OpenAppStack

Release 0.7.0

Greenhost

Nov 08, 2021

INSTALLATION

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

ONE

INSTALLATION OVERVIEW

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.

Note: You should also have a trusted machine to run the installer on, i.e. your laptop. We call this the `provisioning machine`. All commands in these installation instructions need to be run this provisioning machine that is *not* the server that will run OpenAppStack, unless specified otherwise.

1.1 Setup

1.1.1 Common prerequisites

Wether you want to create a kubernetes cluster first or want to install OpenAppStack on an existing cluster, these are the common prerequisites:

- You need Python 3 with its development files, Pip and Git installed (`apt install python3-pip python3-dev git` on Debian)
- 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`.

1.1.2 Clone the OpenAppStack git repository

On your **provisioning machine**, clone the OpenAppStack git repository and checkout the latest release branch (currently `v0.6`):

```
$ git clone -b v0.7 https://open.greenhost.net/openappstack/openappstack.git
$ cd openappstack
```

1.1.3 Create a python virtual environment

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
```

1.1.4 Install requirements

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

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

1.1.5 OpenAppStack CLI client usage

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 `oas.example.org` as the cluster name. Try it out by running

```
$ python -m openappstack oas.example.org --help
```

1.2 Installation options

1. If you want to create a Kubernetes cluster from scratch on a dedicated server or virtual machine please start with *Create a kubernetes cluster* and then continue with *Install OpenAppStack*.
2. If you want to install Openappstack on an existing Kubernetes cluster you can skip the above instructions and start with *Install OpenAppStack*.

CHAPTER

TWO

CREATE A KUBERNETES CLUSTER

This document describes how you can use OpenAppStack to install k3s, a lightweight Kubernetes distribution on a virtual private server.

For OpenAppStack we will set up a "single-node" kubernetes 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.

2.1 Prerequisites

During these instructions, you are asked to create a VPS, or have a bare metal server ready. The server should meet these requirements:

- Debian "buster" installed
- A public IP address
- The ability to create DNS records for this IP
- 6 cores and 12 GB of RAM
- At least 25GB of disk space for installation, plus more for application data. We recommend starting with 30GB.
- Root ssh access
- `ssh-agent` to give you access to your VPS

In this guide, we will create a cluster with IP address `1.2.3.4` on domain `oas.example.org`. Substitute these two variables with your IP address and your domain.

2.2 Step 1: Create cluster configuration

To create a config for your cluster, use the `create` subcommand of the OpenAppStack CLI. First, choose a name (we chose `oas.example.org`) for your cluster. Then run the following command to get information about the `create` subcommand:

```
$ python -m openappstack oas.example.org create --help
```

If you want the installation script to automatically create a VPS for you, check *Cluster creation with the Greenhost API*. Otherwise, continue here.

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*.

Create the initial OpenAppStack configuration for your VPS by running the following command:

```
$ python -m openappstack oas.example.org create \
  oas.example.org \
  --ip-address 1.2.3.4
```

This configures your cluster under the fully qualified domain name (FQDN) `oas.example.org`, To break down the command:

- the first, positional argument `oas.example.org` tells the cluster the domain it will be hosted on. This should be a (subdomain of a) domain you own.
- `--ip-address 1.2.3.4` tells the script the IP address of your VPS. This will be used to find the VPS during the installation procedure.

The configuration has now been written to the `clusters/oas.example.org` on your provisioning machine.

2.3 Step 2: Configure DNS

Next, make sure that you have two DNS records that point to your cluster. Create these two DNS records:

- An `A` record `oas.example.org` pointing to the VPS's 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 VPS's 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.

2.4 Step 3: Create cluster

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.

Run the `install` command with the CLI to completely configure your VPS for OpenAppStack.

```
$ python -m openappstack oas.example.org install
```

This will take a few minutes. It installs k3s, a lightweight Kubernetes and useful tools like kubectl (Kubernetes cli tool), krew (a kubectl plugin manager), flux (used for automated updates) and velero (Kubernetes resources and persistent volumes backup) on it.

Note: It is possible to re-run the `install` command with a newer version of the installation script. This usually updates k3s and can have other benefits.

Now you have a single-node k3s/Kuberetes cluster running and can continue with *Install OpenAppStack*.

2.5 Advanced installation

2.5.1 Cluster creation 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 oas.example.org create --subdomain oas example.org
```

- Here is an example of a complete creation command:

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

Let's break down the arguments:

- `--create-droplet`: Use the Greenhost API to create a new VPS
- `--create-hostname oas.example.org`: Create a VPS with hostname `oas.example.org`
- `--ssh-key-id 112`: Use SSH key ID 112 (you can find your SSH key ID in the Cosmos Service Centre under *VPS Cloud -> Installation SSH Keys*. Hover over a button there to see the ID in the URL it uses.
- `--create-domain-records`: Use the Greenhost API to create DNS records If you do this, you can skip *Step 2: Configure DNS*. The following records are created:
  - An `A` record `oas.example.org` pointing to the VPSs IP address
  - A `CNAME` record `*.oas.example.org` pointing to `oas.example.org`.
- `--subdomain oas`: Only needed when you use `--create-domain-records` so the Greenhost API can find your domain. Instead of using positional argument `oas.example.org` you need to provide

You can now continue to *Step 2: Configure DNS*, or *Step 3: Create cluster* if you used the API to create the DNS records.

