shub-image Documentation

Release 0.2.5

Scrapinghub

October 28, 2016

Contents

Contents:

CHAPTER 1

Deploy a custom image to Scrapy Cloud 2.0

Scrapy Cloud 2.0 is the newest Scrapy Cloud platform version which allows you to run Scrapy spiders in Docker containers. This document will show you how to use the shub-image command line tool to deploy custom Docker images for your Scrapy projects to Scrapy Cloud 2.0.

Note: If you don't need a custom Docker image, you can continue using shub to deploy your spiders. Just make sure to **upgrade** it before doing so:

```
$ pip install shub --upgrade
```

1.1 Installation

Install the `shub-image` tool via pip:

```
$ pip install shub-image
```

1.2 Deployment

This section describes how to build and deploy to Scrapy Cloud 2.0 a custom Docker image for a Scrapy project.

Warning: Make sure you are working on a Scrapy Cloud 2.0 project before following this guide. You can migrate your old projects via Scrapy Cloud web dashboard.

1.2.1 1. Initialization

Open up a terminal and go to your crawler's project folder in your local machine:

```
$ cd path/to/your/project
```

And then run the *init command*:

```
$ shub-image init --requirements path/to/requirements.txt
```

This will create a Dockerfile for your container including `requirements.txt` as a dependency and using python:2.7 as the base image for your custom image. If you want to use a different one, pass the `--base-image` option, like this:

```
$ shub-image init --base-image scrapinghub/base:12.04
```

In this case, it will use the image available at https://hub.docker.com/r/scrapinghub/base tagged with `12.04`.

Warning: Make sure to include scrapinghub-entrypoint-scrapy in your requirements.txt file. It is a support layer to pass all the job data to Scrapy inside a Mesos task. If you ignore this, your Scrapy projects won't work at Scrapy Cloud 2.0.

1.2.2 2. Define your target image

Now you need to define the Docker repository that will store the image built by this tool. To do this, open your project's `scrapinghub.yml` file and add an `images` section to it, like this:

```
projects:
    default: 29629
images:
    default: yourusername/repository
```

The settings above define that `shub-image` will push the image of your Docker container to https://hub.docker.com/r/yourusername/repository. You can also specify the complete URL for your repository if you are not using the default registry (which is https://hub.docker.com).

Tip: Your project might not have a `scrapinghub.yml` file, because it has been introduced with recent versions of shub. Make sure to upgrade your shub package by running:

```
$ pip install shub --upgrade
```

And then create `scrapinghub.yml` by running:

```
$ shub deploy
```

After this, don't forget to add the `images` section to it, since shub doesn't include it for you.

1.2.3 3. Build the image

Once you have the Dockerfile (generated in *step 1*) and your target image settings, run the *build* command to make `shub-image` build the Docker image for you:

```
$ shub-image build
The image yourusername/repository:1.0 build is completed.
```

After doing so, you can run the *test* command to make sure everything is alright for deployment:

```
$ shub-image test
```

1.2.4 4. Push the image to the registry

This step will push the image you just built to the repository defined in the `scrapinghub.yml` file. To do this, run the *push* command:

```
$ shub-image push
Pushing yourusername/repository:1.0 to the registry.
The image yourusername/repository:1.0 pushed successfully.
```

In the example above, the image was pushed to https://hub.docker.com/r/yourusername/repository.

1.2.5 5. Deploy your image to Scrapy Cloud 2.0

Once your image has been uploaded to the Docker registry, you have to deploy it to Scrapy Cloud 2.0 using the *deploy* command:

```
$ shub-image deploy
Deploy task results: <Response [302]>
You can check deploy results later with 'shub-image check --id 10'.
Deploy results:
 {u'status': u'started'}
 {u'status': u'progress', u'last_step': u'pulling'}
 {u'status': u'ok', u'project': 29629, u'version': u'1.0', u'spiders': 1}
```

Now you can schedule your spiders via both web dashboard or shub.

Warning: The deploy step for a project might be slow for the first time you do it.

1.3 Commands

Each of the commands we used in the steps above has some options that allow you to customize their behavior. For example, the *push* command allows you to pass your registry credentials via the `--username` and `--password` options. This section lists the options available for each command.

