cloudlaunch Documentation

Release 4.0.0

CloudVE

Oct 15, 2021

Contents

CloudLaunch is a ReSTful, extensible Django app for discovering and launching applications on cloud, container, or local infrastructure. A live version is available at https://launch.usegalaxy.org/.

CloudLaunch can be extended with your own plug-ins which can provide custom launch logic for arbitrary custom applications. Visit the live site to see currently available applications in the Catalog. CloudLaunch is also tightly integrated with CloudBridge, which makes CloudLaunch natively multi-cloud.

CloudLaunch has a web and commandline front-end. The Web UI is maintained in the CloudLaunch-UI repository. The commandline client is maintained in the cloudlaunch-cli repository.

CHAPTER 1

Installation

The recommended method for installing CloudLaunch is via the CloudLaunch Helm chart.

To install a development version, take a look at development installation page.

CHAPTER 2

Application Configuration

Once the application components are installed and running (regardless of the method utilized), it is necessary to load appliance and cloud provider connection properties. See this page for how to do this.

CHAPTER 3

Authentication Configuration

User authentication to CloudLaunch should be managed via social auth. For development purposes, it is possible to use Django authentication in which case simply creating a superuser is sufficient. If you intend on having users of your CloudLaunch installation, you will want to configure social auth.

CHAPTER 4

Table of contents

4.1 Overview

This section provides a quick overview of the various CloudLaunch pages.

4.1.1 Catalog

Use the catalog to search through the available appliances. A virtual appliance is a virtual machine that packages a ready-to-run application(s), eliminating the need to install and configure complex stacks of software. (e.g. Galaxy, Genomics Virtual Lab, SLURM).

4.1.2 Public Appliances

Public Appliances are appliances which have been made publicly available by an organization or individual. Although these applications are publicly available, they may require registration or impose usage quotas. You can contact us if you would like to list an applainces as public.

4.1.3 My Appliances

My Appliances lists all currently actie appliances. You can use this page to monitor the state of an appliances or to delete an appliance. You can also archive an appliance, in which case it will be moved to the Launch History page.

4.1.4 User Profile

Use this section to manage your user profile. You can use this page to save or edit credentials for various clouds.

4.2 Installing a production server

4.2.1 Upgrading running chart

1. Fetch latest chart version through `helm repo update`
2. Docker pull the latest image

```
sudo docker pull cloudve/cloudlaunch-server:latest
sudo docker pull cloudve/cloudlaunch-ui:latest
```

3. Upgrade then helm chart

```
helm upgrade --reuse-values <chart_name> galaxyproject/cloudlaunch
```

4.2.2 Reinstalling chart from scratch

0. Note down the existing secrets for fernet keys, secret keys, db password etc. through kubernetes in the cloudlaunch namespace. Dashboard access link: https://149.165.157.211:4430/k8s/clusters/c-nmrvs/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login

To obtain login token: https://gist.github.com/superseb/3a9c0d2e4a60afa3689badb1297e2a44

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep␣
↪admin-user | awk '{print $1}')
```

1. `helm delete <existing_chart>`
2. `kubectl delete namespace cloudlaunch`
3. Optionally, delete all cached docker images using

```
docker images
docker rmi
```

4. Delete existing persistent volume in rancher. This does not delete the local folder, so the database will survive. Recreate with following settings:

```
Name: cloudlaunch-database
Capacity: 30
Volume Plugin: Local Node Path
Path on the node: /opt/cloudlaunch/database
Path on node: A directory, or create
Customize -> Many nodes read write
```

5. `helm install galaxyproject/cloudlaunch --set cloudlaunch-server.postgresql.postgresqlPassword=<pg_password> --namespace cloudlaunch --set cloudlaunch-server.fernet_keys[0]='<replace with fernet key 1>' --set cloudlaunch-server.fernet_keys[1]='<replace with fernet key 2>' --set cloudlaunch-server.secret_key=<replace with secret key>`

4.3 Installation for development

CloudLaunch is made up of three services: the server, the user interface (UI), and a message queue. All three processes need to run for the application to function properly. See instructions below on how to install and start each of the processes.

4.3.1 Install the server

CloudLaunch is based on Python 3.6 and although it may work on older Python versions, 3.6 is the only supported version. Use of Conda or virtualenv is also highly advised.

1. Checkout CloudLaunch and create an isolated environment

```
$ conda create --name cl --yes python=3.6
$ conda activate cl
$ git clone https://github.com/galaxyproject/cloudlaunch.git
$ cd cloudlaunch
$ pip install -r requirements_dev.txt
$ cd django-cloudlaunch
```

2. Create a local copy of the settings file and make any desired configuration changes. No changes are required for CloudLaunch to run but it is advisable to at least update the value of the fernet key.

3. Run the migrations and create a superuser

```
$ python manage.py migrate
$ python manage.py createsuperuser
```

4. Start the web server and Celery in separate tabs. If you do not have Redis installed, you can install it via Conda: `conda install -c anaconda redis`

```
$ python manage.py runserver
$ redis-server & celery -A cloudlaunchserver worker -l info --beat
```

5. Visit http://127.0.0.1:8000/cloudlaunch/admin/ to define appliances and add cloud providers.

6. Visit http://127.0.0.1:8000/cloudlaunch/api/v1/ to explore the API.

4.3.2 Install the UI

1. Clone the source code repository

```
$ git clone https://github.com/galaxyproject/cloudlaunch-ui.git
$ cd cloudlaunch-ui
```

2. Install required libraries

Make sure you have `node` (version 6.*) installed (eg, via Conda, `conda install -c conda-forge nodejs`). Then install the dependencies.