CHAPTER

THREE

INSTALL OPENAPPSTACK

This guide explains how to install OpenAppStack either on an existing Kubernetes cluster that you have setup entirely yourself, or on a cluster you created following the OpenAppStack *Create a kubernetes cluster* guide. Please choose one of the below options:

A. Install on cluster created with OpenAppStack cli

If you followed the *Create a kubernetes cluster* guide to setup a Kubernetes cluster using the OpenAppStack CLI tool you're all set and can continue by following the steps below.

B. Install on existing cluster

Note:

- Installation on an existing Kubernetes cluster ist still experiental and might not work depending on the setup of your cluster/cloud provider. We'll be happy to recieve feedback from your experiences though altough we cannot guarrente !

Prerequisites:

- A single-node Kubernetes cluster
- A `kube_config.yml` file for API access

Configure DNS: Please follow *Step 2: Configure DNS* how to setup the DNS records for OpenAppStack.

Setup:

- Create a directory containing your cluster configuration, i.e. `mkdir -p clusters/oas.example.org`
- Copy your `kube_config.yml` file inside your cluster config directory and rename it to `kube_config_cluster.yml`: `cp kube_config.yml clusters/oas.example.org/kube_config_cluster.yml`

Continue by following the steps below.

3.1 Flux configuration

3.1.1 Prerequisites

- `kubectl` (installation instructions)
- `flux version 0.14.2` Download flux_0.14.2_linux_amd64.tar.gz

Copy the file `install/.flux.env.example` to your cluster dir `clusters/oas.example.org/.flux.env`. This file contains the last bit of information you need to configure. You **have to** configure the following values. The rest are optional.

- `ip_address`: The IP address of your cluster
- `domain`: The FQDN of your cluster
- `admin_email`: a valid email address for the system administrator of your cluster.

3.1.2 Outgoing email

If you want apps like Nextcloud, RocketChat and Prometheus to be able to send email notifications, you need to provide an email account.

Note: OpenAppStack does not set up an email server for you. In order to enable outgoing emails you need to provide an already existing email account.

OpenAppStack uses SMTP to send emails. Search your email provider's helpdesk for SMTP configuration details and enter them in the `clusters/oas.example.org/.flux.env` file as follows:

```
# Enable sending mails
outgoing_mail_enabled=true
# Email address that the cluster sends mails from. Needs to be an existing SMTP
# login
outgoing_mail_from_address=admin@example.org
# Same outgoing mail address, but only the part before the '@'
outgoing_mail_from_local_part=admin
# Same outgoing mail address, but only the part after the '@'
outgoing_mail_domain=example.org
# SMTP password for the outgoing mail address
outgoing_mail_smtp_password=CHANGEME
# SMTP username, often the same as the outgoing email address
outgoing_mail_smtp_user=admin@example.org
# SMTP login data.
outgoing_mail_smtp_host=smtp.greenhost.nl
outgoing_mail_smtp_authtype=LOGIN
outgoing_mail_smtp_port=587
```

3.1.3 Backups with Velero

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, Rocket.Chat 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 configure Velero, edit the file `clusters/oas.example.org/.flux.env`, and configure the settings with the `backup_s3_` prefix.

Then continue with the installation procedure as described below. At the end of the installation procedure, you have to install the `velero` application.

3.2 Step 1: Install core applications

Before you can start, you need to execute a few commands from the installation directory **on your provisioning machine**. Don't forget to replace `oas.example.org` with your domain.

```
export CLUSTER_DIR=clusters/oas.example.org

# Copy the installation kustomization to your cluster directory
cp install/kustomization.yaml $CLUSTER_DIR/

# Tell kubectl to use your cluster's kube_config
export KUBECONFIG=$CLUSTER_DIR/kube_config_cluster.yml

# Ensure flux-system namespace is created
kubectl get namespace flux-system 2>/dev/null || kubectl create namespace flux-system

# This inserts the configuration from .flux.env into your cluster as a "secret"
kubectl apply -k $CLUSTER_DIR
```

After you have executed that code, your terminal should show:

```
secret/oas-cluster-variables created
```

Next, run:

```
./install/install-openappstack.sh
```

This installs the *core* of OpenAppStack into your cluster. To see what's included, check the `flux2/infrastructure` and the `flux2/core` folders in the git repository.