1.3.1 init

The first command you have to run when migrating your projects to run on Scrapy Cloud 2.0 is `shub-image init`. This command generates a `Dockerfile` to be used later by the *build* command to create a Docker container based on your Scrapy project.

The generated Dockerfile will likely fit your needs. But if it doesn't, it's just a matter of editing the file.

Options for init

`--project <text>`

Define the Scrapy project where the settings are going to be read from.

Default value: `default` from current folder's `scrapy.cfg`.

`--base-image <text>`

Define which base Docker image your custom image will build upon.

Default value: `python:2.7`

`--requirements <path>`

Set `path` as the Python requirements file for this project.

Default value: project directory `requirements.txt`

`--base-deps <list>`

Add system dependencies for your image, overriding the default ones. The `<list>` parameter should be a comma separated list with no spaces between dependencies.

Default value: `telnet,vim,htop,strace,iputils-ping,lsof`

`--add-deps <list>`

Provide additional system dependencies to install in your image along with the default ones. The `<list>` parameter should be a comma separated list with no spaces between dependencies.

`--list-recommended-reqs`

List recommended Python requirements for a Scrapy Cloud 2.0 project and exit.

Example:

```
$ shub-image init --base-image scrapinghub/base:12.04 \
--requirements other/requirements-dev.txt \
--add-deps phantomjs,tmux
```

1.3.2 build

This command uses the Dockerfile created by the *init* command to build the image that's going to be deployed later.

It reads the target images from the scrapinghub.yml file, which is generated by the deploy command from shub >= 2.0. You should add a section called `images` on it using the following format:

```
images:
    default: username/project
    private: your.own.registry:port/username/project
    fallback: anotheruser/project
```

Options for build

`--list-targets`

List available targets and exit.

`--target <text>`

Define the image for release. The `<text>` parameter must be one of the target names listed by `list-targets`.

Default value: `default`

`--version <text>`

Tag your image with `<text>`. You'll probably not need to set this manually, because the tool automatically sets this for you.

If you pass the `--version` parameter here, you will have to pass the exact same value to any other commands that accept this parameter (*push* and *deploy*).

Default value: identifier generated by shub.

`-d/--debug`

Increase the tool's verbosity.

Example:

```
$ shub-image build --list-targets
default
private
fallback
$ shub-image build --target private --version 1.0.4
```

1.3.3 push

This command pushes the image built by the `build` command to the registry (the `default` one or another one specified with the `--target option`).

Options for push

`--list-targets`

List available targets and exit.

`--target <text>`

Define the image for release. The `<text>` parameter must be one of the target's names listed by `list-targets`.

Default value: `default`

`--version <text>`

Tag your image with `<text>`. If you provided a custom version to the *build* command, make sure to provide the same value here.

Default value: identifier generated by shub.

`--username <text>`

Set the username to authenticate in the Docker registry.

Note: we don't store your credentials and you'll be able to use OAuth2 in the near future.

`--password <text>`

Set the password to authenticate in the Docker registry.

`--email <text>`

Set the email to authenticate in the Docker registry (if needed).

`--apikey <text>`

(beta) Use provided apikey to authenticate in the Scrapy Cloud Docker registry.

`--insecure`

Use the Docker registry in insecure mode.

`-d/--debug`

Increase the tool's verbosity.

Most of these options are related with Docker registry authentication. If you don't provide them, `shub-image` will try to push your image using the plain HTTP `--insecure-registry` docker mode.

Example:

```
$ shub-image push --target private --version 1.0.4 \
--username johndoe --password johndoepwd
```

This example authenticates the user `johndoe` to the registry `your.own.registry:port` (as defined in the *build command example*).

1.3.4 deploy

This command deploys your release image to Scrapy Cloud 2.0.

Options for deploy

`--list-targets`

List available targets and exit.

`--target <text>`

Target name that defines where the image is going to be pushed to.

Default value: `default`

`--version <text>`

The image version that you want to deploy to Scrapy Cloud 2.0. If you provided a custom version to the *build* and *push* commands, make sure to provide the same value here.

Default value: identifier generated by shub

`--username <text>`

Set the username to authenticate in the Docker registry.

Note: we don't store your credentials and you'll be able to use OAuth2 in the near future.

`--password <text>`

Set the password to authenticate in the registry.

`--email <text>`

Set the email to authenticate in the Docker registry (if needed).