```
# Install typescript development support
npm install -g tsd
# Install angular-cli
npm install -g @angular/cli
# Install dependencies
npm install
```

3. Run the development server

Start the development server with

```
npm start
```

Or if you use yarn as your preferred package manager, `yarn start`.

Access the server at `http://localhost:4200/`. The app will automatically reload if you change any of the source files.

If you are installing this on a VM instead your local machine and need to access the UI over the network, instead of using `npm start`, use `ng serve --host 0.0.0.0 --disable-host-check --proxy-config proxy.conf.json` The UI should be availale on the host IP address, port 4200.

4.4 Configure CloudLaunch with data

Once running, it is necessary to load the CloudLaunch database with information about the appliances available for launching as well as cloud providers where those appliances can be launched.

The following commands show how to load the information that is available on the hosted CloudLaunch server available at https://launch.usegalaxy.org/. It is recommended to load those values and then edit them to fit your needs.

4.4.1 Loading clouds

Appliances define properties required to properly launch an application on a cloud provider. Run the following commands from the CloudLaunch server repository with the suitable Conda environment activated.

```
cd django-cloudlaunch
curl https://raw.githubusercontent.com/CloudVE/cloudlaunch-helm/master/
↪cloudlaunchserver/data/1_clouds.json --output clouds.json
python manage.py loaddata clouds.json
```

If we start the CloudLaunch server now and navigate to the admin console, `DJCLOUDBRIDGE -> Clouds`, we can see a list of cloud providers that have been loaded and CloudLaunch can target.

If you would like to add a new cloud provider to be included in either the hosted service or for distribution, please issue a pull request with the necessary connection properties to https://github.com/CloudVE/cloudlaunch-helm/blob/master/cloudlaunchserver/data/1_clouds.json

4.4.2 Loading appliances

Rather than loading application-specific information by hand, we can load apps from an application registry in bulk. At the moment, this action needs to be performed from the CloudLaunch admin console.

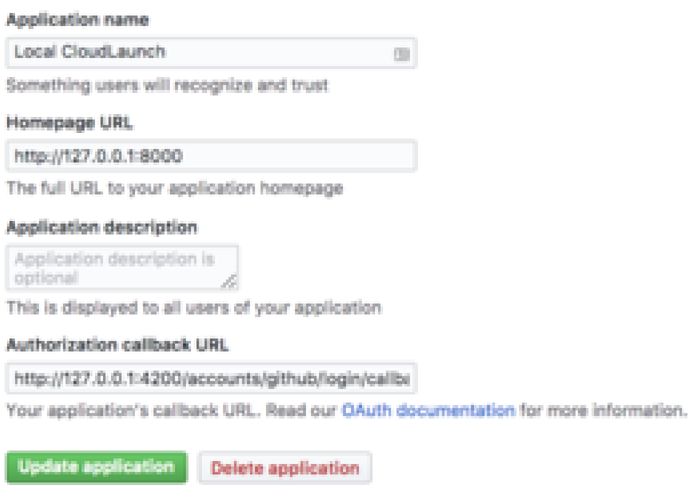