3.3 Step 2: Install additional applications

After the script completes, you can install applications by running the other installation scripts in the `install` folder. At the moment, we have scripts to install:

- Nextcloud and Onlyoffice with `./install/install-app.sh nextcloud`
- Rocket.Chat with `./install/install-app.sh rocketchat`
- Wekan with `./install/install-app.sh wekan`
- WordPress with `./install/install-app.sh wordpress`
- Velero with `./install/install-app.sh velero` (only if you have configured it in *Backups with Velero*).

When the installation scripts complete, the application installation may still be running on the OpenAppStack cluster. You can monitor the progress by running `flux get kustomizations` (use `watch flux get kustomizations` to get updates). If all kustomizations have been applied correctly, you can monitor specific application releases by running `watch flux get helmreleases --all-namespaces`.

3.4 Step 3: 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.

3.5 Step 4: Let us know!

We would love to hear about your experience installing OpenAppStack. If you encountered any problems, please create an issue in our issue tracker. If you didn't please still reach out as described on our contact page and tell us how you like OpenAppStack so far. We want to be in communication with our users, and we want to help you if you run into problems.

CHAPTER

FOUR

USAGE

4.1 OAS User panel

After all the applications are installed, the first thing to do is log into https://admin.oas.example.org. This is the "user panel" , a place where you can create, edit and delete users. These users can be used to log into the applications listed below. You can log in with the user "admin". The password can be found by running

```
python3 -m openappstack oas.example.org secrets
```

Search for `userbackend_admin_password`.

After logging in, you will see an overview of all the installed 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: If you don't see applications, make sure you have installed at least one optional application in additional_apps of the installation procedure.

For creating users follow the user creation documentation.

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.

Links to all available apps are shown in the userpanel dashboard, which can be used to access these apps without a need to bookmark all URLs.

4.2 Core applications

These applications are available after the installation is completed successfully:

4.2.1 Grafana

Grafana is a dashboard application that shows you information about the status of your cluster collected by Prometheus.

Single sign-on users

Users that have the "Admin" label turned on in the user panel, will have admin rights in Grafana. Other users are able to see graphs, but can not change anything.

4.3 Optional applications

4.3.1 Nextcloud

Nextcloud is a file sharing and communication platform. These Nectcloud apps can be accessed from the top navigation bar:

Files & synchronization

You can access your files with the Nextcloud Web interface and create, preview, edit, delete, share, and re-share files. See the Files & synchronization user manual for general usage and Desktop and mobile synchronization for setting up file sync on your mobile or dektop device.

Calendar

The Nextcloud Calendar app works similar to other calendar applications you can sync your Nextcloud calendars and events with. Follow the Calendar user manual for general usage and the *Nextcloud groupware docs <https://docs.nextcloud.com/server/latest/user_manual/th/pim/index.html>* for syncing your calendars with your mobile or desktop devices.

Passwords

A simple, yet feature rich password manager for Nextcloud. See Passwors user handbook for more details, including using the Browser extenstions.

These are the mobile apps that you can use to access your passwords from your smartphone:

- [NC passwords app for Android (by joleaf)](https://gitlab.com/joleaf/nc-passwords-app)
- [Nexcloud password app for Android (by daper)](https://github.com/daper/nextcloud-passwords-app)
- [Nextcloud passwords for iOS](https://github.com/johannes-schliephake/nextcloud-passwords-ios)

You'll find how to configure file or calendar sync with your smartphone or desktop in the Nextcloud Groupware documentation.

4.3.2 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.

4.3.3 RocketChat

RocketChat is a team chat application.

4.3.4 Wekan

Wekan is a Kanban board application.

4.3.5 WordPress

WordPress is a website content management system.

CHAPTER

FIVE

TESTING GUIDE

Great that you want to take OpenAppStack for a test drive ! This guide 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.

5.1 OAS installation

First we'd like you to setup an OpenAppStack cluster by yourself, following the *Installation overview* and *Usage* documentation and make sure you complete all steps.

5.2 Command line tests

Please *Run the CLI tests* which checks the overall functionality of your cluster and include the output in your feedback.

5.3 Testing Instructions for all apps

Please login using single sign-on as `admin` and see if you have admin rights granted (usually there's an app specific admin panel available or admin functionality like configuring users). Afterwards logout and login again as the non-admin single-sign-on user you created earlier in the OAS admin panel. You should *not* have any admin privileges now.

5.4 Testing Instructions for specific apps

5.4.1 Nextcloud

Please browse to Nextcloud using the link from your user dashboard app list 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.

Files & synchronization

Please try uploading, downloading, moving or copying files.

Calendar

Please test the basic functionality of the calendar, including advanced features like recurrence, custom notifications or inviting attendees.

Passwords

Please create and manage different passwords. In addition to that please try

Nextcloud sync applications

- Please try syncing with your smartphone or desktop using one of the apps mentioned in *Nextcloud*.

