Stackspin

Release 2.1

Greenhost

Nov 15, 2022

INSTALLATION

Stackspin is a platform that offers 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 Stackspin, so they can keep control over the data that gets entered into these applications.

Stackspin is:

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

For more information, go to the Stackspin website.

CHAPTER

ONE

INSTALLATION OVERVIEW

Warning: When you install Stackspin on a server, the installation process will make substantial changes to the server's configuration, so please do not use a server that performs any other functions that should not be disrupted.

This page will guide you through the installation of the "Stackspin CLI". The CLI can be used to set up Stackspin on a different machine that is connected to the internet. It also functions as a starting point for running tests.

1.1 Preparing the provisioning machine

You should have a trusted machine to run the installer on, this can be your laptop or pc. In this guide, we call it the *provisioning machine*. This machine does not have to be fast, it will only run a few lightweight scripts. All commands in these installation instructions need to be run on the provisioning machine, except when a command starts with `cluster$`

The following software setup is required **on your provisioning machine**:

- You need Python 3 with its development files, Pip and Git installed. On Debian/Ubuntu: `apt install python3-pip python3-dev git`
- We recommend using a python virtualenv to make sure Stackspin does not change any of your other python projects. Install virtualenv by running `pip3 install --user venv` or `apt install python3-venv`.
- You need a recent version of Pip. If you run into problems during `pip install` you might need to run `pip install --upgrade pip`.
- `kubectl` (installation instructions)
- `flux` version 0.36.0 (Download flux_0.36.0_linux_amd64.tar.gz or use the official installation script: `curl -s https://fluxcd.io/install.sh | sudo FLUX_VERSION=0.36.0 bash`)

Then you can clone the Stackspin git repository and checkout the latest release branch (currently `v2`):

```
$ git clone -b v2 https://open.greenhost.net/stackspin/stackspin.git
$ cd stackspin
```

Apart from the previously installed software, Stackspin will only modify in the `clusters` subdirectory of the repository. So multiple provisioning machines can administer the same instance so long as that subdirectory is synced between them.

1.2 Create a python virtual environment (Optional)

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
```

1.3 Install the requirements

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

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

1.4 Stackspin CLI client usage

Now you can run the Stackspin CLI as follows:

```
$ python -m stackspin 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 `stackspin.example.org` as the cluster name. Try it out by running:

```
$ python -m stackspin stackspin.example.org --help
```

The next chapter, *Create and configure a kubernetes cluster* will explain setting up and/or configuring a Kubernetes cluster to install Stackspin on.

CHAPTER

TWO

CREATE AND CONFIGURE A KUBERNETES CLUSTER

Below, choose whether you want to install Stackspin on a VPS or an existing Kubernetes cluster. We recommend manually installing it on a VPS, this is the most tested solution.

A. Install on a VPS

We recommend manually installing Stackspin on a Greenhost VPS as explained below. However, if you plan on installing many Stackspin instances, click the tab *"ADVANCED: Use the Greenhost API"* to learn how to automatically create VPSs and DNS records and install Stackspin on them.

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

For Stackspin 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.

Manually install on a VPS

Prerequisites

These instructions assume you have a VPS or other kind of server ready that meets the following requirements:

- Debian 11 (bullseye) installed
- A public IPv4 address (Dual-Stack/IPv6 only is not yet supported)
- The ability to create DNS records for this IP
- The VPS meets the system requirements below
- At least 35GB of disk space for installation and monitoring data, plus more for application data.
- Root ssh access

System requirements

The system requirements depend on which applications you want to install into Stackspin. The following table shows the minimum requirements for each application (1000m CPU means 1 core). Sum up the requirements of all the applications you wish to install, together with the "Core components". That is the amount of memory and CPU you need to successfully run Stackspin. If you try to run Stackspin on machines with less than the recommended system requirements, you will notice that some "Pods" (application components) refuse to start.

Table 1: Stackspin application resource requirements

Application	Memory	CPU cores
Core components	5.0 GiB	1000m
Zulip	2.9 GiB	215m
WordPress	890 MiB	100m
Wekan	620 MiB	50m
Velero	625 MiB	20m
Nextcloud	2.4 GiB	360m
Monitoring	2.3 GiB	180m

This guide assumes the VPS's IP address is `1.2.3.4` and we install on the domain `stackspin.example.org`. Substitute these two variables with your IP address and the (sub)domain you want Stackpin to be available on.

Step 1: Create cluster configuration

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

```
$ python -m stackspin stackspin.example.org create --help
```

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

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

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

- the first, positional argument `stackspin.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/stackspin.example.org` on your provisioning machine.

ADVANCED: Use the Greenhost API

Prerequisites

Before you can start, you need the following things:

1. An API key with Customer rights.

   Make sure the Stackspin CLI can use your token by storing it in a `COSMOS_API_TOKEN` variable.

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

   In *the same terminal*, you can now use the `create` subcommand

2. An SSH key ID from the Cosmos Service Centre's "VPS Cloud" section. This SSH key will get Root access on the VPS.

   - 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.

- You can also use the API to list ssh keys and find it there. Read the Greenhost API documentation for more information*

3. The ability to create DNS records that point to the VPS that gets created

Step 1: Create VPS and cluster configuration

We'll use the `create` subcommand of the Stackspin CLI to create a VPS and a corresponding configuration file on the provisioning machine.

Here is an example of a complete creation command:

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

Let's break down the arguments:

- `--create-droplet`: Use the Greenhost API to create a new VPS
- `--create-hostname stackspin.example.org`: Create a VPS with hostname `stackspin.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: DNS Configuration*. The following records are automatically created:
  - An `A` record `stackspin.example.org` pointing to the VPS's IP address
  - A `CNAME` record `*.stackspin.example.org` pointing to `stackspin.example.org`.
- `--subdomain stackspin`: Only needed when you use `--create-domain-records` so the Greenhost API can find your domain. If you use the `--subdomain` argument, the positional argument **must not** include the subdomain (i.e., in the example command the positional argument is `example.org` and not `subdomain.example.org`).

  If you want to host Stackspin on a root domain, you can omit the `subdomain` argument.

Step 2: DNS Configuration

You need to configure some DNS records to make Stackspin available over the internet. These DNS records are not optional, because quite a few of them are actually also used by the applications to talk to the other applications (e.g., the single sign-on provider is used by all applications, or Nextcloud embeds ONLYOFFICE).

The following DNS records need to be set for the dashboard and the single sign-on system:

- An `A` record `dashboard.example.org` pointing to the cluster's IP address,

If you want to guarantee Stackspin will work for all applications you install afterwards, it needs a wildcard DNS record `*.example.org` that points to your cluster's IP address.

Advanced Setup

If you do not want to or cannot set a wildcard DNS rule, you will need the following rule:

- A `CNAME` record `sso.example.org` pointing to `dashboard.example.org`

And similar records for these applications, if you install them:

Monitoring

- `grafana.example.org` is used for Grafana, the monitoring dashboard
- `alertmanager.example.org` (optional) is used for the Alertmanager dashboard
- `prometheus.example.org` (optional) is used for the Prometheus dashboard

Nextcloud and ONLYOFFICE

- `files.example.org` is used for Nextcloud
- `office.example.org` is used for ONLYOFFICE

Wekan

- `wekan.example.org` is used for Wekan

WordPress

- `www.example.org` is the default WordPress domain
- `example.org` (Optional) – Redirects to `www.`

Zulip

- `zulip.example.org` is used for Zulip

Note: Follow *Custom domain names for applications* for instructions on using custom domain names.

Step 3: Create Kubernetes cluster

Before you continue, make sure your DNS configuration is propagated. You can do this by making sure `ping` shows your VPS's IP address:

```
$ ping stackspin.example.org
Example output of the command:
PING stackspin.example.org (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4 (1.2.3.4): icmp_seq=1 ttl=62 time=9.43 ms
64 bytes from 1.2.3.4 (1.2.3.4): icmp_seq=2 ttl=62 time=52.5 ms
...
```

If you've never logged in to this machine before, you might need to accept the SSH hostkey before you can start the installation process. To be sure, log in to your VPS once before you continue:

```
$ ssh root@stackspin.example.org
```

If necessary, check your hostkey fingerprint and type `yes` to continue. You should get a shell on your VPS. Type `exit` to quit the shell and continue on your provisioning machine.

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

```
$ python -m stackspin stackspin.example.org install
```

This will take a few minutes. It tries to connect to your VPS as the root user, then installs k3s. In addition it installs command line tools kubectl (Kubernetes cli tool), flux (used for automated updates) and velero (Kubernetes resources and persistent volumes backup) so you can manage the Kubernetes cluster from a shell on the VPS.

Note: It is possible to re-run the `install` command with a newer version of the installation script. Refer to the *Upgrades* for details.

B. Install on existing cluster

Below, we describe how you can install Stackspin on an existing Kubernetes cluster. This process can be followed if you have a "managed Kubernetes" setup, or if you want to use a Kubernetes version other than K3s.

Note:

- Installation on an existing Kubernetes cluster is still experimental and might not work depending on the setup of your cluster/cloud provider. We are happy to receive feedback from your experiences but at the moment we cannot make any guarantees!

Prerequisites

- A single-node Kubernetes cluster (multi-node support has not been tested by the Stackspin team)
- A `kube_config.yml` file for API access

System requirements

The system requirements depend on which applications you want to install into Stackspin. The following table shows the minimum requirements for each application (1000m CPU means 1 core). Sum up the requirements of all the applications you wish to install, together with the "Core components". That is the amount of memory and CPU you need to successfully run Stackspin. If you try to run Stackspin on machines with less than the recommended system requirements, you will notice that some "Pods" (application components) refuse to start.

