CoReport

Jul 08, 2020

Contents

CoReport is an open-source tool that enables an organisation to collect structured status reports from other organisations at set intervals.

CHAPTER 1

The basic concept

An **organisation** that must make plans and allocate resources to deal with COVID-19, such a health board or city authority, needs regular updates of **information**. This can be quantitative information concerning things like available hospital beds and other equipment, staff, confirmed cases and so on, or qualitative information, such as notice of key decisions, situation assessments, etc.

This information needs to come from a wide variety of disparate **sources**, from hospitals and police services to community organisations and volunteers. At the same time, it needs to be consistently structured so that it can be processed quickly and accurately.

CoReport does this by allowing the organisation that deploys it to define the **questions** that need to be answered (including the appropriate frequency and deadlines for questions) and the different sources information needs to come from.

For example (a real set-up would include many more than this):

- *hospitals* may need to report daily on infection and testing numbers, the number of beds and ventilators available or supplies urgently required
- *police, military and fire services* may need to report thrice a week on the availability of personnel
- *community organisations* may need to report daily on matters of local concern

Reporting users of the application are prompted for the information, and reminded of expired deadlines.

The data thus captured can be exported in a consistent way in a variety of formats for further processing as required.

CHAPTER 2

About development

CoReport is currently being deployed by the Swiss canton of Baselland, on hosting sponsored by Divio. Divio is also donating further hosting of the tool for other organizations.

The development of CoReport was initiated by what. in collaboration with Divio (see our *Contributors*). The first iteration of the tool was developed for the Swiss canton of Baselland.

Project source code

CHAPTER 3

Contact us

For assistance with deployment, please contact Divio.

For all other queries, please contact contact@coreport.ch.

3.1 Get started

3.1.1 Set up a local instance for development

Note: CoReport is a project in development. The project and this documentation are updated regularly, and these notes may have changed since the last time you read them.

Clone the codebase:

```
git clone git@gitlab.com:what-digital/covid19-report.git
```

The project requires that you have Docker installed; you'll need:

- Macintosh users: Docker for Mac
- Windows users: Docker for Windows
- Linux users: Docker CE and Docker Compose

`cd` into the `covid19-report` directory and build the application:

```
docker-compose build
```

This will create images based on the services described in the `docker-compose.yml` file, including the `web` service, The Celery services, the Postgres database and any others.

Migrate the database:

```
docker-compose run web python manage.py migrate
```

This runs the Django `manage.py migrate` inside a `web` container.

Launch the application:

```
docker-compose up
```

It will take a few moments to start up all the containers, and then you will be able to reach the application at http://localhost:8000 (this is configured in `docker-compose.yml`). You should see an **unstyled** login page.

In order to generate the styling for the site, along with all its other frontend components, they will need to be built manually. This requires Yarn, a Node package, to be installed. (It is beyond the scope of this documentation to describe that process.)

In a new terminal session in the project directory, run:

```
yarn install --pure-lockfile
```

This installs the components that the project's frontend requires. Then:

```
yarn start
```

This launches the frontend; the frontend assets will be compiled and the served on port 8090. Yarn watches the frontend source and will re-compile the assets on any change.

When you refresh the login page, the styling should now appear.

3.1.2 Prepare the project for cloud deployment

The CoReport project is ready for deployment on the Divio cloud management platform, and the project already contains the components required for this. All your repository requires in order to test in a cloud deployment is to associate it with a Divio cloud project.

Using the Divio Control Panel (you will need an account, it's free), create a new project.

