OpenAppStack

Release 0.4.0

Jul 02, 2020

Contents:

OpenAppStack (OAS) is a platform that will offer self-managed, click-and-play provisioning of online applications for Civil Society Organisations (CSOs). Users will be able to easily set up a self-hosted instance of OpenAppStack, so they can keep control over the data that gets entered into these applications.

OpenAppStack is:

- Open Source
- Self updating
- Easy to deploy
- Integrated

For more information, go to the OpenAppStack website.

CHAPTER 1

OpenAppStack installation instructions

This document describes how you can install OpenAppStack on a VPS. The installation process wil set up a "Kubernetes cluster" which runs several open source applications. More information about the applications can be found on the OpenAppStack website.

We will set up a "single-node" OpenAppStack cluster. This means everything runs on the same VPS. Support for "multi-node" clusters (a Kubernetes cluster on more than one VPS) will come in the future.

Note: All commands in these installation instructions need to be run on a trusted provisioning machine (i.e., your laptop) that is *not* the VPS that will run OpenAppStack. The installation process will generate some secrets that will be saved to this machine.

If you encounter any difficulties while following these instructions, please [open an issue following our contact guide][https://openappstack.net/contact.html).

Warning:

- OpenAppStack is still under heavy development and is not ready for production use! We anticipate major changes and do not guarantee a data-preserving upgrade path from current installations. However, we encourage you to try OpenAppStack and ask you to report all issues you encounter.
- When you install OpenAppStack on a server, the installation process will make some substantial changes to the server's configuration, so please do not use a server that functions as anything other than a testing ground.

1.1 Prerequisites

- During these instructions, you are asked to create a VPS, or have a bare metal server ready. The server should meet these requirements:
- Current Debian stable "buster"

- A public IP address
- The ability to create DNS records for this IP
- 8GB of RAM
- At least 25GB of disk space for installation, plus more for application data. We recommend starting with 30GB.
- Root ssh access
- `python3` installed. On a debian based system:

  `apt install python3`
- A trusted machine to run the installer on (we call this the `provisioning machine`). All the commands listed in these instructions should be run on the provisioning machine, unless specified otherwise.
- You need Python 3 with its development files, Pip and Git installed (`apt install python3-pip python3-dev git`)
- We recommend using a python virtualenv to make sure we do not change any of your other projects. Install virtualenv by running `pip3 install --user venv` or `apt install python3-venv`
- `ssh-agent` to give you access to your VPS

1.2 Getting the installation script

On your **provisioning machine**, clone the OpenAppStack git repository and checkout the latest tagged version (currently `0.4.0`):

```
$ git clone -b 0.4.0 https://open.greenhost.net/openappstack/openappstack.git
$ cd openappstack
```

Note: Git will display a warning after you use the `git` command saying that you are in a *detached HEAD* state. This is perfectly normal and you can proceed without any further actions.

Create a python virtual environment called "env" that uses python 3. This makes sure we do not change any of your other python projects. The second command "activates" the virtualenv.

Note: Activating the virtualenv means you will use that environment to install and run python programs instead of your global environment. If you close your terminal or open a new one, you need to activate the virtualenv again.

```
$ python3 -m venv env
$ . env/bin/activate
```

Next, install the OpenAppStack CLI client by running the following commands:

```
$ pip3 install -r requirements-stable.txt
```

Now you can run the OpenAppStack CLI as follows:

```
$ python -m openappstack CLUSTER_NAME <command>
```

The CLI *always* needs a `CLUSTER_NAME` argument. Even for getting subcommand help messages. Be sure to run this command in the root directory of the git repository. In this tutorial, we're using `my-cluster` as the cluster name. Try it out by running

```
$ python -m openappstack my-cluster --help
```

1.3 Install OpenAppStack

Setting up OpenAppStack happens in four steps:

1. **Set up cluster**

   Create configuration files, and optionally create VPS.

2. **Optional: enable backups using Velero**

   Optionally add settings and credentials to enable backups of your cluster to external S3 storage.

3. **Install OpenAppStack**

   Install Kubernetes and all the other software that comes with OpenAppStack. See *Usage* for more information on which applications are installed.

4. **Validate setup**

   This runs a test in the browser to validate that the installation was successful.

1.3.1 Step 1: Set up cluster

To set up your cluster, use the `create` subcommand of the OpenAppStack CLI. First, choose a name (we chose `my-cluster`) for your cluster. Then run the following command to get information about the `create` subcommand:

```
$ python -m openappstack my-cluster create --help
```

There are two options to create a cluster, *using the Greenhost API* or *using any other machine*. Choose one of these options, and skip to the "*Installation*" subchapter afterwards.

Option A: Setup with the Greenhost API

- Before you can start, you need to have an API key with Customer rights.
  1. In the Cosmos service centre, click your webmaster account name on the top right corner
  2. Go to "User settings"
  3. Click "API keys"
  4. Click "New API key"
  5. Click "Generate new key"
  6. Give the key "Customer", "CloudCustomer" or "API" access rights. You will need "Customer" rights if you want to automatically generate DNS rules. If you do not have this right, you have to manually set the right DNS rules later.
  7. Copy the generated key and run export it to this variable in a terminal:

     ```
     $ export COSMOS_API_TOKEN=<paste your API key here>
     ```

  8. In *the same terminal*, you can now use the `create` subcommand
- There are two ways to let the installation program know which VPS to use:

1. Based on an already existing Greenhost VPS, using the `--droplet-id` argument.

   Find the ID of your VPS either in the Greenhost Cosmos interface (it is the numeric part of the URL in the "Manage VPS" screen).

2. By creating a new VPS through the API, using the `--create-droplet` argument.

   In that case, make sure to also provide the `--create-hostname` and `--ssh-key-id` arguments.

   You can find your SSH key ID by going to VPS Cloud -> SSH keys and checking the link under "Show key". The numerical part is your SSH key ID.

   Note: You can also use the API to list ssh keys and find it there. Read the 'Greenhost API documentation <https://service.greenhost.net/cloud/ApiDoc#/default>'__ for more information

- In both cases you need to provide the `DOMAIN_NAME` positional argument.

  If you use a subdomain (e.g. `oas.yourdomain.com`), use the `--subdomain` command as follows:

```
$ python -m openappstack my-cluster create --subdomain oas example.org
```

- Here is an example of a complete creation command:

```
$ python -m openappstack my-cluster create \
  --create-droplet \
  --create-hostname oas.example.org \
  --ssh-key-id 112 \
  --create-domain-records \
  --subdomain oas \
  example.org
```

Note: You can use the `--acme-staging` argument for testing purposes. This ensures you use "staging" certificates from Let's Encrypt, to reduce the stress on their servers. However, ONLYOFFICE integration requires valid (live) certificates to work.