Table 2: Stackspin application resource requirements

Application	Memory	CPU cores
Core components	5.0 GiB	1000m
Zulip	2.9 GiB	215m
WordPress	890 MiB	100m
Wekan	620 MiB	50m
Velero	625 MiB	20m
Nextcloud	2.4 GiB	360m
Monitoring	2.3 GiB	180m

Configure DNS

You need to configure some DNS records to make Stackspin available over the internet. These DNS records are not optional, because quite a few of them are actually also used by the applications to talk to the other applications (e.g., the single sign-on provider is used by all applications, or Nextcloud embeds ONLYOFFICE).

The following DNS records need to be set for the dashboard and the single sign-on system:

- An `A` record `dashboard.example.org` pointing to the cluster's IP address,

If you want to guarantee Stackspin will work for all applications you install afterwards, it needs a wildcard DNS record `*.example.org` that points to your cluster's IP address.

Advanced Setup

If you do not want to or cannot set a wildcard DNS rule, you will need the following rule:

- A `CNAME` record `sso.example.org` pointing to `dashboard.example.org`

And similar records for these applications, if you install them:

Monitoring

- `grafana.example.org` is used for Grafana, the monitoring dashboard

- `alertmanager.example.org` (optional) is used for the Alertmanager dashboard
- `prometheus.example.org` (optional) is used for the Prometheus dashboard

Nextcloud and ONLYOFFICE

- `files.example.org` is used for Nextcloud
- `office.example.org` is used for ONLYOFFICE

Wekan

- `wekan.example.org` is used for Wekan

WordPress

- `www.example.org` is the default WordPress domain
- `example.org` (Optional) – Redirects to `www.`

Zulip

- `zulip.example.org` is used for Zulip

Note: Follow *Custom domain names for applications* for instructions on using custom domain names.

Setup

- Create a directory containing your cluster configuration, i.e. `mkdir -p clusters/stackspin.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/stackspin.example.org/kube_config_cluster.yml`

Now that you have a Kubernetes cluster running you can continue to *Install Stackspin*.

CHAPTER

THREE

INSTALL STACKSPIN

This guide explains how to install Stackspin on the Kubernetes cluster you have configured in the previous step.

3.1 Step 1: Flux configuration

Flux will run inside your Stackspin cluster to install and upgrade applications. It needs to be configured once, using the `flux` command line tool and scripts provided by us in the Stackspin repository.

3.1.1 Configuration

Copy the file `install/.flux.env.example` to your cluster dir `clusters/stackspin.example.org/.flux.env`. This file contains the last bit of information you need to configure. Make sure not to put any quotes around your values, because that can make the resulting yaml files invalid. Also, using a dollar sign `$` or double quote `"` may lead to problems, so please avoid using those characters in the values.

Cluster information

```
# The IP address of your cluster
ip_address=1.2.3.4
# The FQDN of your cluster
domain=stackspin.example.org
# The system administrator's email address. Alerts are sent to this address
admin_email=admin@example.org
```

Outgoing email

Stackspin uses SMTP to send emails. This is essential for finishing account setups with password recovery links. Additionally, apps like Nextcloud, Zulip and Wordpress will be able to send email notifications from the email address configured here. You also may receive alert notification emails from Stackspin's monitoring system. See *Email alerts* for more information about those alerts, especially during installation.

Because Stackspin does not include an email server, you need to obtain SMTP configuration details from your (external) email provider.

Backups with Velero (Optional)

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 resources in your cluster; 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, Zulip 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 Stackspin, like programs installed via `apt`, or data added to the VPS disk not through Stackspin.

To configure Velero, edit the file `clusters/stackspin.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.

For information on how to use Velero with Stackspin, please see *Backup*.

3.2 Step 2: 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 `stackspin.example.org` with your domain.

We will use this variable in the following commands, set it to your cluster directory.

```
$ export CLUSTER_DIR=$PWD/clusters/stackspin.example.org
```

Make sure your `virtualenv` is activated.

```
$ . env/bin/activate
```

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/stackspin-cluster-variables created
```

Next, run:

```
$ ./install/install-stackspin.sh
```

This installs the *core* of Stackspin into your cluster. To see what's included, check the `flux2/core/base` folder in the git repository.

You can run the following command to view the status of the installation of all components:

```
$ watch flux get kustomization
```

3.3 Step 3: Install additional applications

After the script has completed, and the `dashboard` kustomization shows as "Ready" in the `flux get kustomization` command, you can use the Dashboard CLI to install applications.

- Monitoring stack (Prometheus, Grafana) `kubectl exec -n stackspin deploy/dashboard -c backend -- flask cli app install monitoring`
- Nextcloud and Onlyoffice with `kubectl exec -n stackspin deploy/dashboard -c backend -- flask cli app install nextcloud`
- Zulip chat with `kubectl exec -n stackspin deploy/dashboard -c backend -- flask cli app install zulip`
- Wekan with `kubectl exec -n stackspin deploy/dashboard -c backend -- flask cli app install wekan`
- WordPress with `kubectl exec -n stackspin deploy/dashboard -c backend -- flask cli app install wordpress`
- Velero with `kubectl exec -n stackspin deploy/dashboard -c backend -- flask cli app install velero` (only if you have configured it in *Backups with Velero (Optional)*).

When the command completes, the application installation may still be running on the Stackspin cluster. You can monitor the progress by running `kubectl exec -n stackspin deploy/dashboard -c backend -- flask cli app status <app_name>`

3.4 Step 4: Validate setup

Once the installation has been completed, you can log in on https://dashboard.stackspin.example.org (as always: replace *stackspin.example.org* with your domain). To get your login details, run:

```
$ python -m stackspin stackspin.example.org admin-credentials
```

Additionally, because Stackspin is still under development, we would like you to follow our *Test your installation* to make sure that the setup process went well.

3.5 Step 5: Let us know!

We would love to hear about your experience installing Stackspin. 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 Stackspin so far. We want to be in communication with our users, and we want to help you if you run into problems.

CHAPTER

FOUR

TEST YOUR INSTALLATION

Great that you want to take Stackspin 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.

4.1 Stackspin installation

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

4.2 Command line tests

Please refer to *Testing* which checks the overall functionality of your cluster and include the output in your feedback.

4.3 Testing Instructions for all apps

Please log in on https://dashboard.stackspin.example.org using single sign-on with your admin user email address 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 log in again as the non-admin single-sign-on user you created earlier in the Stackspin admin panel. You should *not* have any admin privileges now.

4.4 Testing Instructions for specific apps

4.4.1 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 Stackspin`. Please try logging in with your admin account and configure the email settings as shown in the Usage doc. After that please log in 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*.

4.4.2 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.

4.4.3 Wordpress

Using the link in the dashboard app, please try to log in to WordPress as the user you created earlier by pressing "Log in" and using the `Login with OpenID Connect` button.

Your user needs "Admin" privileges to edit the website.

4.5 Providing feedback

If you have not done so already, please create an account on https://open.greenhost.net (or log in 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 Stackspin.

CHAPTER

FIVE

MANAGE STACKSPIN

5.1 Logging in

Stackspin includes a Single sign-on system. That means that you log in once, and then all the applications included in Stackspin will know who you are, without you having to enter a password again.

Stackspin includes a Dashboard to manage these single sign-on users. The Dashboard is the starting point for all Stackspin users. To use the Dashboard, log in to https://dashboard.stackspin.example.org. Here you can create, edit and delete users, as well as see all the applications that run on your Stackspin cluster.

Your first login can happen in one of two ways:

1. Use the email address that was used during setup.

   The admin email address has an account already. If you have access to its mailbox, press "Sign in" in the browser and then press "Forget password?" to start a password recovery procedure.

2. If you do not have access to the administrator email address, ask your system administrator to run the following command on their "provisioning machine" and share the admin credentials with you:

   ```
   $ python3 -m stackspin stackspin.example.org admin-credentials
   ```

   Note: these are the setup credentials, they will not be updated when you change the admin user's password.

5.2 Login and logout limitations

Our single sign-on system uses an authentication protocol called OpenID Connect. This works the same way as "Log in with Google" buttons you might be familiar with, but don't worry: instead of contacting Google, it uses Stackspin's internal user database.

5.2.1 Limitations on login

The Stackspin dashboard allows you to define user privileges in two levels: "Admin" or "User". When admin privileges are given to a user for a specific application, the user needs to log out and into that application again. Some applications, like WordPress and Zulip, have specific limitations. These are listed on the *Applications* page.

5.2.2 Limitations on logout

Users can log out by using the "Logout" button in any application. However, when they do so, they are only logged out of that specific application and the single sign-on system. So if users want to log out of Zulip as well as WordPress, they need to go to both applications, and click the logout button there. "Single sign-off" – logging out of all applications at once – will be implemented in a future version of Stackspin.

5.2.3 Limitations on user management

It is important to realize that although Stackspin has a central user management system, the applications included in stackspin also have their own user management systems. Users created inside applications are not synchronized to the single sign-on system. For that reason, we recommend you always create new users inside the Dashboard.

5.3 Dashboard

After logging in, you will see an overview of all the installed applications. The "Launch app" button on this screen will bring you to the application's screen. If see a login screen, use the "log in with Stackspin" button to log in with your current user. You will not have to enter credentials, because you are already logged in via the Dashboard.

5.4 Users

The Dashboard also features a "Users" tab, where you can manage the Stackspin users. At the moment, all users will be able to log into all applications on your cluster with the "Log in with Stackspin" button. More fine grained control will be added in the future.

To create a new user, click the "Add new user" button. You need to specify the user's (full) name under "Name" and a valid email address under "Email".

Once you have created the user, they will not be automatically notified - please contact individual users and let them know they can finish their account setup by opening any Stackspin-related URL and clicking the "Forget password?" button on the "Login with Stackspin" screen. This will let them set up their password for the first time, and their account will be active from that moment.

Note: The Dashboard currently allows all users to manage the other users. In the (near) future, only users with "admin" privileges will be allowed to add, edit and remove other users.

Only add users for people that you trust.

To edit a user's name or email address, or delete a user, press the "Configure" button on the right side in the user table.

If you want to delete several users, you can select them in the Users overview, using the select buttons on the left side. Then press the "Delete" button on the top right of the table.

Note: A note on deleting users

Deleting users from the Dashboard makes sure they can't log into applications anymore. However, the dashboard cannot control the user's data inside other applications.

To completely remove users from all the applications, you will need to remove them from inside the application's admin settings as well. Refer to the documentation of the specific application to see how that should be done.

If you only remove a user from the dashboard, but not from the application, it is possible that this user keeps access to the application through sessions inside application clients (for example the Zulip or Nextcloud desktop clients). These sessions can also typically be ended through the application's admin interface.

5.5 Plugins

Most of the applications installed with Stackspin include a plugin system. Because most plugins are not developed by the applications' core developers, it is important to realize that by installing them, you place your trust into third party developers.

In the case of Nextcloud, to increase security and stability, we have decided to disable the *app store* completely. The Stackspin team curates a list of plugins that is pre-installed and can be enabled. Other plugins cannot be enabled at the moment. If you need a specific plugin, please reach out to us and we will research whether we can add it to the list of supported plugins.

For other applications, like WordPress, we did not take such a radical approach. Especially for WordPress using at least a few plugins is considered the norm. However, we advice caution when considering to add plugins to your applications.

CHAPTER

SIX

APPLICATIONS

6.1 Dashboard

The Dashboard is always installed into a Stackspin cluster. It is an application developed specifically for Stackspin. It allows you to manage Stackspin's central user database. Soon, the dashboard will also allow you to configure, install and remove applications and set access rights for users.

Administrators can read how to use the Dashboard in *Manage Stackspin*

6.2 Grafana

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

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.

6.3 Nextcloud

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

6.3.1 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.

6.3.2 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.

6.3.3 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.

6.4 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.

6.5 Zulip

Zulip is a team chat application.

6.5.1 Roles management

Unfortunately, Zulip does not yet listen to the Stackspin roles, so once you have installed Zulip, you need to make one of your users the owner. That allows this user to manage Zulip's settings from inside the application. This owner can also edit other users' roles.

For this, you have to replace <your email> in the the following command and run it on your cluster:

```
` kubectl exec -n stackspin-apps zulip-0 -c zulip -- sudo -u zulip /home/zulip/
deployments/current/manage.py change_user_role <your email> owner --realm 2 `
```

6.5.2 Single sign-off

It is currently impossible to log out of the single sign-on system by logging out of Zulip. This is due to a limitation in Zulip. This means that if you want to log into Zulip under a different account, you need to logout through the Dashboard or another application first. You can also visit `https://sso.stackspin.example.com/oauth2/sessions/logout` in your browser to end the SSO session.

6.6 Wekan

Wekan is a Kanban board application.

6.6.1 Single sign-off

It is currently impossible to log out of the single sign-on system by logging out of Wekan. This is due to a limitation in Wekan. This means that if you want to log into Wekan under a different account, you need to logout through the Dashboard or another application first. You can also visit `https://sso.stackspin.example.com/oauth2/sessions/logout` in your browser to end the SSO session.

6.7 WordPress

WordPress is a website content management system.

6.7.1 Roles management

If you want admin privileges inside WordPress, you need to set this in the Dashboard. Then, you need to log in to WordPress again.

If this is your first time logging in to WordPress, you have to log out and in again for your admin privileges to be applied.

For more information and technical details, refer to the issue in our issue tracker.

CHAPTER

SEVEN

MIGRATING APPLICATION DATA

When you first start using Stackspin, you probably already have data that you do not want to lose. This guide will focus on a use-case where you were already running the applications that are included with Stackspin. If you are migrating from different applications, chances are you can still use similar methods to import the data into Stackspin-based applications. For example, Zulip chat includes import functionality for Mattermost, Rocket.Chat, Gitter and Slack.

Before you start any migration, make sure you use a machine that has `kubectl` access to the machine you want to recover Zulip on. This is achieved by running the following on your provisioning machine:

```
export KUBECONFIG=$PWD/clusters/<your_cluster>/kube_config_cluster.yml``
```

Alternatively, you can run the commands from this guide in an SSH session on the machine you run Stackspin on.

Also make sure you have already installed the apps you want to migrate. Follow *Step 3: Install additional applications* if you have not done so yet.

7.1 Nextcloud

To export and import your Nextcloud data into a Nextcloud running on Stackspin, follow the instructions in the Nextcloud documentation. There are a few things that go slightly differently, because in Stackspin you need to run the commands through `kubectl`.

In some commands, you need the Nextcloud pod name that you can get by running:

```
kubectl get pods -n stackspin-apps -l app.kubernetes.io/name=nextcloud
```

Note: the correct pod is the one that does not contain `-cron`.

- In the restore folders step, instead of the command `rsync -Aax nextcloud-dirbkp/ nextcloud/`, you need to use `kubectl cp`:

  ```
  # replace nc-nextcloud-xxxxxxxxx-xxxxx with your NC pod name
  kubectl -n stackspin-apps cp nextcloud-dirbkp nc-nextcloud-xxxxxxxxx-xxxxx:/var/www/
  ```

  Note that `kubectl cp`, unlike `rsync` does not overwrite directories. So you need to use `kubectl exec` into the pod to override the data directory.

  For example, you could run the following to backup the current Nextcloud `html` directory and overwrite it with your whole Nextcloud backup.

```
kubectl -n stackspin-apps exec deploy/nc-nextcloud -it -- /bin/bash
cd /var/www
mv html html_backup
mv nextcloud-dirbkp html
```

- In the restore database step, follow the steps for MySQL (we use MariaDB, which is equivalent). Instead of running `mysql $command`, run `kubectl -n stackspin-apps exec nc-mariadb-0 -it -- mysql $command`. You can also ommit the `-h [server]` part, it will default to the correct server.

Go to https://files.stackspin.example.com and log in using the *Log in with Stackspin* button.

Because Stackspin uses an OIDC provider, it is likely that Nextcloud will not link the existing user from your previous server to your Stackspin account.

If that happens, you can execute the following commands to transfer your old files to your new account:

```
# Enter the Nextcloud container
kubectl -n stackspin-apps exec deploy/nc-nextcloud -it -- /bin/bash
# Change to the www-data user
su -s /bin/bash www-data
# Transfer ownership of your files to a new user
php occ files:transfer-ownership <source-user> <destination-user>

# You can use `php occ files:transfer-ownership --help` to find out more
# about this command
```

7.2 Wekan

Follow the backup instructions on the Wekan wiki to back-up your current Wekan instance.

These restore instructions are based on the Docker restore instructions, but adjusted for Stackspin. This guide assumes you have a MongoDB dump called `dump` in your current directory.

Copy the dump into the Wekan MongoDB pod:

```
kubectl cp dump stackspin-apps/wekan-mongodb-0:/bitnami/mongodb/data
```

Then, enter the mongodb pod, and execute the import command. NOTE: this will remove all the data in your current Wekan instance.

```
kubectl exec -n stackspin-apps wekan-mongodb-0 -it -- /bin/bash
cd /bitnami/mongodb/data
wekan-db mongorestore --drop --dir=/data/dump
```

You should now be able to use Wekan with your old data on https://wekan.stackspin.example.com.

7.3 WordPress

To import an existing WordPress site into Stackspin, you will need:

- A database dump of your old website's database called `dump.sql`.
- A copy of the `wp-content` folder (usually located on the server at `/var/www/html/wp-content`) in a local directory called *wp-content*.

Make sure the copy of the wp-content directory and the database dump are made at the same time.

We will go through the following steps:

1. Customize the WordPress installation in Stackspin
2. Import the Database
3. Import the files from wp-content

7.3.1 Customize the WordPress installation

This is for users with Kubernetes experience only. You can probably skip this step.

If necessary, you can override the following variables to reflect your current site's configuration. Follow the *Customizing* guide in that case.

- Set `DB_PREFIX` to the database prefix that your current WordPress uses (default: `wp_`)
- Set `WP_VERSION` to the version of your current WordPress site.
- Check if your current WordPress uses Redis. This is usually the case when you have the *redis-cache* plugin installed. If so, enable redis by setting *redis.enabled* to *true*.

After installing, check if your site is available at www.stackspin.example.com.

7.3.2 Import the database

Make sure to remove all explicit URLs to your site from the database. If you host your site on `example.com`, WordPress tends to put links in the database as `https://example.com/<link to your post>` or `http://example.com/<link to your post>`. If you want to try this new deployment on a different domain, that will cause problems. To fix this, search for *https://example.com* in your database dump and remove it, leaving only the relative URL `/<link to your post>`. Repeat this step for `http://example.com`.

You need the WordPress database password. The easiest way to get it is to run:

```
helm get values -n stackspin-apps wordpress
```

And look for the value of `database.auth.password`

Now, import your database dump with the following command:

```
kubectl exec -n stackspin-apps -i wordpress-database-0 -- mysql -uwordpress -p<your␣
↪password> --database=wordpress_db < dump.sql
```

A breakdown of the command:

- `kubectl exec` means you want to execute a command in a container
- `-n stackspin-apps` uses the `stackspin-apps` namespace

- `-i` makes sure that `stdin` is passed to the container. This means that we can use `< dump.sql` to import the database export from your local system.
- We assume your database is running in a pod named `wordpress-master-mariadb-master-0`.
- `--` tells kubectl to stop interpreting command line arguments as kubectl arguments
- `mysql` executes the `mysql` command in the container, with the following arguments: - `-uwordpress` sets the MySQL user to wordpress - `-p` provides the password of the `wordpress` user - `--database=wordpress_db` selects the `wordpress_db` database.
- `< dump.sql` reads the file `dump.sql` and inserts it into the mysql command. Change this if your MySQL database dump has a different filename.

7.3.3 Import the files from wp-content/uploads

Similar to how you would normally use `scp` to copy files to a server, you can use `kubectl cp` to copy files to your wordpress pod. Make sure wp-content/uploads does not contain a *.htaccess* file, because one is mounted by this chart.