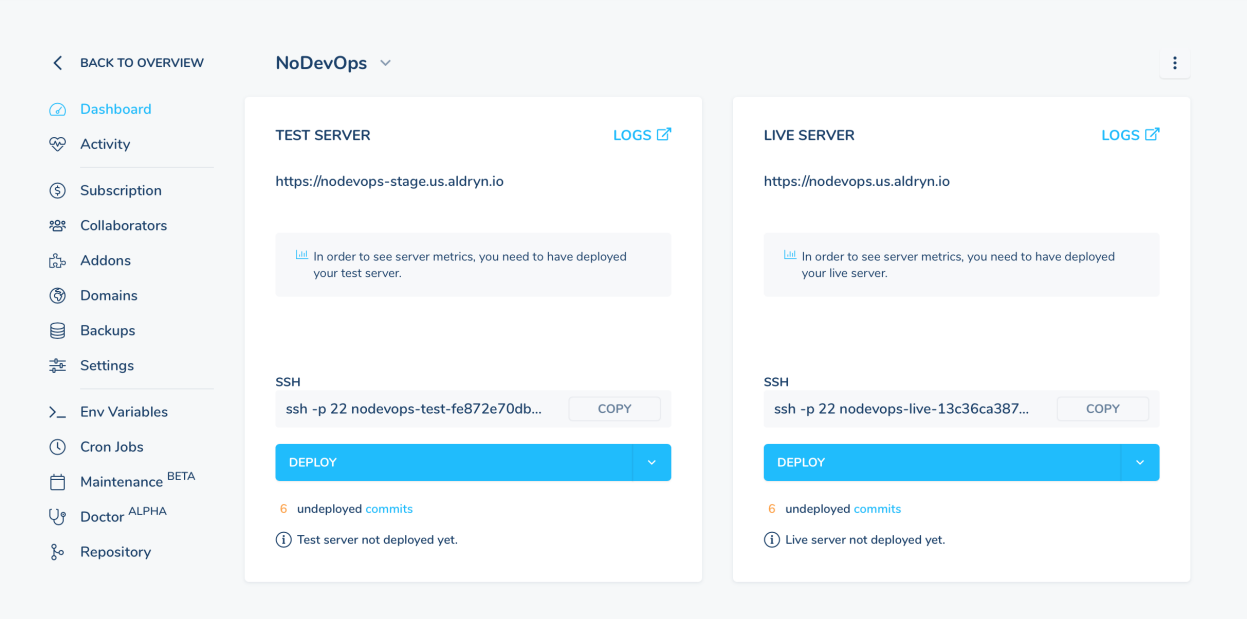
Select the defaults:

- *Platform*: `Python`
- *Type*: `Django`

You can use Divio's Git server (also the default).

Hit **Skip** in the *Subscription* view.

After a few moments, you'll be presented with its dashboard, which will look something like this:



Install the Divio CLI: `pip install divio-cli`.

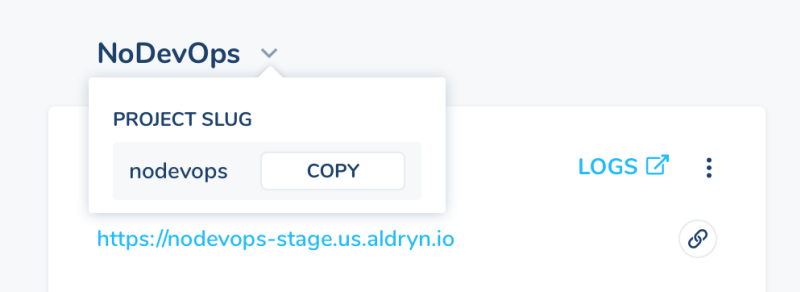
Run:

```
divio login
```

This will fetch a token from https://control.divio.com/account/desktop-app/access-token/.

Upload your public key to https://control.divio.com/account/ssh-keys/

Get your project's *slug* from the Control Panel:



Or you can see it by listing your projects:

```
divio project list
```

Set up the project:

```
divio project setup <project slug>
```

Once the `divio project setup` command has finished pulling down the repository, you will find a file in the newly-created directory:: `.aldryn`. The file contains the slug and id of the cloud project, and is what the Divio CLI uses to associate the local project with its counterpart on the cloud.

Move this file into the CoReport `covid19-report` project.

We need to do a similar thing with Git. In the local Divio project, find its Git URL:

```
git remote show origin
```

In the `covid19-report` project, add a new remote called `divio`, using this URL, for example:

```
git remote add divio git@git.divio.com:my-divio-coreport-project.git
```

Your `covid19-report` project is now associated with the Divio cloud project you just created. You can dispose of the local project that was created by the `divio project setup` command; it's no longer needed.

You can now interact with both the `origin` remote (for example, to pull new updates) and push your changes to the `divio` remote (to test it on the cloud).