This will create configuration files for a cluster named `my-cluster`. It will also create a Greenhost VPS with the hostname `oas.example.org` and on which you can log in with SSH key with ID `112`.

These DNS records will also be created by Greenhost (assuming you own the domain `example.org` at Greenhost):

- An `A` record `oas.example.org` pointing to the VPSs IP address
- A `CNAME` record `*.oas.example.org` pointing to `oas.example.org`.

Option B: Setup any VPS or bare metal server

Skip this step and continue to *Installation* if you already set up a Greenhost VPS.

If you want to install OpenAppStack on a non-Greenhost VPS, we assume you already have a machine with a world-facing IP address. Make sure that your VPS meets our *prerequisites*. You'll need its *hostname* and its *IP address*.

Note: You can use the `--acme-staging` argument for testing purposes. If you are automating this, please use this to ensure you use "staging" certificates from Let's Encrypt, to reduce the stress on their servers. However, ONLYOFFICE and single sign-on integration require valid (live) certificates to work properly so please don't use this option by default.

If you want your cluster to be reachable under the fully qualified domain name (FQDN) `oas.example.org`, the corresponding parameters would be:

- `--subdomain`: `oas`
- the `DOMAIN` positional argument: `example.org`

Create the OpenAppStack settings for your VPS by running the following command:

```
$ python -m openappstack my-cluster create \
  --ip-address IP_ADDRESS \
  --create-hostname my-clusters-hostname \
  --subdomain oas \
  example.org
```

DNS entries

Before you continue, if you have not made DNS entries with the CLI tool, you need to make them now. It is important to start with configuring DNS because depending on your DNS setup/provider, it takes a while (sometimes hours) to propagate.

You need one dedicated (sub)domain entry and a wildcard entry for everything inside it. For example, create an A record for these domains:

- An `A` record `oas.example.org` pointing to the VPSs IP address,
- A `CNAME` record `*.oas.example.org` pointing to `oas.example.org`.

Note: It is also possible to host OpenAppStack on a domain (with no dedicated subdomain). That does imply that the included Wordpress site will be hosted on your root domain `example.org`. In that case, make these DNS records instead:

- An `A` record `example.org` pointing to the VPSs IP address,
- A `CNAME` record `*.example.org` pointing to `example.org`.

OpenAppStack will fetch https certificates with Let's Encrypt by default. In order to do this DNS entries need to be created. If you don't need https certificates for your cluster while testing you can skip this step. Please be aware of the limitations of this:

- Onlyoffice won't work since it requires a valid certificate connecting to Nextcloud.
- You need to be able to resolve the domain names locally.

1.3.2 Step 2: Optional: cluster backups using Velero

At this point you can enable Velero, a program that runs on your cluster and uploads backups of your cluster and user data to an S3 storage service of your choice.

If enabled, Velero will create a backup of your cluster once every night and upload it to the S3 storage you configure. This includes:

- your cluster state. Technically speaking, it will back up all Kubernetes namespaces in your cluster, except `velero` itself; this includes things like which applications are installed, including their version number and installation-time settings;

- persistent data of all applications: for example, single sign-on users that you created, Nextcloud files and metadata, Wordpress site data and comments, Rocketchat chat history, etc. A single exception to this is Prometheus data (statistics of system properties), which takes up a lot of space and we consider not valuable enough to back up.

It does not include anything on the VPS that you may have set up but is not part of OpenAppStack, like programs installed via `apt`, or data added to the VPS disk not through OpenAppStack.

To enable Velero:

- edit the file `clusters/my-cluster/group_vars/all/settings.yml`, reviewing all settings under the `backup` field and entering the data specific to your backup storage location;
- in that same file `clusters/my-cluster/group_vars/all/settings.yml`, add `'velero'` to the list of `enabled_applications`;
- create the directory `clusters/my-cluster/secrets`;
- create the file `clusters/my-cluster/secrets/s3_access_key`, with as only contents your S3 access key;
- create the file `clusters/my-cluster/secrets/s3_secret_key`, with as only contents your S3 secret key.

Then continue with the installation procedure as described below.

1.3.3 Step 3: Installation

You're almost ready to start the OpenAppStack installation script. First, make sure your DNS configuration is propagated. To do so, make sure 'ping' shows your VPS's IP address:

```
$ ping oas.example.org
```

The `install` command will try to log into your machine as the `root`user using SSH.

To start the installation process, run:

```
$ python -m openappstack my-cluster install
```

This will take between 5 and 20 minutes. It generates secrets that will be added to the `clusters/my-cluster/secrets` directory. If you ever need any credentials after this installation, you can probably find them there. **Make sure this directory stays safe.** Feel free to encrypt it when you are not using the `openappstack` CLI tool.

OpenAppStack uses Flux to install applications. After the installation process has compleded, Flux has not necessarily finished installing and integrating all the applications. This process usually takes 10-20 minutes to complete, but can also take longer depending on your machine's compute and/or network resources.

In the future, we will add commands that show you the status of the application installation. For now, just wait half an hour for everything to settle, and then continue to the next step (validating your setup).

Note: It is possible to re-run the `install` command. Make sure you re-run it on a machine with the same `secrets` as generated the first time. You can achieve this by making sure you have the `clusters/my-cluster` directory and it contains the same `secrets` directory before you run the installation command.

1.3.4 Step 4: Validate setup

Because OpenAppStack is still under development, we would like you to follow our testing instructions to make sure that the setup process went well.

CHAPTER 2

A testing guide and test reporting form for testers of the OpenAppStack project

Great that you want to take OpenAppStack for a test drive. This tutorial contains instructions to get you going, some pointers on what we think would be useful to test, and guesses at what results of those tests would be useful to write down. At any point please feel invited to test whatever functionality you come across, and reporting whatever you think is interesting. Our contact details are listed here, and we'll descibe how to give feedback via our issue tracker at the *end of these instructions*.

During these instructions, please replace *example.org* with your own domain name.

2.1 Installation

First we'd like you to setup an OpenAppStack cluster by yourself, following the installation tutorial.

2.2 Command line tests

Please run the *command line tests* which checks the overall functionality of your cluster and include the output in your feedback.