```
kubectl cp wp-content/ wordpress-master-0:/var/www/wp-content-mount
```

You'll have to change the ownership of the files to UID 33 (the *www-data* user in the WordPress container):

```
kubectl exec -it wordpress-master-0 -- chown -R 33:33 /var/www/wp-content-mount
```

Note: this will say `chown: changing ownership of '/var/www/wp-content/uploads/.htaccess': Read-only file system`. Don't worry, that's the mounted *.htaccess* file. All the other files' ownership *will* have changed.

Your WordPress site should now be available under https://www.stackspin.example.com

7.4 Zulip

To migrate your data to Zulip, make an export of your existing Zulip server, following the export instructions available here.

To recover the data, we are going to follow the import instructions available here, except we are going to do a few things differently:

- Because Stackspin has already fully installed Zulip, instead of "Skipping step 3 during installation", we will remove the realm that the Stackspin installation creates.
- Do not edit `settings.py` in a Stackspin deployment of Zulip. That file is generated by the Zulip startup script. If you need to change any settings, refer to the *Customizing* section of the documentation.

Copy your Zulip export to the Zulip pod by using `kubectl cp`, then enter a shell in your Zulip pod with `kubectl exec` (replace the filename with the name of your export).

```
kubectl cp zulip-export-xxxxxxxx.tar.gz stackspin-apps/zulip-0:/home/zulip/deployments/
↪current/
kubectl exec -n stackspin-apps zulip-0 -it -- /bin/bash
```

Inside the pod, change to the `zulip` user and access the current deployment folder. Then unpack the zulip export `.tar.gz`.

```
su zulip
cd ~/deployments/current
tar -xf zulip-export-xxxxxxxx.tar.gz
```

We start by removing the realm that was created by the Stackspin installation. NOTE: If you have already used Zulip on Stackspin, this will delete all the data!

```
./manage.py delete_realm -r ''
```

The command will ask you to enter the name of the realm. In our case, the name is an empty string, so just press enter. Zulip should now say something like:

```
# Realm has been successfully permanently deleted.
```

Now, you can import the data from your Zulip export:

```
./manage.py import '' zulip-export-xxxxxxxx
```

CHAPTER

EIGHT

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

CHAPTER

NINE

MONITORING

For monitoring your Stackspin cluster we included the kube-prometheus-stack helm chart, which bundles the applications Grafana, Prometheus and Alertmanager, and also includes pre-configured Prometheus alerts and Grafana dashboards.

9.1 Grafana

Grafana can be accessed by clicking on the `Monitoring` icon in the `Utilities` Section of the dashboard. Use Stackspin single sign-on to login.

9.1.1 Dashboards

Browse through the pre-configured dashboards to explore metrics of your Stackspin cluster. Describing every dashboard would be too much here, reach out for us if you don't find what you are looking for.

9.2 Prometheus

Prometheus can be reached by adding `prometheus.` in front of your cluster domain, i.e. `https://prometheus.stackspin.example.org`. Until we configure single sign-on for prometheus you need to login using basic auth. The user name is `admin`, the password can get retrieved by running

```
python -m stackspin CLUSTERNAME secrets | grep prometheus-basic-auth
```

9.3 Alertmanager

Alertmanager can be reached by adding `alertmanager.` in front of your cluster domain, i.e. `https://alertmanager.stackspin.example.org`. Until we configure single sign-on for prometheus you need to login using basic auth. The user name is `admin`, the password can get retrieved by running

```
python -m stackspin CLUSTERNAME secrets | grep alertmanager-basic-auth
```

9.4 Email alerts

From time to time you might get email alerts sent by Alertmanager to the email address you have set in the cluster configuration. Common alerts include (listed by the `alertname` references in the email body):

- **KubeJobCompletion**: A job did not complete successfully. Often happens during initial setup phase. If the alert persists use i.e. `kubectl -n stackspin-apps get jobs` to see all jobs in the `stackspin-apps` namespace and delete the failed job to silence the alert with i.e. `kubectl -n stackspin-apps delete job nc-nextcloud-cron-27444460`.
- **ReconciliationFailure**: A flux helmRelease could not get reconciled successfully. This also happen often during initial setup phase. It can have different root causes though. Use `flux -n stackspin-apps get helmreleases` to view the current state of all `helmReleases` in the `stackspin-apps` namespace. In case the `helmRelease` in question is stuck in a `install retries exhausted` or `upgrade retries exhausted` state you can force a reconciliation with

  ```
  flux -n stackspin-apps suspend helmrelease zulip
  flux -n stackspin-apps resume helmrelease zulip
  ```

  Depending on the underlying cause this will fix the `helmRelease` state or not. For more information on this issue see helmrelease upgrade retries exhausted regression

CHAPTER

TEN

MAINTENANCE

10.1 Backup

10.1.1 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 stackspin git repository. Although these files are not essential for your Stackspin cluster to continue functioning, you may want to back this folder up because it allows easy access to your cluster.

10.1.2 On your cluster

Stackspin supports using the program Velero to make backups of your Stackspin instance to external storage via the S3 API. See *Backups with Velero (Optional)* in the installation instructions for setup details.

For the maintenance operations described below – in particular, restoring backups – you need the `velero` client program installed, typically on your provisioning machine although you can also run it on the VPS if preferred. You may find it at Velero's github release page.

By default Velero 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/Stackspin/local-storage`.

10.2 Restore

Restoring from backups is a process that for now has to be done via the command line. We intend to allow doing this from the Stackspin dashboard instead in the future.

These instructions explain how to restore the persistent data of an individual app (such as Nextcloud, or Zulip) to a previous point in time, from a backup to S3-compatible storage made using velero, on a Stackspin cluster that is in a healthy state. Using backups to recover from more severe problems, like a broken or completely destroyed Stackspin cluster, is also possible, by reinstalling the cluster from scratch and restoring individual app data on top of that. However, that procedure is not so streamlined and not documented here. If you are in that situation, please reach out to us for advice or assistence.

10.2.1 Select backup

To show a list of available backups, perform the following command on your VPS:

```
$ kubectl get backup -A
```

Once you have chosen a backup to restore from, record its name as written in the `kubectl` output.

Note: Please be aware that for technical reasons the restore operation will restore not only the persistent data from this backup, but also the app's software version that was running at that time. Although the auto-update mechanism should in turn update the app to a recent version, and the recent app version should be able to automatically perform any necessary data format migrations on the old data, this operation has not been tested for older backups, so please proceed carefully. As an example of what could go wrong, Nextcloud requires upgrades to be done in a serial fashion, never skipping a major version upgrade, so if your backup is from two or more major Nextcloud versions ago, some manual intervention is required. If you have any doubts, please reach out to us.

10.2.2 Restore app data

Warning: Please note that restoring data is a destructive operation! It will replace the app's data as they are now. There is no way to undo a restore operation, unless you have a copy of the current app data, in the form of a current Stackspin backup or an app-specific data export. For that reason, we recommend making another backup right before beginning a restore operation.

To restore the data of app `$app` (for restoring the dashboard, see the note at the end of this subsection) from the backup named `$backup`, perform the following commands:

```
$ flux suspend kustomization $app
$ flux suspend helmrelease -n stackspin-apps $app
$ kubectl delete all -n stackspin-apps -l stackspin.net/backupSet=$app
$ kubectl delete secret -n stackspin-apps -l stackspin.net/backupSet=$app
$ kubectl delete configmap -n stackspin-apps -l stackspin.net/backupSet=$app
$ kubectl delete pvc -n stackspin-apps -l stackspin.net/backupSet=$app
$ velero restore create arbitrary-name-of-restore-operation --from-backup=$backup -l␣
↪stackspin.net/backupSet=$app
```

At this point, please first wait for the restore operation to finish, see text below.

```
$ flux resume helmrelease -n stackspin-apps $app
$ flux resume kustomization $app
```

Note: Specifically for Nextcloud, the `kubectl delete pvc ...` command might hang due to a Kubernetes job that references that PVC. To solve that, look for such jobs using `kubectl get job -n stackspin-apps` and delete any finished ones using `kubectl delete job ...`. That should let the `kubectl delete pvc ...` command finish; if it was already terminated, run it again.

The `velero restore create ...` command initiates the restore operation, but it doesn't wait until the operation is complete. You may use the commands suggested in the terminal output to check on the status of the operation. Additionally, once the restore operation is finished, it may take some more time for the various app components to be fully started and for the app to be operational again.

Note: To restore the "dashboard" data, which contains among other things the set of Stackspin users, follow the instructions above, using `dashboard` as `$app`, except that the kustomization to suspend and resume is the `single-sign-on` one, and the helmrelease to suspend and resume is the `single-sign-on-database` one in the `stackspin` namespace.

10.3 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/stackspin.example.org`.

10.4 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'
```

10.5 Run Nextcloud occ commands

Nextcloud includes a CLI tool called `occ` ("OwnCloud Console"). This tool can be used for all kinds of tasks you might want to do as a system administrator.

To use the tool, you need to enter Nextcloud's "pod" and change to the correct user. The following commands achieve that:

`exec` opens a root terminal inside the pod:

```
$ kubectl -n stackspin-apps exec deploy/nc-nextcloud -it -- bash
```

Change to the `www-data` user:

```
$ su -s /bin/bash www-data
```

Run `occ`:

```
$ php occ list
```

CHAPTER

ELEVEN

UPGRADES

Most of the time, a Stackspin cluster will pull and apply upgrades from our repository automatically. You can find details on automated upgrades in the *Automated upgrades* section.

From time to time though, changes are introduced that require some manual action. We mark this by increasing the first component of the Stackspin version number, and call this a "major release". Please follow *Manual upgrades* for more information.