5.4.2 Onlyoffice

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.

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.

5.4.3 Wordpress

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.

5.5 Providing feedback

If you have not done so 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

SIX

MAINTENANCE

6.1 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.

6.2 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.

6.2.1 LogQL query examples

Please also refer to the LogQL documentation.

Query all aggregated logs (unfortunatly we can't find a better way of doing this since LogQL always expects a stream label to get queried):

```
logcli query '{foo!="bar"}'
```

Query all logs for a keyword:

```
logcli query '{foo!="bar"} |= "error"'
```

Query all k8s apps for errors using a regular expression:

```
logcli query '{job=~".*"} |~ "error|fail|exception|fatal"'
```

Flux

Flux is responsible for installing applications. It uses four controllers:

- `source-controller` that tracks Helm and Git repositories like https://open.greenhost.net/openappstack/openappstack for updates.
- `kustomize-controller` to deploy `kustomizations` that often install `helmreleases`.
- `helm-controller` to deploy the `helmreleases`.
- `notification-controller` that is responsible for inbound and outbound flux messages

Query all messages from the `source-controller`:

```
{app="source-controller"}
```

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

```
{app=~"(source-controller|helm-controller)"}
```

`helm-controller` messages containing `wordpress`:

```
{app = "helm-controller"} |= "wordpress"
```

`helm-controller` messages containing `wordpress` without `unchanged` events (to only show the installation messages):

```
{app = "helm-controller"} |= "wordpress" != "unchanged"
```

Filter out redundant `helm-controller` messages:

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

Debug oauth2 single sign-on with rocketchat:

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

Query kubernetes events processed by the `eventrouter` app containing `warning`:

```
logcli query '{app="eventrouter"} |~ "warning"'
```

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"
```

6.3 Backup

6.3.1 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. Although these files are not essential for your OpenAppStack cluster to continue functioning, you may want to back this folder up because it allows easy access to your cluster.

6.3.2 On your cluster

OpenAppStack supports using the program Velero to make backups of your OpenAppStack instance to external storage via the S3 API. See *Backups with Velero* in 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

```
cluster$ 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`.

6.4 Restore

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

6.5 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 and inventory.yml in the cluster directory `clusters/oas.example.org`.

6.6 Delete evicted pods

In case your cluster disk is full, kubernetes taints the node with `DiskPressure`. Then it tries to evict pods, which is pointless in a single node setup but can still happen. We have experienced hundreds of pods in `evicted` state that still showed up after `DiskPressure` had recovered. See also the out of resource handling with kubelet documentation.

You can delete all evicted pods with this command:

```
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

SEVEN

UPGRADING

7.1 Upgrading to 0.7.0

Because of problems with Helm and secret management we had to move away from using a helm chart for application secrets, and now use scripts that run during installation to manage secrets. Because we have removed the `oas-secrets` helm chart, Flux will remove the secrets that it has generated. **It is important that you back up these secrets before switching from v0.6 to v0.7!**

Note: Before you start, please ensure that you have the right `yq` tool installed, because you will need it later. There are two very different versions of `yq`. The one you need is the go based yq from Mike Farah, which installs the same binary name `yq` as the python-yq, while both have different command sets. The yq needed here can be installed by running `sudo snap install yq`, `brew install yq` or with other methods from the yq installation instructions.

If you're unsure which `yq` you have installed, look at the output of `yq --help` and make sure `eval` shows up under `Available Commands:`.

To back-up your secrets, run the following script:

```
bash
#!/usr/bin/env bash

mkdir secrets-backup

kubectl get secret -o yaml -n flux-system  oas-cluster-variables > secrets-backup/oas-
↪cluster-variables.yaml
kubectl get secret -o yaml -n flux-system  oas-wordpress-variables > secrets-backup/oas-
↪wordpress-variables.yaml
kubectl get secret -o yaml -n flux-system  oas-wekan-variables > secrets-backup/oas-
↪wekan-variables.yaml
kubectl get secret -o yaml -n flux-system  oas-single-sign-on-variables > secrets-backup/
↪oas-single-sign-on-variables.yaml
kubectl get secret -o yaml -n flux-system  oas-rocketchat-variables > secrets-backup/oas-
↪rocketchat-variables.yaml
kubectl get secret -o yaml -n flux-system  oas-kube-prometheus-stack-variables > secrets-
↪backup/oas-kube-prometheus-stack-variables.yaml
kubectl get secret -o yaml -n oas          oas-prometheus-basic-auth > secrets-backup/
↪oas-prometheus-basic-auth.yaml
kubectl get secret -o yaml -n oas          oas-alertmanager-basic-auth > secrets-backup/
↪oas-alertmanager-basic-auth.yaml
kubectl get secret -o yaml -n flux-system  oas-oauth-variables > secrets-backup/oas-
↪oauth-variables.yaml
```

(continues on next page)

(continued from previous page)