2.3 Usage

Please go through the *Usage documentation* and make sure you complete all steps.

2.4 Nextcloud

2.4.1 Logging into Nextcloud

Please browse to https://files.oas.example.org and try to log in using single sign-on. Use the button labeled `Login with OpenAppStack`. Please try logging in with your admin account and configure the email settings as shown in the Usage doc. After that please login with the user you created in the user panel.

2.4.2 Nextcloud client application

- If you feel like it, please try the Nextcloud desktop client, connect it to your OpenAppStack instance, and use it to manage some files.
- If you feel like it, please try the Nextcloud mobile client for your smartphone, connect it to your OpenAppStack instance, and use it to download and/or open some files, upload a new file, etc.

2.5 Onlyoffice

2.5.1 Creating a new office document

From the main Nextcloud webpage, please try to create a new office document, by clicking the round `plus` button near the top of the screen, then picking the `Document` type with the blue icon (third one from below on my screen), and enter a name for it. After that, please try some basic editing of the document, and save it. Maybe check you can open it again afterwards, and that it has the contents that you saved earlier.

2.5.2 Collaborating on an office document

This part of the test requires the cooperation of another person; feel free to skip it now if that's not convenient at this point.

- First, try to share your document with a different user.
- Then, try to open the shared document from a few different user accounts simultaneously, and let all participants edit the document mercilessly. There are also some collaboration features that you may want to try: on the left of the Onlyoffice screen there are buttons for chat and for text comments.

2.6 Rocketchat

You can find Rocketchat at https://chat.oas.example.org. Once you configured Rocketchat for single sign-on as desribed in the Usage docs please login using single sign-on as `admin` and afterwards as the user you created.

2.7 Wordpress

You can find Wordpress at https://www.oas.example.org. Please try to login as the new user you created earlier by pressing "Log in" and using the `Login with OpenID Connect` button.

At the moment Administrator privileges will not be available for single sign-on users of WordPress. You can sign in with the automatically created administrator account. The username is `admin` and the password can be found in the `wordpress_admin_password` file in the `secrets` folder of your provisioning machine's config directory.

2.8 Providing feedback

If you have not done already, please create an account on https://open.greenhost.net (or login with your existing github account) and create a new issue using the `Feedback` template.

Thanks a lot for your testing work! We'll use your input to try to improve OpenAppStack.

CHAPTER 3

Usage

After all the applications are installed, the first thing to do is log into https://admin.oas.example.org. Here you can find the "user panel", a place where you can create, edit and delete users. You can log in with the user "admin". The password can be found in `clusters/my-cluster/secrets/userbackend_admin_password`. After logging in, you will see an overview of all the applications your user has access to. For more information on how to create users and give them access to applications, take a look at the user panel documentation.

> **NOTE:** at the moment none of the applications are available at `oas.example.org`, we only provide applications in subdomains. In the future this might change.

3.1 Applications

These applications are available after the installation is completed successfully:

3.1.1 OAS User panel

The OAS user panel can be used to create and edit users. These users can be used to log into the applications listed below. The user panel is available at https://admin.oas.example.org. You can login as `admin` using the `userbackend_admin_password` password from your secrets folder.

After logging in to the user panel follow the user panel documentation to create a new user.

Note: The email address is important because some applications need a valid email address for notification mails. Single sign-on with Grafana will fail for users lacking an email address.

You can now use the new user to log in to all apps which were granted access to in the last step using single sign-on.

3.1.2 Nextcloud

Nextcloud is a file sharing and communication platform and is available at https://files.oas.example.org.

Single sign-on

Nextcloud needs to be configured to properly send out emails. You can do so by logging in as `admin` using signle sign-on and then going to `Settings -> Basic settings -> Email server` and entering your SMTP email config and credentials. Please complete this configuration before you login as non-admin user using single sign-on, otherwise the first login will not succeed.

3.1.3 Onlyoffice

Onlyoffice is an online document editing suite. Your can open documents in Onlyoffice by clicking them in Nextcloud. You can open new documents by clicking the "Plus" button in Nextcloud and selecting Document, Spreadsheet or Presentation.

3.1.4 Rocketchat

Rocketchat is a team chat application and available at https://chat.oas.example.org.

Single sign-on

Until we fully automate SSO integration for Rocketchat manual intervention is neccessary to activate it. You need to follow these steps once:

- Log in as `admin` using the `rocketchat_admin_password` from your secrets folder.
- On the top left side click on the `Options` button (three dots) and then click on `Administration`
- In the left menu scroll down and click on `OAuth` (not `oauth apps`)
- Click on `add custom oauth` and enter `Openappstack`
- Click on the newly added `Custom OAuth: Openappstack` provider
- Change the following settings (leave all others like they are):
  - Enable: `True`
  - URL: `https://sso.oas.example.org` (change `oas.example.org` to your domain)
  - Token Path: `/oauth2/token`
  - Identity Path: `/userinfo`
  - Authorize Path: `/oauth2/auth`
  - Scope: `openid profile openappstack_roles email`
  - Id: `rocketchat`
  - Secret: Paste the `rocketchat_oauth_client_secret` from your secrets folder
  - Login Style: `Redirect`
  - Button Text: `Login with OpenAppStack`
  - Username field: `preferred_username`
  - Name field: `name`
  - Roles/Groups field name: `openappstack_roles`
  - Merge roles from SSO: `True`

  - Merge users: `True`

- Click `Save changes`, log out and you are done.

Next time you log in to Rocketchat you will be able to use single sign-on using the `Login` button.

3.1.5 Known issues

- Rocketchat isn't configured yet to send out email notifications

3.1.6 Wordpress

Wordpress is a website content management system and available at https://www.oas.example.org. Click the `Log in` button and then click `Login with OpenID Connect` to use single sign-on.

Single sign-on

- If you log in as `admin` using single sign-on, you will not have admin rights within Wordpress. In order to use admin rights you need to log in without single sign-on using the `wordpress_admin_password` password in the `secrets` folder.

3.1.7 Grafana

Grafana that shows you information about the status of your cluster. Read more about Grafana in the *monitoring chapter below*

Single sign-on

- If you log in as `admin` using single sign-on, you will not have admin rights within Grafana. In order to use admin rights you need to log in without signgle sign-on using the `grafana_admin_password` password in the `secrets` folder.