11.1 Automated upgrades

11.1.1 Flux

Flux maintenance window

Automated upgrades based on Flux are configured to run during the night so they don't produce a possible downtime during working hours. Currently the maintenance window is configured for 2am until 4am (using the same timezone of your cluster host). We plan to make this time configurable via the Stackspin Dashboard.

During this time the Flux `Stackspin` `gitRepository` is resumed, and suspended outside this window.

If you want to apply automated upgrades outside this window you can manually resume the `Stackspin` `gitRepository`:

```
$ flux resume source git stackspin
```

And afterwards suspend it again:

```
$ flux suspend source git stackspin
```

Apply changes to flux variables

Before installing, you configured cluster variables in your cluster directory in *.flux.env*. If you change any of these variables after installation you can apply the changes by following the *Step 2: Install core applications* instructions until the step `kubectl apply -k $CLUSTER_DIR`. Then, use the following command that will apply the changes to all installed kustomizations:

```
$ flux get -A kustomizations --no-header | awk -F' ' '{system("flux reconcile -n " $1 "␣
↪kustomization " $2)}'
```

11.1.2 System-upgrade-controller

We use the Rancher system-upgrade-controller to auto-upgrade k3s on the cluster node. `system-upgrade-controller` is pretty versatile and could be used for other automated upgrades in the future, but right now we only use it for `k3s`.

Similar to automated application upgrades with Flux, automated upgrades for `k3s` are configured to run during the night so they don't produce a possible downtime during working hours. Currently the maintenance window is configured for 1am until 2am (using the same timezone of your cluster host). We plan to make this time configurable via the Stackspin Dashboard.

During this time the `system-upgrade-controller` deployment is scaled up so that the pod is running and performing possible upgrades, it there are any. Outside this time the `system-upgrade-controller` is scaled down.

If you want to apply automated `k3s` upgrades outside this window you can manually scale up `system-upgrade-controller deployment`.

```
$ kubectl -n system-upgrade create job --from=cronjob/system-upgrade-controller-scale-up␣
↪system-upgrade-controller-up
```

And afterwards scale it down again:

```
$ kubectl -n system-upgrade create job --from=cronjob/system-upgrade-controller-scale-
↪down system-upgrade-controller-down
```

Please be aware that is currently not supported to apply a custom `k3s` version because Stackspin would override this again, resulting in a possible ``k3s`` downgrade.

11.2 Manual upgrades

General instructions for upgrading to a new major version:

1. Check out the new branch on provisioning machine:

   ```
   git checkout v2
   ```

2. Load python virtualenv if necessary.
3. Run the upgrade script. Enter the name of your cluster's folder in `clusters/` for `<cluster_name>`.

   ```
   ./bin/upgrade-scripts/to-2.0/upgrade.sh <cluster_name>
   ```

4. Check that all components are being upgraded by flux:

   ```
   watch flux get kustomizations
   ```

   This may take quite a while, but all Kustomizations should become Ready after some time. If not, investigate and/or ask for support.

11.2.1 From 1.0 to 2

Please follow the general upgrade guide above.

- If the upgrade script fails (i.e. with `Helm upgrade failed: cannot patch "kube-prometheus-stack-prometheus-node-exporter" with kind DaemonSet`) please run the upgrade script again. If the issue persists, please reach out to us so we can help out.
- If you have Nextcloud and/or Zulip installed, after following the upgrade instructions, they will most likely end up in a failing state. If that happens, run `./bin/upgrade-scripts/to-2.0/fix-app.sh <cluster_name> <app>` for each of them. Especially Nextcloud installation can take a while, the script can take up to 20 minutes to complete.

11.2.2 From 0.8 to 1.0

1. Check out the new branch on provisioning machine:

```
git checkout v1.0
```

2. Load python virtualenv if necessary.
3. Update python packages:

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

4. Run the ansible playbook:

```
python -m stackspin stackspin.example.org install
```

5. Export the KUBECONFIG env variable, pointing to the *kube_config_cluster.yml* in your cluster directory, i.e.:

```
export KUBECONFIG=stackspin.example.org/kube_config_cluster.yml
```

6. Run the upgrade script:

```
./bin/upgrade-scripts/to-1.0/upgrade.sh
```

After running the last step (upgrade script), flux will update all components. In some cases an error can occur at that step.

- If the nextcloud upgrade fails because the *setup-apps* job consistently fails with *App "ONLYOFFICE" cannot be installed because it is now compatible with this version of the server.*, then manually run *php occ app:update –all* and then *php occ app:enable onlyoffice* inside the nextcloud container.
- If the *kube-system-config* Kustomization fails calling a validation webhook, simply retry it with *flux reconcile kustomization kube-system-config*.

11.2.3 Upgrading to 0.8

Note: 0.8 introduce many breaking changes. We gave our best to make the upgrade smooth but this will require a lot of manual intervention. Please reach out to us for help if needed !

When upgrading to version 0.8 OpenAppStack will be renamed to its final name: *Stackspin*. This comes with many changes, some of which need to be applied manually.

We have written a script to automate a lot of the preparations for the upgrade. However, afterwards you might need to get your hands dirty to get all your applications to work again. **Read this whole upgrade guide carefully, before you get started!**

Log in to your Stackspin server

```
$ ssh <server>
```

Download our upgrade script

```
$ wget https://open.greenhost.net/stackspin/stackspin/-/raw/main/bin/upgrade-scripts/to-
↪0.8.0/rename-to-stackspin.sh
$ chmod +x rename-to-stackspin.sh
```

First of all, if you have any `-override` configmaps or secrets, you'll want to move them from the `oas` namespace to the `stackspin` namespace, and from `oas-apps` to `stackspin-apps` (you also need to make these namespaces first). You also need to rename them from `oas-X` to `stackspin-X`. You can use a command like this to rename the cm and move it to the right namespace.

```
$ kubectl get cm -n oas-apps oas-$APP-override -o json | jq '.metadata.name="stackspin-
↪$APP-override"' | jq '.metadata.namespace="stackspin-apps"' | kubectl apply -f -
```

This script will cause serious down time and it will not do everything for you. Rather, it will prepare your cluster for the upgrade.

The script does the following:

1. Install `jq`
2. Shut down the cluster, make a back-up of the data, and bring the cluster back up
3. Copy all relevant `oas-*` secrets to `stackspin-*`
4. Move all PersistentVolumeClaims to the `stackspin` and `stackspin-apps` namespaces and sets the PersistentVolumes ReclaimPolicy to "Retain" so your data is not accidentally deleted.
5. Delete all OAS `flux` kustomizations
6. Delete the `oas` and `oas-apps` namespace
7. Create the new `stackspin` source and kustomization

Because there are not many Stackspin users yet, the script can need some manual adjustments. It was written for clusters on which all applications are installed. If you have *not* installed some of the applications, please remove these applications form the script manually.

Execute the upgrade preparation script:

```
$ ./rename-to-stackspin.sh
```

After this, you need to update secrets and Flux in the cluster by running `install/install-stackspin.sh`. Then re-install applications by running `install/install-app.sh <app>` from the Stackspin repository. See the application specific upgrade guides below.

After all your applications work again, you can clean up the old secrets and reset the Persistent Volume ReclaimPolicy to `Delete`

```
$ wget https://open.greenhost.net/stackspin/stackspin/-/raw/main/bin/upgrade-scripts/to-
↪0.8.0/cleanup.sh
$ chmod +x cleanup.sh
$ ./cleanup.sh
```

Nextcloud

Your SSO users will have new usernames, because the OIDC provider has been renamed from `oas` to `stackspin` and because the new SSO system uses UUIDs to uniquely identify users.

You can choose from these options:

1. Manually re-upload and re-share your files after logging in to your new user for the first time.
2. It is possible to transfer files from your previous user to the new user. To do so, find your new username. It is visible in Settings -> Sharing behind "Your Federated Cloud ID" after you've logged out and in to Nextcloud with the new SSO (the part *before* the `@`).
    1. Exec into the Nextcloud container

        ```
        $ kubectl exec -n stackspin-apps nc-nextcloud-xxx-xxx -it -- /bin/bash
        ```

    2. Change to the www-data user

        ```
        $ su www-data -s /bin/bash
        ```

    3. Repeat this command for each username

        ```
        $ php occ files:transfer-ownership oas-<old username> <new user ID>
        ```

        Note: the files are tranferred to a subfolder in the new user's directory

Depending on when you first installed Nextcloud, the `setup-apps` job may fail during the upgrade. If that happens, execute these commands in order to update the failing apps to their newest version, and to remove old files that can cause problems.

```
kubectl exec -n stackspin-apps deployment/nc-nextcloud -- rm -r /var/www/html/custom_
↪apps/onlyoffice
kubectl exec -n stackspin-apps deployment/nc-nextcloud -- rm -r /var/www/html/custom_
↪apps/sociallogin
flux suspend hr -n stackspin-apps nextcloud && flux resume hr -n stackspin-apps nextcloud
```

Rocket.Chat

We replaced Rocket.Chat with Zulip in this release. If you want to migrate your Rocket.Chat data to your new Zulip installation please refer to Import from Rocket.Chat.

Monitoring

The monitoring stack will work after the upgrade, but monitoring data from the previous version will not be available.

Wekan

In our testing we didn't need to change anything for Wekan to work.

WordPress

In our testing we didn't need to change anything for WordPress to work.

11.2.4 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 as the python-yq one, 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:

```
#!/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
```

(continues on next page)

(continued from previous page)

```
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
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 script:

```
#!/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 -
```

(continues on next page)

(continued from previous page)

```
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 -
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.

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.

11.2.5 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.

11.2.6 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.