```
kubectl get secret -o yaml -n flux-system  oas-nextcloud-variables > secrets-backup/oas-
↪nextcloud-variables.yaml
```

This script assumes you have all applications enabled. You might get an error like:

```
Error from server (NotFound): secrets "oas-wekan-variables" not found
```

This is not a problem, but it *does* mean you need to add an oauth secret for Wekan to the file `secrets-backup/oas-oauth-variables.yaml`. Copy one of the lines under "data:", rename the field to `wekan_oauth_client_secret` and enter a different random password. Make sure to base64 encode it (`echo "<your random password>" | base64`).

This script creates a directory called `secrets-backup` and places the secrets that have been generated by Helm in it as `yaml` files.

Now you can upgrade your cluster by running `kubectl -n flux-system patch gitrepository openappstack --type merge -p '{"spec":{"ref":{"branch":"v0.7"}}}'` or by editing the `gitrepository` object manually with `kubectl -n flux-system edit gitrepository openappstack` and setting `spec.ref.branch` to `v0.7`.

Flux will now start updating your cluster to version `0.7`. This process will fail, because it will remove the secrets that you just backed up. Make sure that the `oas-secrets` helmrelease has been removed by running `flux get hr -A`. You might also see that some helmreleases start failing to be installed because important secrets do not exist anymore.

As soon as the `oas-secrets` helmrelease does not exist anymore, you can run the following code:

```
#!/usr/bin/env bash

# Again: make sure you use https://github.com/mikefarah/yq -- install with `snap install␣
↪yq`
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-wordpress-variables.yaml |␣
↪kubectl apply -f -
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-wekan-variables.yaml |␣
↪kubectl apply -f -
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-single-sign-on-variables.
↪yaml | kubectl apply -f -
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-rocketchat-variables.yaml␣
↪| kubectl apply -f -
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-kube-prometheus-stack-
↪variables.yaml | kubectl apply -f -
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-prometheus-basic-auth.yaml␣
↪| kubectl apply -f -
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-alertmanager-basic-auth.
↪yaml | kubectl apply -f -
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-oauth-variables.yaml |␣
↪kubectl apply -f -
```

(continues on next page)

(continued from previous page)

```
yq eval 'del(.metadata.annotations,.metadata.labels,.metadata.creationTimestamp,.
↪metadata.resourceVersion,.metadata.uid)' secrets-backup/oas-nextcloud-variables.yaml |␣
↪kubectl apply -f -
```

Again this script assumes you have all applications installed. If you get the following error, you can ignore it:

```
error: error validating "STDIN": error validating data: [apiVersion not set, kind not␣
↪set]; if you choose to ignore these errors, turn validation off with --validate=false
```

Now Flux should succeed in finishing the update. Some helmreleases or kustomizations might have already failed because the secrets did not exist. Once failed, you can retrigger reconciliation of a kustomization using the commands `flux reconcile kustomization ...` or `flux reconcile helmrelease ...`. This can take quite a while (over an hour some times), because Flux waits for some long timeouts before giving up and re-starting a reconciliation.

7.1.1 Potential upgrade issues

Some errors we've seen during our own upgrade process, and how to solve them:

SSO helm upgrade failed

```
oas          single-sign-on          False Helm upgrade failed: template: single-sign-on/
↪templates/secret-oauth2-clients.yaml:9:55: executing "single-sign-on/templates/secret-
↪oauth2-clients.yaml" at <b64enc>: invalid value; expected string  0.2.2     False
```

This means that the `single-sign-on` helmrelease was created with empty oauth secrets. The secrets will get a value once the `core` *kustomization* is reconciled: `flux reconcile ks core` should solve the problem.

If that does not solve the problem, you should check if the secret contains a value for all the apps:

```
# kubectl get secret -n flux-system oas-oauth-variables -o yaml
apiVersion: v1
data:
  grafana_oauth_client_secret: <redacted>
  nextcloud_oauth_client_secret: <redacted>
  rocketchat_oauth_client_secret: <redacted>
  userpanel_oauth_client_secret: <redacted>
  wekan_oauth_client_secret: <redacted>
  wordpress_oauth_client_secret: <redacted>
...
```

If your secret lacks one of these variables, use `kubectl edit` to add them. You can use any password generator to generate a password for it. Make sure to base64 encode the data before you enter it in the secret.

Loki upgrade retries exhausted

While running `flux get helmrelease -A`, you'll see:

```
oas          loki                    False   upgrade retries exhausted          2.5.2    ␣
↪False
```

This happens sometimes because Loki takes a long time to upgrade. Usually it is solved by running `flux reconcile hr loki -n oas` again.

7.2 Upgrading to 0.6.0

A few things are important when upgrading to 0.6.0:

- We now use Flux 2 and the installation procedure has been overhauled. For this reason we advice you to set up a completely new cluster.
- Copy your configuration details from `settings.yaml` to a new `.flux.env`. See `install/.flux.env.example` and the *Installation overview* instructions for more information.

Please reach out to us if you are using, or plan to use OAS in production.

7.3 Upgrading from 0.4.0 to 0.5.0

Unfortunately we can't ensure a smooth upgrade for this version neither. Please read the section below on how to do an upgrade by installing a the new OAS version from scratch after backing up your data.

7.4 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:

```
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:
```

(continues on next page)

(continued from previous page)

```
    # 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.
    # 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'
  - 'promtail'
  - 'nextcloud'
  - 'prometheus'
  - 'rocketchat'
  - 'wordpress'
```

7.5 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 `pvc`s 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`

CHAPTER

EIGHT

CUSTOMIZING

In this guide we show how to customize your cluster installation, i.e. if you want to install additional applications, or change the configuration of extisting apps installed by OAS this is the right place. Customizing other parts of your cluster is possible but not yet covered by this guide. This guide is written for users with advanced knowledge of the tools behind Openappstack, most importantly: Kubernetes, Helm, Ansible and Flux 2.

Warning: Customizing your OAS cluster could break your cluster in a way that it's not easy to recover. Please be aware of the potential risk when proceeding.

8.1 Prerequisites

- A functional OAS cluster installed following the Openappstack installation instructions

8.2 Customize OAS applications

Apps deployed by OAS are configured using helm values from templates in `flux2/apps/<application>/release.yaml`. It is possible to override values from the helmrelease by adding a custom `ConfigMap` or `Secret` to the cluster. The secret or configmap name is specified in the `valuesFrom` section of the `release.yaml` file. Read more in the Flux documentation

8.2.1 Example: Customize Nextcloud to work with staging certificates

Our CI pipeline works with staging certificates from Let's Encrypt, for that reason we need to allow insecure connections for the integration with ONLYOFFICE. You can find the file at `install/overrides/oas-nextcloud-override.yaml`.

To apply it, run the following commands:

```
# If you want to run this on your provisioning machine, tell kubectl to use
# your cluster:
export KUBECONFIG=$PWD/clusters/oas.example.org/kube_config_cluster.yml
# Check the current state of the helmrelease you want to modify:
flux get helmrelease -A
# If all is OK, make sure to apply your override configmap or secret in the
# same namespace as your helmrelease with the '-n' argument
kubectl apply \
```

(continues on next page)

(continued from previous page)

```
  -n oas-apps \
  -f ./install/overrides/oas-nextcloud-override.yaml
```

8.3 Adding custom apps to the cluster

OpenAppStack uses Flux 2 to install and auto-update applications. If you want to install extra applications or other things into the Kubernetes cluster, our advice would be to set up your own GitRepository and add it to the Flux system.

When you do this, you are fully responsible for keeping those applications secure and updated. If any of those applications is insecure, that can also invalidate the security of your OpenAppStack applications, because they are part of the same cluster and VPS.

Refer to the Flux 2 documentation for more information.

CHAPTER

NINE

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.

We would love to hear from you! If you have problems, please create an issue in our issue tracker or reach out as described on our contact page. We want to be in communication with our users, and we want to help you if you run into problems.

9.1 Known issues

If you run into a problem, please check our issue tracker to see if others have run into the same problem. We might have suggested a workaround or temporary solution in one of our issues. If your problems is not described in an issue, please open a new one so we can solve the problems you encounter.

9.2 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](https://testinfra.readthedocs.io/en/latest/) tests, and [Taiko](https://taiko.dev) tests.

9.2.1 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 returns 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. Our testinfra tests are a combination of blackbox and clearbox tests.

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

```
cd test
```

To run the test against your cluster, first export the `CLUSTER_DIR` environment variable with the location of your cluster config directory (replace `oas.example.org` with your cluster name):

```
export CLUSTER_DIR="../clusters/oas.example.org"
```

Run all tests

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

Test all applications

This will check for:

- The applications return proper certificates
- All helm releases are successfully installed
- All app pods are running and healthy (this test includes all optional applications)

These tests includes all optional applications and will fail for optional applications that are not installed.

```
pytest -s -m 'app' --connection=ansible --ansible-inventory=${CLUSTER_DIR}/inventory.yml␣
↪--hosts='ansible://*'
```

Tests a specific application

```
pytest -s -m 'app' --app="wordpress" --connection=ansible --ansible-inventory=${CLUSTER_
↪DIR}/inventory.yml --hosts='ansible://*'
```

Known Issues

The Default ssh backend for testinfra tests is `paramiko`, which doesn't work out 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://...`

9.2.2 Taiko tests

Taiko 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

Install [Taiko](https://taiko.dev) on your provisioning machine:

```
npm install -g taiko
```

Run Taiko tests

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

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

To learn more about the *test* subcommand, run:

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

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

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

9.3 Advanced usage

9.3.1 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
```

9.3.2 Taiko tests

Using Taiko without the OpenAppStack CLI

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

For nextcloud & onlyoffice tests:

```
export DOMAIN='oas.example.net'
export SSO_USERNAME='user1'
export SSO_USER_PW='...'
export TAIKO_TESTS='nextcloud'
taiko --observe taiko-tests.js
```

You can replace `nextcloud` with `grafana` or `wordpress` to test the other applications, or with `all` to test all applications.