3.1.8 Other applications installed into the cluster

Besides these applications, some other components are installed. These are part of the OpenAppStack back end and they dont't have a user facing web interfaces, but we like to list them here for reference:

- rancher's local-path-provisioner provides an easy way for the cluster to use a directory on the node (by default `/var/lib/OpenAppStack/local-storage`) for storage;
- NGINX is a webserver that functions as a so-called ingress controller, routing web traffic that enters the cluster to the various applications;
- cert-manager acquires and stores Let's Encrypt certificates, enabling encrypted web traffic to all applications running in the cluster;
- Flux checks for application updates approved by the OpenAppStack team and installs them automatically.

CHAPTER 4

Troubleshooting

If you run into problems, there are a few things you can do to research the problem. This document describes what you can do.

NOTE: `cluster$` indicates that the commands should be run as root on your OAS machine.

4.1 Known issues

Take a look if the problem you have encountered is already in our issue tracker.

4.2 Run the cli tests

To get an overall status of your cluster you can run the tests from the command line.

There are two types of tests: testinfra tests, and behave tests.

4.2.1 Testinfra tests

Testinfra tests are split into two groups, lets call them `blackbox` and `clearbox` tests. The blackbox tests run on your provisioning machine and test the OAS cluster from the outside. For example, the certificate check will check if the OAS will return valid certificates for the provided services. The clearbox tests run on the OAS host and check i.e. if docker is installed in the right version etc.

First, enter the `test` directory in the Git repository on your provisioning machine.

To run the test against your cluster, first export the `CLUSTER_DIR` environment variabel with the location of your cluster config directory:

```
export CLUSTER_DIR="../clusters/CLUSTERNAME"
```

Run all tests:

```
py.test -s --ansible-inventory=${CLUSTER_DIR}/inventory.yml --hosts='ansible://*'
```

Known Issues

- Default ssh backend for testinfra tests is `paramiko`, which doesn't work oout of the box. It fails to connect to the host because the `ed25519` hostkey was not verified. Therefore we need to force plain ssh:// with either `connection=ssh` or `--hosts=ssh://...`

4.2.2 Behave tests

Behave tests run in a browser and test if all the interfaces are up and running and correctly connected to each other. They are integrated in the `openappstack` CLI command suite.

Prerequisites

By default the behave tests use the Chromium browser and Chromium webdriver. If you want/need to use the Firefox webdriver please refer to the manual behave test instructions below, under *Advanced Usage*.

Install Chromedriver, i.e. for Debian/Ubuntu use:

```
apt install chromium-chromedriver
```

Usage

To run all behave tests, run the following command in this repository:

```
python -m openappstack CLUSTERNAME test
```

In the future, this command will run all tests, but now only *behave* is implemented. To learn more about the `test` subcommand, run:

```
python -m openappstack CLUSTERNAME test --help
```

You can also only run a behave test for a specific application, i.e.:

```
python -m openappstack CLUSTERNAME test --behave-tags nextcloud
```

4.2.3 Advanced usage

Testinfra tests

Specify host manually:

```
py.test -s --hosts='ssh://root@example.openappstack.net'
```

Run only tests tagged with `prometheus`:

```
py.test -s --ansible-inventory=${CLUSTER_DIR}/inventory.yml --hosts='ansible://*' -m␣
↪prometheus
```

Run cert test manually using the ansible inventory file:

```
py.test -s --ansible-inventory=${CLUSTER_DIR}/inventory.yml --hosts='ansible://*' -m␣
↪certs
```

Run cert test manually against a different cluster, not configured in any ansible inventory file, either by using pytest:

```
FQDN='example.openappstack.net' py.test -sv -m 'certs'
```

or directly:

```
FQDN='example.openappstack.net' pytest/test_certs.py
```

Running testinfra tests with local gitlab-runner docker executor

Export the following environment variables like this:

```
export CI_REGISTRY_IMAGE='open.greenhost.net:4567/openappstack/openappstack'
export SSH_PRIVATE_KEY="$(cat ~/.ssh/id_ed25519_oas_ci)"
export COSMOS_API_TOKEN='...'
```

then:

```
gitlab-runner exec docker --env CI_REGISTRY_IMAGE="$CI_REGISTRY_IMAGE" --env SSH_
↪PRIVATE_KEY="$SSH_PRIVATE_KEY" --env COSMOS_API_TOKEN="$COSMOS_API_TOKEN" bootstrap
```

Behave tests

Using Firefox instead of Chromium

If you want to use Firefox instead of Chromium, you need to install the gecko driver

```
apt install firefox-geckodriver
```

Now you only need to add `-D browser=firefox` to the behave command line options, so run:

```
python -m openappstack CLUSTER_NAME test --behave-param='-D browser=firefox'
```

Using behave without the OpenAppStack CLI

Go to the `test/behave` directory and run:

For nextcloud & onlyoffice tests:

```
behave -D nextcloud.url=https://files.example.openappstack.net \
       -D nextcloud.password="$(cat ../../clusters/YOUR_CLUSTERNAME/secrets/nextcloud_
↪admin_password)" \
       -t nextcloud
```

You can replace `nextcloud` with `grafana` or `rocketchat` to test the other applications.

Run behave tests in openappstack-ci docker image

```
docker run --rm -it open.greenhost.net:4567/openappstack/openappstack/openappstack-ci␣
↪sh

  apk --no-cache add git
  git clone https://open.greenhost.net/openappstack/openappstack.git
  cd openappstack/test/behave
  behave -D nextcloud.url=https://files.ci-20410.ci.openappstack.net \
   -D nextcloud.admin.password=...
```

4.3 SSH access

You can SSH login to your VPS. Some programs that are available to the root user on the VPS:

- `kubectl`, the Kubernetes control program. The root user is connected to the cluster automatically.
- `helm` is the "Kubernetes package manager". Use i.e. `helm ls --all-namespaces` to see what apps are installed in your cluster. You can also use it to perform manual upgrades; see `helm --help`.

4.4 Using kubectl to debug your cluster

You can use `kubectl`, the Kubernetes control program, to find and manipulate your Kubernetes cluster. Once you have installed `kubectl`, to get access to your cluster with the OAS CLI:

```
$ python -m openappstack my-cluster info
```

Look for these lines:

```
To use kubectl with this cluster, copy-paste this in your terminal:

export KUBECONFIG=/home/you/projects/openappstack/clusters/my-cluster/secrets/kube_
↪config_cluster.yml
```

Copy the whole `export` line into your terminal. In *the same terminal window*, kubectl will connect to your cluster.