On the CloudLaunch admin console, head to the `CloudLaunch -> Applications` page and click `Add application` button in the top right corner. Provide an arbitrary name, say *placeholder*, for the application name and click save. Any information provided for this application will get overwritten with the information from the application registry. Back on the page listing applications, select the checkbox next to the newly created application and then from the `Action` menu, select `Import app data from url`. Click `Update` on the next page to load the default set of applications and your installation of CloudLaunch will have loaded all currently available apps.

4.5 Social Auth Setup

After you have setup the server, you will probably want to setup social auth to be able to log in using an external service. This setup is required for end-users so they can self register. If you are setting this up on localhost, use GitHub or Twitter.

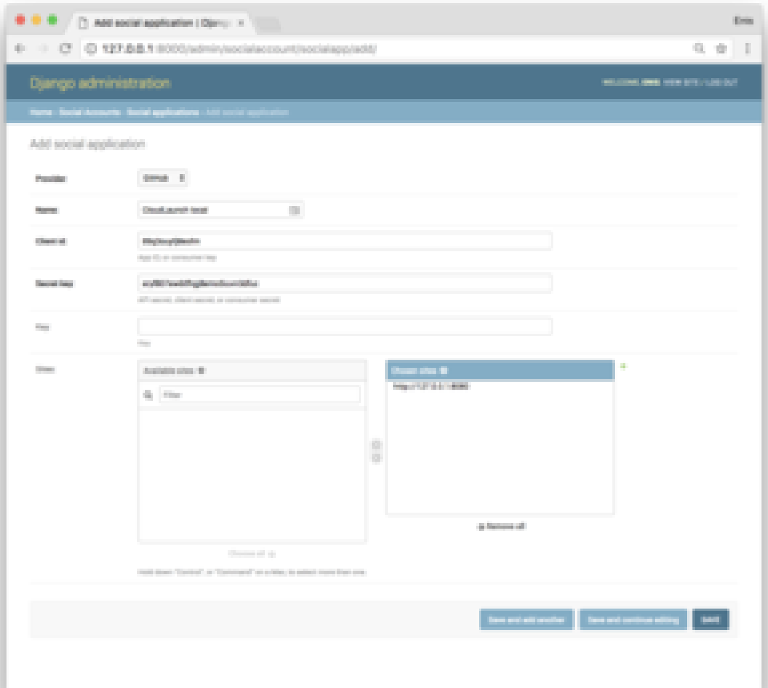
4.5.1 Integration with GitHub

1. Register your server with GitHub: Visit your Github account Settings → Developer settings and add a new OAuth application. Settings should look as in the following screenshot. Note port 4200 on the *Authorization callback URL*; this needs to match the port on which the CloudLaunch UI is served (4200 is the default). Also take note of the *Client ID* and *Client Secret* at the top of that page as we'll need that back in CloudLaunch.



2. Back on the local server, login to Django admin and change the domain of example.com in Sites to `http://127.0.0.1:8080`. To login to Admin, you'll need the superuser account info that was created when setting up the server.

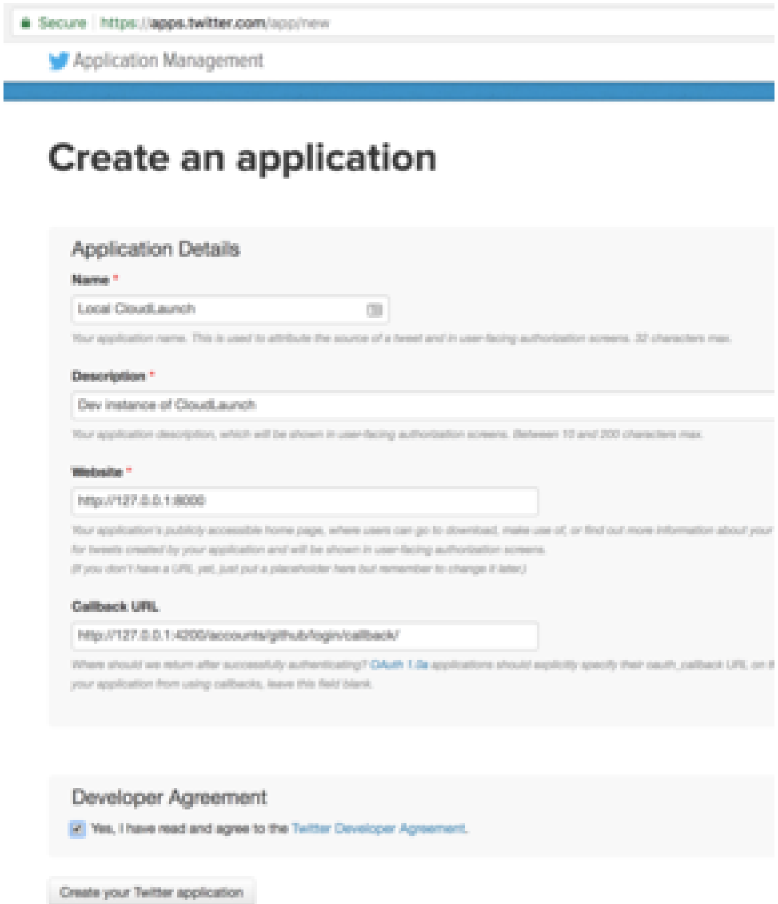
3. Still in Django Admin, now navigate to *Social Accounts* → *Social applications* and add a new application. Select GitHub as the provider, supply a desired application name, and enter the *Client ID* and *Client Secret* we got from GitHub. Also choose the site we updated in Step 2.



Save the model and integration with GitHub is complete! You can now log in to the CloudLaunch UI using Github.

4.5.2 Integration with Twitter

1. Register your dev server under your Twitter account. Visit https://apps.twitter.com/, click *Create New App*, and fill out the form as in the following screenthot. Once the app has been added, click on the *Keys and Access Tokens* tab and take a note of *Consumer Key (API Key)* and *Consumer Secret (API Secret)*.



2. Proceed with the same steps as in the docs about about GitHub integration, supplying the *Consumer Key (API Key)* and *Consumer Secret (API Secret)* as the values of *Client ID* and *Client Secret* for the new defintion of the Social application.