11.2.7 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:
    # https://docs.openappstack.net/en/latest/installation_instructions.html#step-2-
↪optional-cluster-backups-using-velero
```

(continues on next page)

(continued from previous page)

```
    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'
```

11.2.8 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`

CHAPTER

TWELVE

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 Stackspin 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 Stackspin, most importantly: Kubernetes, Helm, Ansible and Flux 2.

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

12.1 Prerequisites

- A functional Stackspin cluster installed following the Stackspin installation instructions

12.2 Customize Stackspin applications

Apps deployed by Stackspin 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

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

12.3 Custom flux repository

Stackspin uses Flux 2 to install and auto-update applications. If *Customize Stackspin applications* is not enough and you need more advanced customization, i.e. to install extra applications or other things into the Kubernetes cluster, our advice would be to set up your own custom Flux git repository.

Please refer to the Stackspin Flux example and the Flux 2 documentation for more information.

12.3.1 Adding custom apps to the cluster

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 Stackspin applications, because they are part of the same cluster and VPS.

Please follow our Gitlab issue template for adding new apps. Not all steps are needed if you only want to add a new app to your cluster though (see the template for details). If you intend to contribute to Stackspin with a new app merge request please make sure to check all boxes.

12.4 Custom domain names for applications

It is possible to override the default domain names we have defined for the applications. In the future, it should be possible to do that through the dashboard, but currently you need to get your hands dirty inside the Kubernetes cluster.

To change an application's domain name, you need an already running and installed Stackspin cluster. After you have installed the Stackspin core, you can continue.

1. If you haven't already, create a DNS record for your desired domain that points to your cluster's IP address.
2. Edit the configmap called `stackspin-<app>-kustomization-variables` in the `flux-system` domain

   ```
   $ kubectl edit configmap -n flux-system stackspin-<app>-kustomization-variables
   ```

3. Find the `<app>_domain` value in the configmap, and change it to your desired domain

If you did not install the application yet, follow *Step 3: Install additional applications* to install the application to the custom domain.

If you *have* already installed the application, you need to "reconcile" the kustomization and the helmrelease that correspond to it. Execute the following commands in order:

```
$ flux reconcile kustomization <app>
```

```
$ flux reconcile helmrelease -n stackspin-apps <app>
```

Note that if you're changing the domain for Grafana, the last command needs to run in the `stackspin` namespace rather than `stackspin-apps`.

CHAPTER

THIRTEEN

EXTERNAL APPS

13.1 Adding non-native stackspin apps

It is possible to install non-native Stackspin apps to the dashboard and use Stackspin to log in to that application.

It is required that the application supports OAuth2/OpenID Connect mechanism to authenticate.

This document describes how to add apps. When examples are used, it assumes *gitlab* is added as external app.

13.1.1 Requirements

- Provisioning machine
- Access to *kubectl* to control the cluster
- Details of external app requirements

13.1.2 Create oAuth objects in the cluster

In an oauth authorization sequence, the app needs to be authorized by the OAuth2 provider, and uses application credentials to do so. These are the *client_id* and *client_secret*. These should be unique per added app. It is used to identify which app is trying to authorize. On top of that, an app has a *return url*, which is used to return to after authentication. This URL needs to be static and can not contain a wildcards.

Because it is possible the app you are trying to add will be added to Stackspin in the future, it is recommended to prefix the app name with *ext-*. So if you want to add *gitlab* as an external app, it is advised to use *ext-gitlab* as *client_id* and other references to the app name. This way it can never conflict with naming in Stackspin.

Stackspin requires to register two objects in the cluster to let applications authenticate. The kubernetes objects are:

- Secret: Containts the app credentials (*client_secret* and *client_id*)
- OAuth2Client: Containts the oAuth configuration for the app

Below are templates for those objects, which can be saved in a *yaml* file. The *client_secret* and *client_id* are free to define by yourself. They need to be the same in your external app, as on the stackspin cluster. We advise the *client_id* to be identical to *ext-{APPNAME}*

```
apiVersion: v1
metadata:
  name: stackspin-ext-{APPNAME}-oauth-variables
  namespace: flux-system
stringData:
```

(continues on next page)

(continued from previous page)

```
  client_id: {CLIENT_ID}
  client_secret: {CLIENT_SECRET}
kind: Secret
```

Template for the OAuth2Client object:

```
apiVersion: hydra.ory.sh/v1alpha1
kind: OAuth2Client
metadata:
  name: ext-{APPNAME}-oauth-client
  namespace: flux-system
spec:
  grantTypes:
  - authorization_code
  - refresh_token
  - client_credentials
  hydraAdmin: {}
  metadata: null
  redirectUris:
  - https://APP.HOSTNAME/path/to/callback
  responseTypes:
  - id_token
  - code
  scope: openid profile email stackspin_roles
  secretName: stackspin-ext-{APPNAME}-oauth-variables
  tokenEndpointAuthMethod: client_secret_basic
```

The latter template configures the specifics of the client. This is a general template, however, it is possible that changes are require to make the template work with the app you want to add.

Most important are:

- redirectUris: the URL where the APP redirects to after authentication. This is specified by the app.
- tokenEndPointAuthMethod: Two common endpoint methods are *client_secret_basic* and *client_secret_post*. Verify with your app which is used by your application.

13.1.3 Install objects in your cluster

Once you created the object, you need to install them in your cluster:

```
$ kubectl apply -f ext-appname-secret.yaml
$ kubectl apply -f ext-appname-oauth.yaml
```

13.1.4 Configure your application

Now it is time to configure your application. How to do this really depends on the application you are using. Read the documentation on how to configure oAuth. For this you will need the client_id and client_secret.

Once this is completed, you should already be able to login with a Stackspin admin user. Other users don't have access to the application by default. You can configure user access through the Stackspin dashboard after finishing this guide.

Required information to configure your application:

- SSO url: https://sso.YOUR-DOMAIN/
- Authorize URL: /oauth2/auth
- Token URL: /oauth2/token
- Userinfo: /userinfo

Note: Stackspin provides as *sub* a unique identifier. This is the internal ID of the user and is unique. Even if a use changes their username or email. How information Stackspin shares with the app is used to map (or create) local users differs per application. By default, *email* and *preferred_username* are shared with the application.

13.1.5 Configure the dashboard

Now your application is working, you want to add it to the dashboard, so you can actually assign roles for the app to other users.

This is done by adding the application from the CLI:

```
$ kubectl exec -n stackspin exec deploy/dashboard -c backend -- \
    flask cli app create ext-APPNAME "App Name" https://APP.DOMAIN
```

Note: The last argument is an URL. If the URL is specified, the app is handled as an external application. For internal/Stackspin managed applications this value should not be specified and is determined by Stackspin iteself.

Done! You now added the app and it should show up at the dashboard. It should be possible to assign roles from the web-interface.

CHAPTER

FOURTEEN

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 Stackspin machine. All other commands need to be run on your *provisioning 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.

14.1 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.

14.2 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

14.3 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 Stackspin CLI:

```
$ python -m stackspin stackspin.example.org info
```

Look for these lines in the output:

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

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

Alternatively, use SSH to log into your machine, and `kubectl` will be available there.

14.3.1 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 resource 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.

Be aware that there is a difference between Flux Kustomization objects and Kubernetes kustomizations. In this section we refer to the Flux kustomizations.

To find out if all *kustomizations* have been applied correctly, run the following flux command in your cluster or from the provisioning machine:

```
cluster$ flux get kustomizations --all-namespaces
```

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

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

If there is an issue, use `kubectl` to inspect the respective service, for example `nginx`:

```
$ kubectl describe helmrelease -n stackspin nginx
```

If the error message mentions a problem in a `HelmChart` or `GitRepository`, you can get information about those objects in a similar fashion:

For git repositories:

```
$ flux get source git
```

For helm repositories:

```
$ flux get source helm
```

For helm charts:

```
$ flux get source chart
```

For more information, use `flux --help`, or `flux get --help`.

HelmReleases that have no problems with their sources, but still fail, can often be fixed by simply suspending and resuming them. Use these `flux` commands:

```
$ flux --namespace NAMESPACE suspend helmrelease NAME
$ flux --namespace NAMESPACE resume helmrelease NAME
```

If your HelmRelease is outdated, you can often resolve complications by telling Flux to *reconcile* them. This will tell Flux to compare the HelmRelease's current state with the desired state.

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

Viewing upgrade history

To see when an application was updated to which version, you can use the `helm history` command. For example, to see the update history for Zulip, you can run:

```
$ helm history -n stackspin-apps zulip
```

14.3.2 Debugging on a lower level

You can also debug the `pods` that run applications. To get an overview of all pods, run:

```
$ kubectl get pods --all-namespaces
```

This will show you all pods. Check for failing pods by looking at the `READY` column. If you find failing pods, you can access their logs with:

```
$ kubectl --namespace NAMESPACE logs POD
```

You can also enter the pod's shell, by running:

```
$ kubectl --namespace NAMESPACE exec POD -it -- /bin/sh
```

14.4 HTTPS certificates

Stackspin 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 Zulip (https://zulip.stackspin.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.stackspin.example.org*. We will start by checking if `cert-manager` is set up correctly.

14.4.1 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.

14.4.2 Are all cert-manager pods in the *stackspin* 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
stackspin         hydra-public.tls                True    hydra-public.tls                ␣
↪14d
stackspin         single-sign-on-userpanel.tls    True    single-sign-on-userpanel.tls    ␣
↪14d
stackspin-apps    stackspin-nextcloud-files             True    stackspin-nextcloud-files␣
↪            14d
stackspin-apps    stackspin-nextcloud-office            True    stackspin-nextcloud-
↪office            14d
stackspin         grafana-tls                     True    grafana-tls                     ␣
↪13d
stackspin         alertmanager-tls                True    alertmanager-tls                ␣
↪13d
stackspin         prometheus-tls                  True    prometheus-tls                  ␣
↪13d
```

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

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

For even more information, inspect the logs of the *cert-manager* pod:

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

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

14.4.3 Example

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

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

We see that the zulip certificate resources have been in a bad state for 15 hours.