4.5 HTTPS Certificates

OAS uses cert-manager to automatically fetch Let's Encrypt certificates for all deployed services. If you experience invalid SSL certificates, i.e. your browser warns you when visiting Rocketchat (`https://chat.example.org`), here's how to debug this. A useful resource for troubleshooting is also the official cert-manager Troubleshooting Issuing ACME Certificates documentation.

In this example we fix a failed certificate request for `chat.example.org`. We will start by checking if `cert-manager` is set up correctly.

Did you create your cluster using the `--acme-staging` argument? Please check the resulting value of the `acme_staging` key in `clusters/YOUR_CLUSTERNAME/group_vars/all/settings.yml`. If this is set to `true`, certificates are fetched from the Let's Encrypt staging API, which can't be validated by default in your browser.

Are all cert-manager pods in the `oas` namespace in the `READY` state ?

```
$ kubectl -n oas get pods | grep cert-manager
```

Are there any `cm-acme-http-solver-*` pods still running, indicating that there are unfinished certificate requests ?

```
$ kubectl get pods --all-namespaces | grep cm-acme-http-solver
```

Show the logs of the main `cert-manager` pod:

```
$ kubectl -n oas logs -l "app.kubernetes.io/name=cert-manager"
```

You can `grep` for your cluster domain or for any specific subdomain to narrow down results.

Query for failed certificates, -requests, challenges or orders:

```
$ kubectl get --all-namespaces certificate,certificaterequest,challenge,order | grep -
↪iE '(false|pending)'
oas-apps    certificate.cert-manager.io/oas-rocketchat                 False   oas-
↪rocketchat                 15h
oas-apps    certificaterequest.cert-manager.io/oas-rocketchat-2045852889
↪   False   15h
oas-apps    challenge.acme.cert-manager.io/oas-rocketchat-2045852889-1775447563-
↪837515681   pending   chat.example.org   15h
oas-apps    order.acme.cert-manager.io/oas-rocketchat-2045852889-1775447563
↪      pending   15h
```

We see that the Rocketchat certificate resources are in a bad state since 15h.

Show certificate resource status message:

```
$ kubectl -n oas-apps get certificate oas-rocketchat -o jsonpath="{.status.
↪conditions[*]['message']}"
Waiting for CertificateRequest "oas-rocketchat-2045852889" to complete
```

We see that the `certificate` is waiting for the `certificaterequest`, lets query it's status message:

```
$ kubectl -n oas-apps get certificaterequest oas-rocketchat-2045852889 -o jsonpath="{.
↪status.conditions[*]['message']}"
Waiting on certificate issuance from order oas-apps/oas-rocketchat-2045852889-
↪1775447563: "pending"
```

Show the related order resource and look at the status and events:

```
        kubectl -n oas-apps describe order oas-rocketchat-2045852889-1775447563
```

Show the failed challenge resource reason:

```
        $ kubectl -n oas-apps get challenge oas-rocketchat-2045852889-1775447563-
↪837515681 -o jsonpath='{.status.reason}'
        Waiting for http-01 challenge propagation: wrong status code '503', expected
↪'200'
```

In this example, deleting the challenge fixed the issue and a proper certificate could get fetched:

```
$ kubectl -n oas-apps delete challenges.acme.cert-manager.io oas-rocketchat-
↪2045852889-1775447563-837515681
```

4.6 Application installation fails

Find applications that fail to install:

```
helm ls --all-namespaces | grep -i -v DEPLOYED
kubectl get helmreleases --all-namespaces | grep -i -v DEPLOYED
```

Especially the nextcloud installation process is brittle and error-prone. Lets take it as an example how to debug the root cause.

4.7 Purge OAS and install from scratch

If ever things fail beyond possible recovery, here's how to completely purge an OAS installation in order to start from scratch:

```
cluster$ apt purge docker-ce-cli containerd.io
cluster$ mount | egrep '^(.*kubelet|nsfs.*docker)' | cut -d' ' -f 3 | xargs umount
cluster$ rm -rf /var/lib/docker /var/lib/OpenAppStack /etc/kubernetes /var/lib/etcd /
↪var/lib/rancher /var/lib/kubelet /var/log/OpenAppStack /var/log/containers /var/log/
↪pods
cluster$ systemctl reboot
```

CHAPTER 5

Maintaining an Openappstack cluster

5.1 Logging

Logs from pods and containers can be read in different ways:

- In the cluster filesystem at `/var/log/pods/` or `/var/logs/containers/`.
- Using kubectl logs.
- Querying aggregated logs with grafana, see below.

5.2 Central log aggregation

We use promtail, Loki and grafana for easy access of aggregated logs. The Loki documentation is a good starting point how this setup works, and the Using Loki in Grafana gets you started with querying your cluster logs with grafana.

You will find the loki grafana integration on your cluster at https://grafana.oas.example.org/explore together with some generic query examples.

5.2.1 LogQL query examples

Please also refer to the LogQL documentation.

Query all apps for errors etc:

```
{job=~".*"} |~ "error|fail|exception|fatal"
{job=~".*"} |= "level=error"
```

Flux

Flux is responsible for installing applications. It used `helm-operator` to deploy the desired helm releases.

Query all messages from `flux`:

```
{app="flux"}
```

Query all messages from `flux` and `helm-operator`:

```
{app=~"(flux|helm-operator)"}
```

`flux` messages containing `wordpress`:

```
{app = "flux"} |= "wordpress"
```

`flux` messages containing `wordpress` without `unchanged` events (to only show the installation messages):

```
{app = "flux"} |= "wordpress" != "unchanged"
```

Filter out redundant `flux` messages:

```
{ app = "flux" } !~ "(unchanged | event=refreshed | method=Sync |␣
↪component=checkpoint)"
```

Debug oauth2 single sign-on with rocketchat:

```
{container_name=~"(hydra|rocketchat)"}
```

Cert-manager

Cert manager is responsible for requesting Let's Encrypt TLS certificates.

Query `cert-manager` messages containing `chat`:

```
{app="cert-manager"} |= "chat"
```

Hydra

Hydra is the single sign-on system.

Show only warnings and errors from `hydra`:

```
{container_name="hydra"} != "level=info"
```

5.3 Backup

5.3.1 On your provisioning machine

During the installation process, a cluster config directory is created on your provisioning machine, located in the top-level sub-directory `clusters` in your clone of the openappstack git repository. You may want to back this up. It contains all files generated during the `create` and `install` commands of the CLI, together with the generated secrets that are stored during installation.

