLED` environment variable to `True`.

Note: be careful when using “Shell Commands” integration, and only enable it when you fully trust the users of your Healthchecks instance. The commands will be executed by the `manage.py sendalerts` process, and will run with the same system permissions as the `sendalerts` process.

Matrix

To enable the Matrix integration you will need to:

- Register a bot user (for posting notifications) in your preferred homeserver.
- Use the Login API call to retrieve bot user’s access token. You can run it as shown in the documentation, using curl in command shell.
- Set the `MATRIX_` environment variables. Example:

```
1 MATRIX_HOMESERVER=https://matrix.org
2 MATRIX_USER_ID=@mychecks:matrix.org
3 MATRIX_ACCESS_TOKEN=[a long string of characters returned by the login
  call]
```

PagerDuty Simple Install Flow

To enable PagerDuty Simple Install Flow,

- Register a PagerDuty app at PagerDuty › Developer Mode › My Apps
- In the newly created app, add the “Events Integration” functionality
- Specify a Redirect URL: `https://your-domain.com/integrations/add_pagerduty/`
- Copy the displayed `app_id` value (PXXXXX) and put it in the `PD_APP_ID` environment variable

Running in Production

Here is a non-exhaustive list of pointers and things to check before launching a Healthchecks instance in production.

- Environment variables, `settings.py` and `local_settings.py`.
 - `DEBUG`. Make sure it is set to `False`.
 - `ALLOWED_HOSTS`. Make sure it contains the correct domain name you want to use.

-
- Server Errors. When `DEBUG=False`, Django will not show detailed error pages, and will not print exception tracebacks to standard output. To receive exception tracebacks in email, review and edit the `ADMINS` and `SERVER_EMAIL` settings. Consider setting up exception logging with Sentry.
 - Management commands that need to be run during each deployment.
 - `manage.py compress` – creates combined JS and CSS bundles and places them in the `static-collected` directory.
 - `manage.py collectstatic` – collects static files in the `static-collected` directory.
 - `manage.py migrate` – applies any pending database schema changes and data migrations.
 - Processes that need to be running constantly.
 - `manage.py runserver` is intended for development only. **Do not use it in production**, instead consider using uWSGI or gunicorn.
 - `manage.py sendalerts` is the process that monitors checks and sends out monitoring alerts. It must be always running, it must be started on reboot, and it must be restarted if it itself crashes. On modern linux systems, a good option is to define a systemd service for it.
 - Static files. Healthchecks serves static files on its own, no configuration required. It uses the Whitenoise library for this.
 - General
 - Make sure the database is secured well and is getting backed up regularly
 - Make sure the TLS certificates are secured well and are getting refreshed regularly
 - Have monitoring in place to be sure the Healthchecks instance itself is operational (is accepting pings, is sending out alerts, is not running out of resources).

Docker Image

Healthchecks provides a reference Dockerfile and prebuilt Docker images for every release. The Dockerfile lives in the `/docker/` directory, and Docker images for amd64, arm/v7 and arm64 architectures are available on Docker Hub.

The Docker images:

- Use uWSGI as the web server. uWSGI is configured to perform database migrations on startup, and to run `sendalerts`, `sendreports`, and `smtpd` in the background. You do not need to run them separately.

-
- Ship with both PostgreSQL and MySQL database drivers.
 - Serve static files using the whitenoise library.
 - Have the apprise library preinstalled.
 - Do *not* handle TLS termination. In a production setup, you will want to put the Healthchecks container behind a reverse proxy or load balancer that handles TLS termination.