Show certificate resource status message:

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

We see that the *certificate* is waiting for the *certificaterequest*, let's query its status message:

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

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

```
$ kubectl -n stackspin-apps describe order stackspin-zulip-2045852889-1775447563
```

Show the failed challenge resource reason:

```
$ kubectl -n stackspin-apps get challenge stackspin-zulip-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 stackspin-apps delete challenges.acme.cert-manager.io stackspin-zulip-
↪2045852889-1775447563-837515681
```

14.4.4 Common installation failures

var substitution failed

When you execute `flux get kustomization` and you see this error:

```
$ flux get kustomization
var substitution failed for 'kube-prometheus-stack': YAMLToJSON: yaml: line 32: found␣
↪character that cannot start any token
```

That can mean that one of your values contains a double quote (") or that you quoted a value in *.flux.env* during the *Step 1: Flux configuration*. Make sure that *.flux.env* does not contain any values that are quoted.

If you need to change *.flux.env*, run the following commands:

```
$ kubectl apply -k $CLUSTER_DIR
```

Afterwards, you can speed up the process that fixes your *kustomization*, by running the following (replace *kube-prometheus-stack* with the *kustomization* mentioned in the error message):

```
$ flux reconcile kustomization kube-prometheus-stack
```

14.5 Purge Stackspin and install from scratch

Warning:

You will lose all your data!

This completely destroys Stackspin and takes everything offline. If you choose to do this, you will need to re-install Stackspin and make sure that your data is stored somewhere other than the VPS that runs Stackspin.

If things ever fail beyond possible recovery, here is how to completely purge a Stackspin installation to start from scratch:

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

CHAPTER

FIFTEEN

TESTING

To get an overall status of your cluster, run the tests from the command line. There are two types of tests: testinfra tests verify the setup, and end-to-end tests that check the webservice availability.

15.1 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 Stackspin cluster from the outside. For example, the certificate check will check if the Stackspin returns valid certificates for the provided services. The clearbox tests run on the Stackspin 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.

If you have installed the Stackspin python requirements inside a *virtual environment* then you need to activate it before testing:

```
$ . env/bin/activate
```

Then 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 *stackspin.example.org* with your cluster name): Then export the *KUBECONFIG* variable that lets `kubectl` talk to your cluster. Thirdly, export a variable that contains the password for the Prometheus HTTP endpoint.

```
$ export CLUSTER_DIR="${PWD}/../clusters/stackspin.example.org"
$ export KUBECONFIG=${CLUSTER_DIR}/kube_config_cluster.yml
$ export BASIC_AUTH_PW=$(kubectl get secret -n stackspin stackspin-prometheus-basic-auth␣
↪--template '{{ .data.pass }}' | base64 -d)


$
```

15.1.1 Run all tests

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

15.1.2 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://*'
```

15.1.3 Tests a specific application

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

15.1.4 Advanced usage of Testinfra tests

Specify host manually:

```
$ py.test -s --hosts='ssh://root@example.stackspin.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.stackspin.net' py.test -sv -m 'certs'
```

or directly:

```
$ FQDN='example.stackspin.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/stackspin/stackspin'
$ export SSH_PRIVATE_KEY="$(cat ~/.ssh/id_ed25519_stackspin_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
```

15.1.5 Known Issues

The default ssh backend for testinfra tests is *paramiko*, which does not work out of the box. It fails to connect to the host because the *ed25519* hostkey is not verified. Therefore we need to force plain *ssh://* with either `connection=ssh` or `--hosts=ssh://...`

15.2 End to end tests

Warning: We are migrating our end to end tests from Taiko to Cypress. You can track the progress in this issue. You need to check below which applications can be tested with which framework. We are sorry for the current inconvenience, we hope we'll soon finish the migration.

End to end tests run in a browser to test if all the web services are up and running and functioning correctly.

15.2.1 Taiko tests

Used for testing these applications:

- Dashboard
- Wekan
- Wordpress
- Zulip

Install Taiko on your provisioning machine:

```
$ npm install -g taiko
```

Run Taiko tests

Taiko tests are integrated in the `stackspin` CLI command suite. To run all Taiko tests, run the following command in this repository:

```
$ python -m stackspin CLUSTERNAME test
```

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

```
$ python -m stackspin CLUSTERNAME test --help
```

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

```
$ python -m stackspin CLUSTERNAME test --taiko-tags nextcloud
```

Advanced Taiko tests

If you want to use Taiko without invoking the stackspin CLI, go to the *test/taiko* directory and run:

For nextcloud & onlyoffice tests:

```
$ export DOMAIN='stackspin.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.

15.2.2 Cypress tests

Cypress tests are *not yet* integrated in the `stackspin` CLI command suite. Used for testing these applications:

- Grafana
- Nextcloud

To install Cypress on your provisioning machine go to the *test/* directory and run:

```
$  npm install -g cypress
```

Test Grafana

```
$ export CYPRESS_BASE_URL='https://grafana.stackspin.example.net'
$ export CYPRESS_USE_SSO_LOGIN="true"
$ export CYPRESS_SSO_USER='admin'
$ export CYPRESS_SSO_PW='...'
$ cypress run --headed
```

Test nextcloud

The cypress end to end test is not included in the main Stackspin repository, but in the Nextcloud helm chart repository.

```
$ git clone https://open.greenhost.net/stackspin/nextcloud.git
$ export CYPRESS_BASE_URL='https://files.stackspin.example.net'
$ export CYPRESS_USE_SSO_LOGIN="true"
$ export CYPRESS_SSO_USER='admin'
$ export CYPRESS_SSO_PW='...'
$ cypress run --headed
```

CHAPTER

SIXTEEN

SECURITY

16.1 Access control

By default, the applications on your Stackspin 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 Stackspin ingress (ingress-nginx) to only accept connections from a certain IP address or range.

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

```
---
apiVersion: v1
kind: secret
metadata:
  name: stackspin-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

SEVENTEEN

CONTRIBUTING

17.1 Code of conduct

We follow the Contributor Covenant code of conduct, please see the CODE_OF_CONDUCT.md document for details.

17.2 Preparing the development environment

Make sure you have development dependencies installed in your development environment.

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

17.2.1 pre-commit hooks

We use pre-commit to test our changes before commiting. We configured these tests:

- Basic linting and style checks shipped with pre-commit
- Conventional commit messages (optional)
- Check for private keys or secrets using gitleaks
- … and more, see the `.pre-commit-config.yaml` file in the root of the Stackspin repo

Please install these required tools on your system:

- shellcheck and
- shfmt to lint and validate shell scripts

Then install the pre-commit hooks:

```
$ pre-commit install
```

Optional:

If you want to lint your commit msgs according to Conventional Commits uncomment the conventional commits section in `.pre-commit-config.yaml` and install the commit-msg hook with:

```
$ pre-commit install --hook-type commit-msg
```

Running `git commit` for the first time after installing the hook usually takes a little longer because `pre-commit` pulls its hooks from upstream repositories. You can find all hooks in `.pre-commit-config.yaml`.

In case you need to skip the execution of the pre-commit hooks (please don't!), use `git commit --no-verify`.

17.3 Adding dependencies

Make sure you update our `requirements.txt` file before you push your changes. Whenever you need a new python package, add it to requirements.in and run

```
$ pip-compile
```

to generate an new `requirements.txt` which does not only pin the new package but also its dependencies.

If the new package you are adding is only used by developers, please add it to the `requirements-dev.txt` file.

17.4 CI pipeline image

We use a custom disk image for the VPSs used by the CI pipeline. On this image, the `install-kubernetes.yaml` playbook has already been applied, which usually saves a few minutes of pipeline running time.

17.4.1 What to do when I change a part of the `install-kubernetes.yaml` playbook?

Don't worry, the playbook *runs* in the CI (just faster, because usually nothing needs to change). So if you make changes, you can test those in the CI without problems.

If you want to start with a clean slate, however, you might want to change `.gitlab/ci_scripts/create_vps.sh` and temporarily remove the `--disk-image-id` argument.

Before you merge, make sure your changes are applied to a new custom image

If you changed the `install-kubernetes.yaml` playbook, for example to upgrade the k3s version in use, you'll want to generate a new disk image template and use it. This is a manual process for now. Follow these steps:

1. Create a new VPS

```
$ export HOST_NAME=baseimage$(date +'%Y%m%d')
# Make sure you use your private ssh key id, 411 is the ssh key used in CI
$ export SSH_ID=411
$ python -m stackspin ${HOST_NAME} create --create-droplet --create-hostname ${HOST_
↪NAME}.stackspin.net --ssh-key-id $SSH_ID --create-domain-records --subdomain $
↪{HOST_NAME} stackspin.net
```

2. Accept ssh host key

```
$ ssh root@${HOST_NAME}.stackspin.net
```

3. Run the following to install *only kubernetes* on the VPS:

```
$ python3 -m stackspin ${HOST_NAME} install
```

4. Log into your machine and clean up the k3s server, then delete the cluster data:

```
$ ssh root@${HOST_NAME}.stackspin.net
# Clean up running containers and firewall
$ /usr/local/bin/k3s-killall.sh
# Remove k3s state
$ rm -rf /var/lib/rancher/k3s
```

5. Log into Cosmos with the Stackspin account
6. Go to VPS Cloud -> VPS and shut down your VPS
7. Go to VPS Cloud -> Disk Images and click `Manage` for your VPSs disk image
    1. Change the Disk Label to something like `k3s-template-DATE`
    2. Set VPS to `-- not assigned --`
    3. Click save
    4. `Manage` the same image again
    5. Click 'make template'
    6. Choose "Linux Latest LTS" as the kernel
    7. Choose OS type Debian 11 (bullseye)
    8. Remember the disk image ID that you can see in the current URL as `id=...`
    9. Click save
8. Change the `--disk-image-id` argument in `.gitlab/ci_scripts/create_vps.sh` to your current disk image id **with a minus in front of it**. This is because custom images are negative integers, whereas Greenhost's disk images are positive integers. Don't forget to change the comment above to include the current commit sha and date.
9. Remove the droplet

You are now ready to merge the changes you made to the `install-kubernetes` playbook

CHAPTER

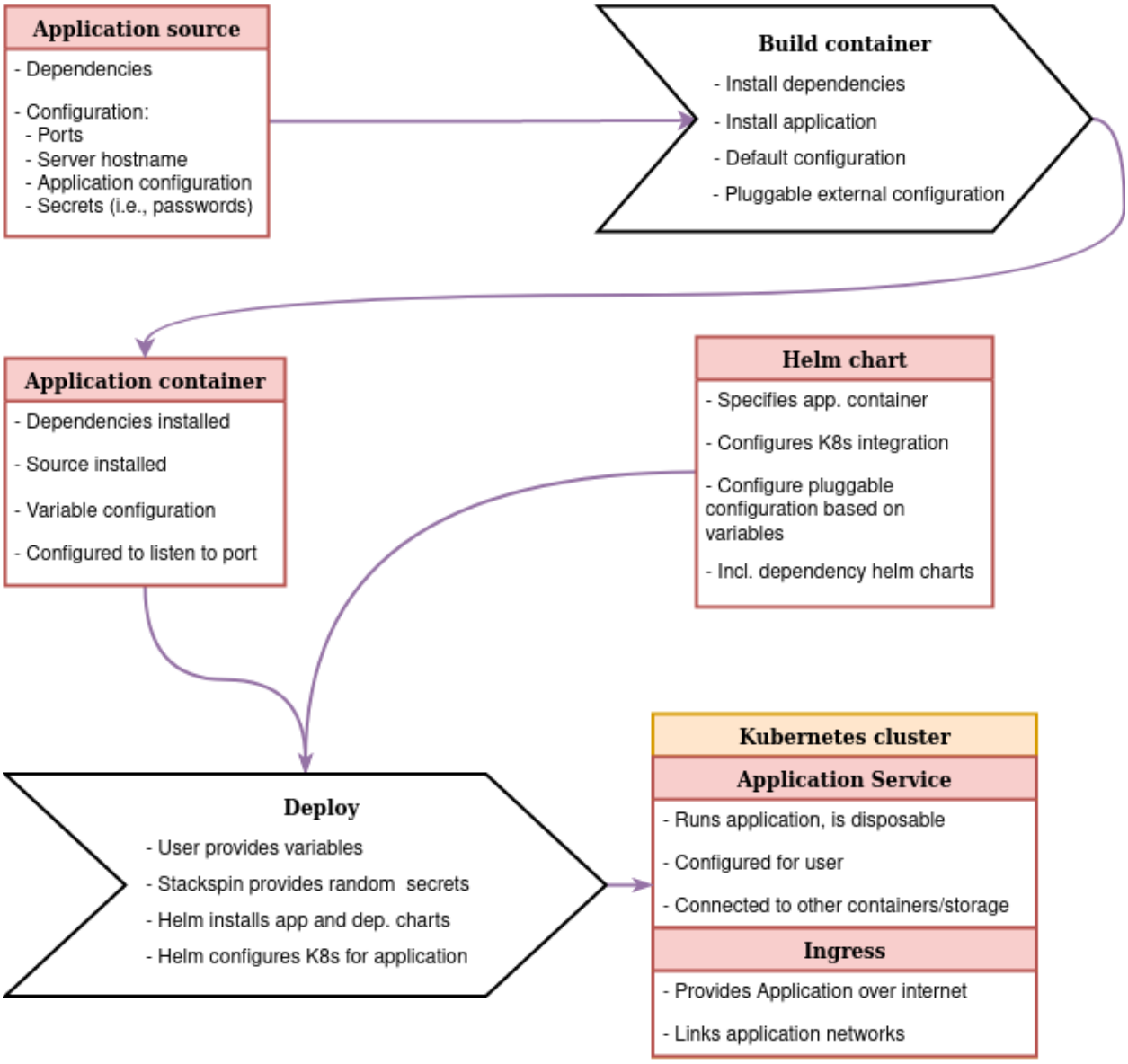
EIGHTEEN

STACKSPIN DESIGN

This article covers the basic design of Stackspin.

18.1 Application build pipeline

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



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.

18.2 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.

Stackspin 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.

18.3 Application containers

For inclusion in Stackspin, 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.

18.3.1 Container updates

When an application update is available, these updates need to be rolled out to Stackspin 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 Stackspin 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. Stackspin 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 Stackspin clusters will be developed by us.

18.4 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 Stackspin's simple setup, we use a local storage provisioner that automatically provides persistent data on the VPS running Stackspin to an application that requests it.

18.5 Automatic updates

Stackspin 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 Stackspin code repository. Once changes are pushed to the branch that Flux tracks, the changes are applied to the cluster.

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

18.6 APIs

Stackspin includes several APIs that are available for some pieces of the software to talk to. Specifically the Dashboard application uses most of these APIs.

- Ory Kratos is the user management API that we use. It comes with several handy features, like sending password reset emails and (soon) MFA.
- Ory Hydra provides us with an OpenID Connect workflow.
- Velero is a backup mechanism that integrates well with Kubernetes: it includes "Custom Resource Definitions" for backups and backup locations. As such, we can access it through the Kubernetes API.
- Flux 2 is used to configure and auto-update applications. App configuration happens mostly through Kubernetes `ConfigMaps` and `Secrets` and are applied through the Flux CRDs `Kustomization` and `HelmRelease`.

CHAPTER

NINETEEN

REFERENCE DOCUMENTATION

19.1 File structure

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

- `/var/lib/Stackspin/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

TWENTY

COMPARABLE PROJECTS

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

20.1 What makes Stackspin unique

However there are changes in implementation that make Stackspin 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. Also, all mentioned platforms require application modifications, for example to make use of the authentication system.

Stackspin avoids changing the included applications to prevent maintenance overhead and enable seamless native updates. It only requires OpenID Connect support, so the application can be integrated with single sign-on.

So far, Stackspin requires one intervention before an update can be pushed to users: It needs to be tested. We want to make sure that an application works well with the platform and other applications before we let you use it. We work towards a fully automated test suite to minimize failure while providing automatic updates.

20.2 Platforms

20.2.1 Fully Open

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 runs into limits.

YunoHost is mostly based on Debian and its package management and user management system. As a result, Yunohost is relatively lightweight. However, if an application on your Yunohost contains a security hole, the data of other applications is likely compromised as well. This is less likely in Stackspin as applications run in sealed-off containers.

SelfPrivacy automatically sets up an open stack focused on personal use from your phone on an established cloud provider. From the documentation it seems to not add any integration (such as SSO) between the services and the only dashboard is the app itself.

Mail-in-a-Box is a narrowly focused project that tightly integrates Nextcloud and various applications to create a full-featured mailserver with a control panel. As such there is no SSO and no configuration options.

20.2.2 Open Core

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

Univention Corporate Server is focused on business and does not make reproducible builds easily available or facilitate community contribution.

CHAPTER

TWENTYONE

SINGLE SIGN-ON ARCHITECTURE

The single sign-on system consists of a few components:

1. The *OAuth 2* and *OpenID Connect (OIDC) provider*, Hydra
2. The *Identity provider*, Kratos
3. The Login application which serves as a login panel, consent application and a settings screen for the Kratos settings. The login application code is part of the Dashboard Backend repository

21.1 Overview

The single sign-on system is a combination of applications that serve as a central user database and authenticates them to other applications. Users are stored inside Kratos's database. Kratos also serves an API that helps us generate interfaces for many user-related tasks, such as:

1. Setting your name and username
2. The whole password reset flow
3. The login form
4. 2FA (not implemented in the login application yet)

The Login application is mostly a front-end that uses the Kratos API to generate interfaces for logging in, resetting the password and setting some user data.

21.2 Flows

21.2.1 Logging in

The Kratos login flow is documented in the Kratos documentation. Our implementation is slightly different from what you see there:

21.2.2 User creation

We have not implemented Kratos's *Registration* flow, because users cannot self-register with a Stackspin cluster. An administrator can make new users using the Dashboard application. When a user is created, an email address always needs to be provided.

Once a user has been created, they can start the Account Recovery and Password Reset flow in order to set or reset their password. We use the "Recovery `link` Method" described in the Kratos documentation.

21.2.3 User settings

Although users can change their settings through the Dashboard application, the login application also implements the user-settings Kratos flow. Users that receive a password recovery link use this flow to reset their passwords. It also allows them to change their username and full name.

21.2.4 Authentication

The following is an adaptation of the sequence diagram provided in the Hydra documentation

21.3 Configuring OIDC clients

If you have installed the SSO system using the Helm chart, following this documentation, these are the settings that you usually need to for setting up new OIDC clients:

- OAuth 2 server URL: `https://sso.stackspin.example.org`
- OAuth 2 Auth Endpoint: `https://sso.stackspin.example.org/oauth2/auth`
- OAuth 2 Userinfo endpoint: `https://sso.stackspin.example.org/userinfo`
- OAuth 2 Token endpoint: `https://sso.stackspin.example.org/oauth2/token`

In addition you'll need to add the client to Hydra. This happens with Hydra Maester, a helper application that reads `oauth2clients.hydra.ory.sh` Kubernetes objects and synchronises them with Hydra.

An example `oauth2client`:

```
apiVersion: hydra.ory.sh/v1alpha1
kind: OAuth2Client
metadata:
  name: dashboard-oauth-client
spec:
  grantTypes:
    - authorization_code
    - refresh_token
    - client_credentials
    - implicit
  hydraAdmin: {}
  metadata: null
  redirectUris:
  - https://dashboard.stackspin.example.org/login-callback
  responseTypes:
  - id_token
```

(continues on next page)

(continued from previous page)

```
  - code
  scope: openid profile email stackspin_roles
  secretName: stackspin-dashboard-oauth-variables
  tokenEndpointAuthMethod: client_secret_post
```