5.3.2 On your cluster

OpenAppStack supports using the program Velero to make backups of your OpenAppStack instance to external storage via the S3 API. See the installation instructions for setup details. By default this will make nightly backups of the entire cluster (minus Prometheus data). To make a manual backup, run

```
velero create backup BACKUP_NAME --exclude-namespaces velero --wait
```

from your VPS. See `velero --help` for other commands, and Velero's documentation for more information.

Note: in case you want to make an (additional) backup of application data via alternate means, all persistent volume data of the cluster are stored in directories under `/var/lib/OpenAppStack/local-storage`.

5.4 Restore

Restore instructions will follow, please reach out to us if you need assistance.

5.5 Change the IP of your cluster

In case your cluster needs to migrate to another IP, make sure to update the IP address in `/etc/rancher/k3s/k3s.yaml` and, if applicable, your local kube config.

5.6 Delete evicted pods

In case your cluster disk usage is over 80%, kubernetes taints the node with `DiskPressure`. Then it tries to evict pods, which is pointless in a single node setup but still happened anyway. Sometimes hundreds of pods will end up in `evicted` state but still showed up after `DiskPressure` recovered. See also the out of resource handling with kubelet documentation.

You can delete all evicted pods with this:

```
kubectl get pods --all-namespaces -ojson | jq -r '.items[] | select(.status.reason!
↪=null) | select(.status.reason | contains("Evicted")) | .metadata.name + " " + .
↪metadata.namespace' | xargs -n2 -l bash -c 'kubectl delete pods $0 --namespace=$1'
```

CHAPTER 6

Upgrading OpenAppStack

6.1 Upgrading from 0.3.0 to 0.4.0

There is no easy upgrade path from version 0.3.0 to version 0.4.0. As far as we know, nobody was running OpenAppStack apart from the developers, so we assume this is not a problem.

If you do need to upgrade, this is how you can migrate your data. Backup all the data available under `/var/lib/OpenAppStack/local-storage`, create a new cluster using the installation instructions, and putting back the data. This migration procedure might not work perfectly.

Use `kubectl get pvc -A` on your old cluster to get a mapping of all the PVC uuids (and thus their folder names in `/var/lib/OpenAppStack/local-storage`) to the pods they are bound to.

Then, delete your old OpenAppStack, and install a new one with version number 0.4.0 or higher. You can upload your backed up data into `/var/lib/OpenAppStack/local-storage`. All PVCs will have new unique IDs (and thus different folder names). You have to manually match the folders from your backup with the new folders.

Additionally, if you want to re-use your old `settings.yaml` file, this data needs to be added to it:

```
prometheus_enable_ingress: false

backup:
  s3:
    # Disabled by default. To enable, change to `true` and configure the
    # settings below. You'll also want to add "velero" to the enabled
    # applications a bit further in this file.
    # Finally, you'll also need to provide access credentials as
    # secrets; see the documentation:
    # https://docs.openappstack.net/en/latest/installation_instructions.html#step-2-
↪optional-cluster-backups-using-velero
    enabled: false
    # URL of S3 service. Please use the principal domain name here, without the
    # bucket name.
    url: "https://store.greenhost.net"
    # Region of S3 service that's used for backups.
```

(continues on next page)

(continued from previous page)

```
    # For some on-premise providers this may be irrelevant, but the S3
    # apparently requires it at some point.
    region: "ceph"
    # Name of the S3 bucket that backups will be stored in.
    # This has to exist already: Velero will not create it for you.
    bucket: "openappstack-backup"
    # Prefix that's added to backup filenames.
    prefix: "test-instance"

# A whitelist of applications that will be enabled.
enabled_applications:
  # System components, necessary for the system to function.
  - 'cert-manager'
  - 'letsencrypt-production'
  - 'letsencrypt-staging'
  - 'ingress'
  - 'local-path-provisioner'
  - 'single-sign-on'
  # The backup system Velero is disabled by default, see settings under `backup`␣
↪above.
  # - 'velero'
  # Applications.
  - 'grafana'
  - 'loki-stack'
  - 'nextcloud'
  - 'prometheus'
  - 'rocketchat'
  - 'wordpress'
```

6.2 Upgrading to 0.3.0

Upgrading from versions earlier than `0.3.0` requires manual intervention.

- Move your local `settings.yml` file to a different location:

```
cd CLUSTER_DIR
mkdir -p ./group_vars/all/
mv settings.yml ./group_vars/all/
```

- Flux is now used to install and update applications. For that reason, we need you to remove all helm charts (WARNING: You will lose your data!):

```
helm delete --purge oas-test-cert-manager oas-test-local-storage \
    oas-test-prometheus oas-test-proxy oas-test-files`
```

  - After removing all helm charts, you probably also want to remove all the `pvcs` that are left behind. Flux will not re-use the database PVCs created for these applications. Find all the pvcs by running `kubectl get pvc --namespace oas-apps` and `kubectl get pvc --namespace oas`

6.3 Troubleshooting

If you encounter problems when you upgrade your cluster, please make sure first to include all potential new values of `ansible/group_vars/all/settings.yml.example` to your `clusters/YOUR_CLUSTERNAME/`

`group_vars/all/settings.yml`, and rerun the installation script.

CHAPTER 7

Comparable projects

Other open source projects similar to OpenAppStack exist. Each of the platforms listed here, like OpenAppStack, provide a suite of open source cloud applications. Each of the platforms, like OpenAppStack, include their own user management dashboard and all of them offer single sign-on (SSO) features.

However there are changes in implementation that make OpenAppStack different from these alternatives. As far as we found, none of the projects listed below will automatically update your applications without you having to push a button, for example.

Sandstorm allows applications to be installed as so called Grains. Each grain is a copy of the complete application made for a specific purpose. For example, a grain for document editor "Etherpad" contains not only the data written in the notepad, but also the Etherpad and database software. If you have two notepads, you also have two copies of the software. With many users, this approach can run into limits.

YunoHost is mostly based on Debian and its package management and user management system. As a result, Yunohost is relatively lightweight. However, another result of this is that it is likely that if one of the applications on your Yunohost contain a security hole, the data of the other applications is compromised as well. This is less likely in OpenAppStack because it separates the environments of applications from each other to seal them off.

Cloudron offers a similar application suite. In contrast to OpenAppStack, Cloudron requires a paid account if you want to use more than two applications or more than five users.