3.1.3 Working with the project

Basic workflow:

```
git add <files>  # to stage changes for commit
git commit  # to commit changes
git push divio  # push changes to your cloud project
divio project deploy  # deploy the cloud Test server
```

Also useful:

```
divio project pull [or push] db  # copy the database to/from the cloud project
```

You can access the Test and Live sites of your cloud project via the Control Panel. See the Divio developer handbook for a quick guide to using the Divio CLI.

Whenever you pull or make changes that will require rebuilding the local project (for example, changing `requirements.in`) re-run:

```
docker-compose build
```

3.2 How-to guides

3.2.1 Work in and with the Dockerised environment

Almost any command can be executed inside the Dockerised environment. One way is to precede the command with:

```
docker-compose run web
```

followed by the command. For example, to run Django migrations

```
docker-compose run web python manage.py migrate
```

If you need to run several commands it may be more convenient to open a bash [or fish] shell:

```
docker-compose run web bash [or fish]
```

or a Django shell:

```
docker-compose run web python manage.py shell
```

3.2.2 Update `requirements.txt`

The project uses the Python dependencies listed in `requirements.txt`. This is a fully-pinned list of all dependencies. The list is derived from the project's `requirements.in`.

The build process uses `requirements.txt` if it's present, *not* `requirements.in`. To update `requirements.txt`:

```
docker-compose run pip-reqs compile
```

To rebuild the containers with the new requirements:

```
docker-compose build
```

(The actual build process will also execute `pip-reqs resolve` (which creates a `requirements.urls` file) followed by `pip install --no-index --no-deps --requirement requirements.urls` (which uses the `requirements.urls` file).

See How to pin all of your project's Python dependencies in the Divio developer handbook for more information about how this works.

3.2.3 Interact with the local and cloud database

See How to interact with your project's database in the Divio developer handbook.

3.2.4 How to build the documentation

The documentation is written in Restructured Text, built with Sphinx and hosted at Read the Docs.

The documentation can be built locally:

```
cd docs  # navigate to the documentation directory

# use the supplied Makefile to set up a virtual environment for building the␣
↪documentation
# - this only needs to be done once
make install

make run  # builds and serve the documentation
```

The documentation is served at http://localhost:9090/.

3.3 Reference

3.3.1 Key components

- Python 3.6
- Django 2.2
- Webpack 4
- TypeScript 3

3.3.2 Backend (Django) code

The root `backend` module is reserved for non-project related applications, eg `auth` or `articles`.

Project-specific applications should live in a project module of their own, for example `backend.dectris`.

Similarly, the `backend/templates` directory is only for global templates. For anything, else use app-specific templates (i.e. do things the standard Django way).

3.3.3 Frontend code

For any questions, please refer to Victor Yunenko (victor.yunenko@what.digital)

Project-specific scss styles can be in `global`; however, JS logic must not be.

When you need to add a script for a new page/block add a new `entry` in `webpack.config.js` and a page/block submodule.

For global scripts and styles use the `global` entry, and add the respective HTML code in your template.

If you need to add a global variable to JS, add it to the `backend/templates/default.html#DJANGO` const and extend `frontend/global/ts/django.ts#DJANGO`.

If you need to add a static file, e.g. on URL `https://localhost/static/img/icon.png`, add it to `frontend/` - everything in there will be accessible under the `/static/` URL path.

If you need to add a new font to CSS, use the global path as you normally would, e.g. `url('~pages/homepage/fonts/frutiger.woff')`.

If you need to add images or other assets, add them under the respective module, e.g. `global`, `vendor`, `pages/homepage`.

We use 24 columns in our Bootstrap 4 configuration.

3.3.4 Contributors

- Victor Yunenko victor.yunenko@what.digital
- Jonathan Stoppani jonathan.stoppani@divio.com
- Anand Patel anand@what.digital
- Elliott White elliott@what.digital
- Mario Colombo mario@what.digital
- Marcus Kuhn marcus@what.digital
- Daniele Procida daniele.procida@divio.com

3.4 Explanation

3.4.1 Key functionality

Administrators can:

- can create organisations and invite users to them
- manage groups that to which different organisations are assigned

- manage questions, that are assigned to groups; groups determine which questions apply to each organisation
- export all answers provided by users for purposes of data analysis

Users can:

- see the questions for their organisations
- enter and edit (until a deadline) their answers

The system will notify users when answers are overdue.

3.4.2 In development

The system will provide real-time reporting of the data, in a dashboard accessible only to authorised users.