9.4 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`.
- `flux` is the flux command line tool

9.5 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 oas.example.org info
```

Look for these lines:

```
To use kubectl with this cluster, copy-paste this in your terminal:
export KUBECONFIG=/home/you/projects/openappstack/clusters/oas.example.org/kube_config_
↪cluster.yml
```

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

9.6 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.oas.example.org), a useful resource for troubleshooting is the official cert-manager Troubleshooting Issuing ACME Certificates documentation. First, try this:

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

Is your cluster using the live ACME server?

```
$ kubectl get clusterissuers -o yaml | grep 'server:'
```

Should return `server: https://acme-v02.api.letsencrypt.org/directory` and not something with the word *staging* in it.

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

```
$ kubectl -n cert-manager get pods
```

Cert-manager uses a "custom resource" to keep track of your certificates, so you can also check the status of your certificates by running:

This returns all the certificates for all applications on your system. The command includes example output of healthy certificates.

```
$ kubectl get certificates -A
NAMESPACE   NAME                            READY   SECRET                          AGE
oas         hydra-public.tls                True    hydra-public.tls                14d
oas         single-sign-on-userpanel.tls    True    single-sign-on-userpanel.tls    14d
oas-apps    oas-nextcloud-files             True    oas-nextcloud-files             14d
oas-apps    oas-nextcloud-office            True    oas-nextcloud-office            14d
oas         grafana-tls                     True    grafana-tls                     13d
oas         alertmanager-tls                True    alertmanager-tls                13d
oas         prometheus-tls                  True    prometheus-tls                  13d
```

If there are problems, you can check for the specific `certificaterequests`:

```
$ kubectl get certificaterequests -A
```

If you still need more information, you can dig into the logs of the `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.

9.6.1 Example

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.oas.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 its 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
```

9.7 Application installation or upgrade failures

Application installations and upgrades are managed by flux. Flux uses `helm-controller` to install and upgrade applications with `helm charts`.

An application installed with Flux consists of a `kustomization`. This is a resource that defines where the information about the application is stored in our Git repository. The `kustomization` contains a `helmrelease`, which is an object that represents an installation of a Helm chart. Read more about the difference between `kustomizations` and `helmreleases` in the flux documentation

To find out if all `kustomizations` have been applied correctly, run the following flux command in your cluster:

```
cluster$ flux get kustomizations -A
```

If all your `kustomizations` are in a `Ready` state, take a look at your `helmreleases`:

```
cluster$ flux get helmreleases -A
```

Often, you can resolve complications with `kustomizations` or `helmreleases` by telling Flux to *reconcile* them:

```
cluster$ flux reconcile helmrelease nextcloud
```

Will make sure that the Nextcloud `helmrelease` gets brought into a state that our OpenAppStack wants it to be in.

9.8 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:

Warning: You will lose all your data! This completely destroys OpenAppStack and takes everything offline. If you chose to do this, you will need to re-install OpenAppStack and make sure that your data is stored somewhere other than the VPS that runs OpenAppStack.

```
cluster$ /usr/local/bin/k3s-killall.sh
cluster$ systemctl disable k3s
cluster$ rm -rf /var/lib/{rancher,OpenAppStack,kubelet,cni,docker,etcd} /etc/{kubernetes,
↪rancher} /var/log/{OpenAppStack,containers,pods} /tmp/k3s /etc/systemd/system/k3s.
↪service
cluster$ systemctl reboot
```

CHAPTER

TEN

SECURITY

10.1 Access control

By default, the applications on your OAS cluster will be exposed to the whole internet (although they are password protected).

If you like to limit who can access your cluster resources you can configure the OAS ingress (ingress-nginx) to only accept connections from a certain IP address or range.