All mentioned platforms require applications running on it to be changed in some way, for example to make use of the authentication system. OpenAppStack tries to steer clear of changing the applications it includes. As long as they support OpenID Connect, so you can sign into it, we can usually run the application as-is. In most cases, OpenAppStack will only need one intervention before an update can be pushed to users: it needs to be tested. We want to make sure that an application works together with the platform and other applications before we let you use it. We work towards a fully automated test suite, so even this would not require human intervention.

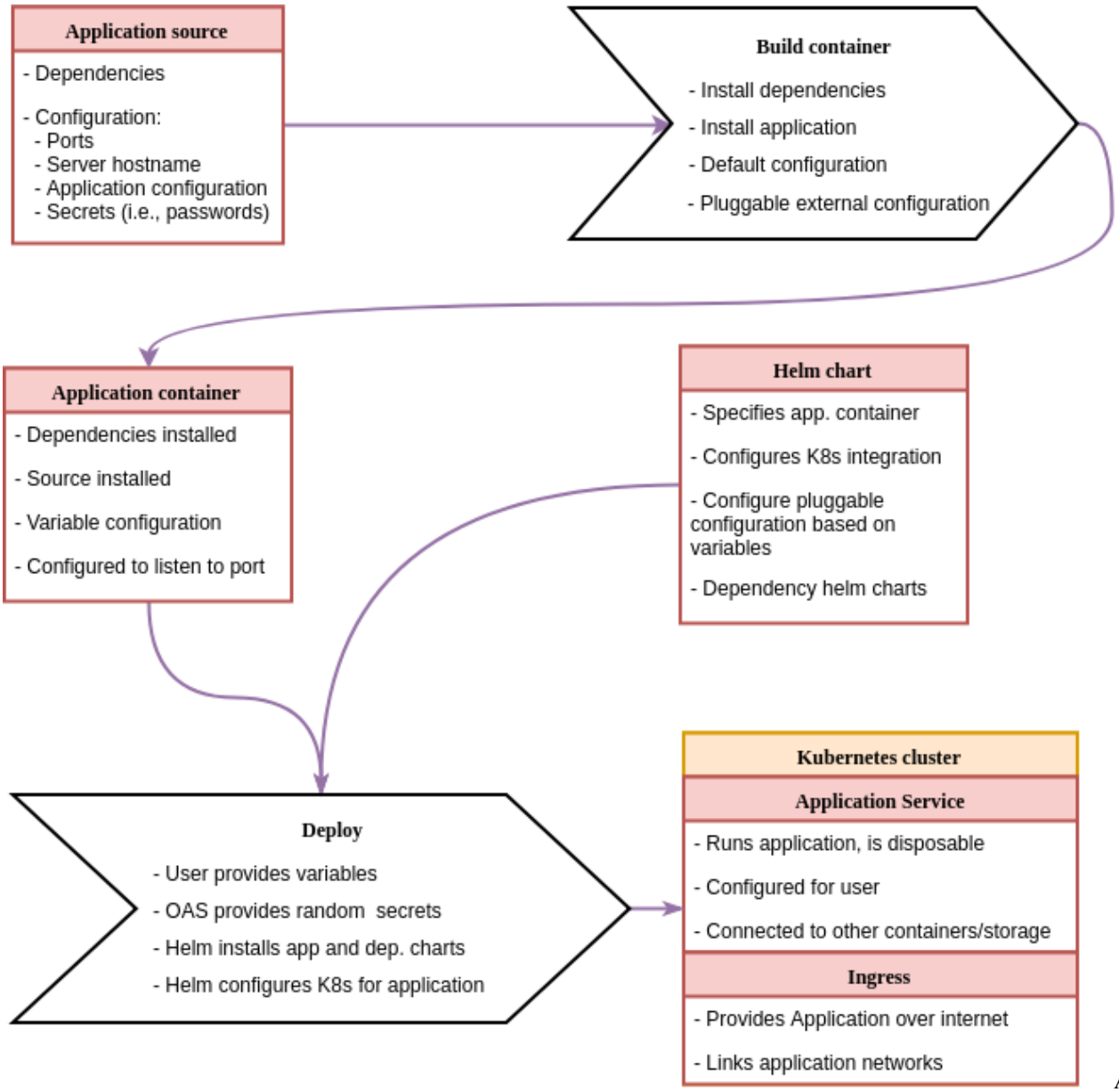
CHAPTER 8

OpenAppStack Design

This article covers the basic design of OpenAppStack.

8.1 Application build pipeline

The following diagram explains the process to go from an application's source code to a deployment on OpenAppStack.



Application build process

These are the steps in more detail:

- Build container (this process should be maintained by application developer by providing a Dockerfile with the application)
    1. Get application package (source code, installation package, etc.)
        1. If not part of the package: get default configuration for the application
    2. Build container with application package installed
        1. Install application dependencies
        2. Install application package
        3. Setup default configuration
        4. Setup pluggable configuration override, can be:

    - Reading environment variables
    - Extra configuration file mounted into the container elsewhere
- Helm chart
  - Deployment configuration to specify:
    - The container(s) that should be deployed.
    - The port(s) that they expose.
    - Volume mounts for configuration files and secrets.
    - Live/readyness probes
    - Persistent storage locations and methods
    - A lot of other things
  - Service configuration to specify:
    - Ports exposed to the user of the application
  - Ingress configuration to specify:
    - How to proxy to the application (which hostname or URL)
    - Some authentication plugins (http auth, for example)
  - Custom files:
    - Add file templates for mountable application configuration files
    - Files that specify integrations with other services
- Deploy
  1. Create `values.yaml` file with the variables for the Helm deployment to the Kubernetes cluster
  2. "Manually" add secrets to the Kubernetes cluster.
  3. Run `helm install` to install the customised application.

8.2 Configuration

As can be seen in the images, applications are expected to have two different types of configuration. Containers should provide a default configuration, that at least configures things like the port the application runs on, the locations for log files, etc.

What we call the *external configuration* is provided by the user. This includes overrides of the default application, as well as variables like the hostname that the application will run on and listen to and the title of the web interface.

OpenAppStack will use Helm charts to provide the external configuration for the "Deploy" step. Helm charts can contain configuration file templates with default values that can be overridden during the installation or upgrade of a helm chart.

8.3 Application containers

For inclusion in OpenAppStack, it is required that the application developers provide Docker containers for their applications. There are several reasons for this:

- If application developers do not provide a container, chances are they also do not think about how their application would update itself after a new container is deployed. This can lead to problems with things like database migrations.
- Maintaining the containerisation for an application can, in most cases, not be fully automated.

8.3.1 Container updates

When an application update is available, these updates need to be rolled out to OpenAppStack instances. This will be done according the following steps:

1. Application container is built with new application source and tagged for testing.
2. Helm chart for application is updated to provide new container.
3. Helm chart is deployed to an OpenAppStack test cluster following the steps in the diagram above.
4. Application is tested with automated tests
5. If tests succeed, new container is tagged for release.
6. OpenAppStack automated update job fetches new Helm chart and upgrades current instance using Helm.

Most of these steps can be developed by configuring a CI system and configuring Kubernetes and Helm correctly. The automated update job that will run on OpenAppStack clusters will be developed by us.

8.4 Persistent data

Containerised applications are normally "stateless" (meaning no data is saved inside the containers). However, it is possible to mount persistent volumes to specific directories in the container, basically adding a persistent layer on top of the containerised application. To provide this in OAS's simple setup, we use a local storage provisioner that automatically provides persistent data on the VPS running OAS to an application that requests it.

8.5 Automatic updates

OpenAppStack has an auto-update mechanism that performs unattended upgrades to applications. Flux is the system running in the cluster that is responsible for these updates.

Technically, flux is split up in two components: `flux` and `helm-operator`. `flux` watches a git repository (or subdirectory thereof) for source files that prescribe which application versions should be installed, and stores a copy of those prescriptions inside the cluster as a Kubernetes manifest (of kind `HelmRelease`).

`helm-operator` watches those in-cluster `HelmRelease` manifests, checks whether the listed applications are already installed – including correct versions and other settings – and performs any actions that are necessary to make sure that the cluster state matches the prescriptions: installing new applications, upgrading others.

Which git repository is watched by flux, is configurable. For typical production OpenAppStack deployments, this is set to be `https://open.greenhost.net/openappstack/openappstack` – the `HelmRelease` files are stored in the `flux` directory. The OpenAppStack team considers available upstream updates (say an update to Nextcloud). If the new Nextcloud version passes our tests, the team changes the corresponding application description file in the git repository (in this case `flux/nextcloud.yaml`) to reference the new version. OpenAppStack deployments that are configured to watch this git repository will see the change, record it in the in-cluster `HelmRelease` manifest, and have their `helm-operator` perform the Nextcloud upgrade.

8.5.1 Customising which applications are installed

The `HelmRelease` files in the `flux` directory form the applications that are available for installation. There is an additional mechanism though, to allow the cluster administrator to choose which applications are actually installed. You might want to leave out some apps that you think you won't use, and save some resources that way. You can choose which apps to enable/disable by modifying the `enabled_applications` list in `CLUSTERDIR/group_vars/all/settings.yml`, and afterwards running the OAS install procedure.

Every `HelmRelease` file should have a corresponding Kubernetes Secret, which we call a "settings secret". For the apps that are part of OpenAppStack, these secrets are created by the OpenAppStack installation procedure, so you don't need to handle them unless you want to customise the set of installed applications.

The subdirectory of `/flux` where the `HelmRelease` file is located corresponds to the namespace of the secret. For example, the `HelmRelease` file `/flux/oas-apps/nextcloud.yml` corresponds to the Kubernetes secret `nextcloud-settings` in the namespace `oas-apps`.

This Kubernetes secret contains two keys:

- `enabled`: this should contain a simple string: `true` to enable the application, or `false` to disable it;
- `values.yaml`: this contains a yaml-formatted string with Helm values that are supplied to Helm when the application is installed.

8.5.2 Local development

When developing OpenAppStack, it's nice to be able to change the application versions for your development cluster only, without affecting production clusters. One way to do that is to set the `flux_source.repo` and/or `flux_source.branch` ansible variables to point to another branch of the `open.greenhost.net/openappstack/openappstack` repository, or to a different repository.

To make this easier, we included a way to serve up a git repository with `HelmRelease` manifests from the cluster itself, so you don't need an external Gitlab or Github project. This feature is disabled by default, and can be enabled by setting the `local_flux` ansible variable to `true`. If enabled, this will change several things:

- when you run the OpenAppStack bootstrap ansible playbook from your workstation, the current contents of the `flux` directory from your workstation are copied to a directory (`/var/lib/OpenAppStack/local-flux`) on the OpenAppStack host machine; also, a git commit is created in that directory on the host machine from these updated contents;
- as part of the OpenAppStack installation, an nginx instance is deployed that serves up the contents of `/var/lib/OpenAppStack/local-flux` over http;
- `flux` is configured to read the `HelmRelease` files from that git repository, served by the in-cluster nginx. In particular, the `flux_source` variables are ignored if `local_flux` is enabled.

The `local-flux` feature is also used by our CI pipeline, in order to be as similar as possible to a production installation, in particular using `flux` and `helm-operator` for the installation process, while still being completely separate from the production applications versions as prescribed by the master repository at `open.greenhost.net/openappstack/openappstack`.

Triggering an update

Both `flux` and `helm-operator` check at regular intervals (currently 1 hour and 20 minutes, respectively) whether there is an upgrade to perform. If you don't want to wait for that after making a change, you can trigger an update:

- to let `flux` re-read the `HelmRelease` files from the git repo (be it the OpenAppStack master one, a `local-flux` one, or yet another one), log in to the host machine and do

```
$ fluxctl --k8s-fwd-ns=oas sync
```

If there are any changes to `HelmRelease` manifests, `helm-operator` is notified of that through the Kubernetes API and should act on them more or less instantly (though the actual installation or upgrade could take quite a while, depending on what needs to be done).

- If, for some reason, the `HelmRelease` manifests are still in sync between git repository and cluster, but you'd still like to let `helm-operator` check whether these `HelmReleases` match what's actually installed, you can force that by deleting the `helm-operator` pod:

```
$ kubectl delete pod -n oas -l app=helm-operator
```

A new `helm-operator` pod should be created automatically, and in our experience will do a reconciliation run soon.

CHAPTER 9

Reference documentation

9.1 File structure

During the installation process, the following files and directories are created:

- `/var/lib/OpenAppStack/local-storage`: all application data (e.g., Nextcloud files) are stored here.
- `/var/lib/OpenAppStack/source`: On development setups, this contains the `local-flux` git repository.
- `/var/lib/rancher/k3s`: this contains the files related to your "Kubernetes" cluster.
  - The kubectl configuration file is located at `/etc/rancher/k3s/k3s.yaml`