Follow the *Customize OAS applications* instructions, but use the following secret as `install/overrides/oas-nginx-override.yml` and apply the secret in the `oas` namespace instead of `oas-apps`. Replace the source range with the IP address ranges you want to allow.

```
---
apiVersion: v1
kind: secret
metadata:
  name: oas-nginx-override
data:
  values.yaml: |
    controller:
      config:
        # https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/
↪annotations/#whitelist-source-range
        # comma separated list of CIDRs, e.g. 10.0.0.0/24,172.10.0.1.
        whitelist-source-range: 1.2.3.4/24
```

CHAPTER

ELEVEN

REFERENCE DOCUMENTATION

11.1 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/rancher/k3s`: this contains the files related to your "Kubernetes" cluster.
  - The kubectl configuration file is located at `/etc/rancher/k3s/k3s.yaml`

CHAPTER

TWELVE

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

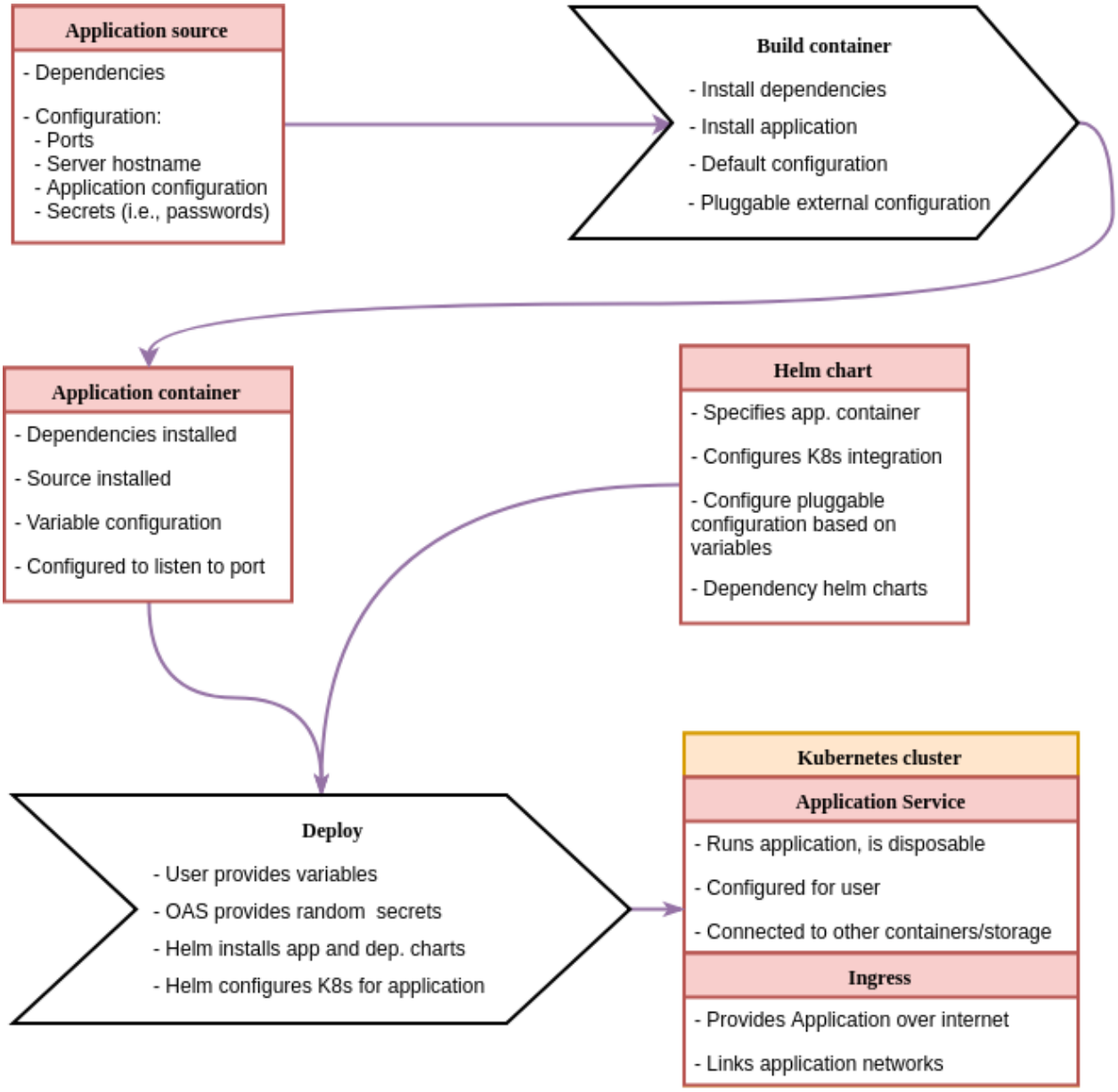
THIRTEEN

OPENAPPSTACK DESIGN

This article covers the basic design of OpenAppStack.

13.1 Application build pipeline

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



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.

13.2 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.

13.3 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.

13.3.1 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.

13.4 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.

13.5 Automatic updates

OpenAppStack has an auto-update mechanism that performs unattended upgrades to applications. Flux 2 is the system running in the cluster that is responsible for these updates.

Flux 2 tracks all the files in the `flux2` directory of the OpenAppStack code repository. Once changes are pushd the branch that Flux tracks, the changes are applied to the cluster.

We use Flux 2 in "read only" mode, which means that your OpenAppStack cluster does not push changes to our Git repository. You can read more about Flux 2 and its components in the flux 2 documentation.