`--apikey <text>`

(beta) Use provided apikey to authenticate in the Scrapy Cloud Docker registry.

`--insecure`

Use the Docker registry in insecure mode.

`--async`

Make deploy asynchronous. When enabled, the tool will exit as soon as the deploy is started in background. You can then check the status of your deploy task periodically via the *check* command.

Default value: `False`

`-d/--debug`

Increase the tool's verbosity.

Example:

```
$ shub-image deploy --target private --version 1.0.4 \
--username johndoe --password johndoepwd --async
```

This command will deploy the image from the `private` target, using user credentials passed as parameters and exit as soon as the deploy process starts (`--async`).

1.3.5 upload

It is a shortcut for the build -> push -> deploy chain of commands.

Example:

```
$ shub-image upload --target private --version 1.0.4 \
--username johndoe --password johndoepwd
```

Options for upload

The `upload` command accepts the same parameters as the *deploy* command.

1.3.6 check

This command checks the status of your deployment and is useful when you do the deploy in asynchronous mode.

By default, the `check` command will return results from the last deploy.

Options for check

`--id <number>`

The id of the deploy you want to check the status.

Default value: the id of the latest deploy.

Example:

```
$ shub-image check --id 0
```

This command above will check the status of the first deploy made (id 0).

1.3.7 test

This command checks if your local setup meets the requirements for a deployment at Scrapy Cloud 2.0. You can run it right after the *build command* to make sure everything is ready to go before you push your image with the *push command*.

Options for test

`--list-targets`

List available targets and exit.

`-d/--debug`

Increase the tool's verbosity.

1.4 Troubleshooting

1.4.1 Image not found while deploying

Make sure the repository you set in your `scrapinghub.yml` images section exists in the registry. Consider this `scrapinghub.yml` example file:

```
projects:
    default: 555555
images:
    default: johndoe/scrapy-crawler
```

`shub-image` will try to deploy the image to http://hub.docker.com/johndoe/scrapy-crawler, since hub.docker.com is the default registry. So, to make it work, you have to log into your account there and create the repository.

Otherwise, you are going to get an error message like this:

```
Deploy results: {u'status': u'error', u'last_step': u'pulling', u'error': u"DockerCmdFailure(u'Error
```

1.4.2 Uploading to a private repository

If you are using a private repository to push your images to, make sure to pass your registry credentials to both *push* and *deploy* commands:

```
$ shub-image push --username johndoe --password yourpass
$ shub-image deploy --username johndoe --password yourpass
```

1.4.3 ImportError while initializing the project

If you are getting an `ImportError` like this while running `shub-image init`:

```
...
    from shub import config as shub_config
ImportError: cannot import name config
```

You should make sure you have the latest version of shub installed by running:

```
$ pip install shub --upgrade
```

CHAPTER 2

Custom Images contract

A contract is a set of requirements that any crawler custom Docker image have to comply with to be able to run on Scrapy Cloud.

Scrapy crawler Docker images are already supported via the *scrapinghub-entrypoint-scrapy* contract implementation. If you want to run crawlers built using other framework/language than Scrapy/Python, you have to make sure your image follows the contract statements listed below.

2.1 Contract statements

1. Docker image should be able to run via `start-crawl` command without arguments.

```
docker run myscrapyimage start-crawl
```

2. Docker image should be able to return a spiders list via `list-spiders` command without arguments.

```
docker run myscrapyimage list-spiders
```

3. Crawler should be able to get all needed params using *system environment variables*.

2.2 Environment variables

2.2.1 SHUB_JOBKEY

Job key in format `PROJECT_ID/SPIDER_ID/JOB_ID`.

Example:

```
123/45/67
```

2.2.2 SHUB_JOB_DATA

Job arguments, in json format.

Example:

```
{"key": "1111112/2/2", "project": 1111112, "version": "version1",
"spider": "spider-name", "spider_type": "auto", "tags": [],
"priority": 2, "scheduled_by": "user", "started_by": "admin",
"pending_time": 1460374516193, "running_time": 1460374557448, ... }
```

2.2.3 SHUB_SETTINGS

Job settings (i.e. organization / project / spider / job settings), in json format.

There are several layers of settings, and they all serve to different needs.

The settings may contain the following sections (dict keys):

- `organization_settings`
- `project_settings`
- `spider_settings`
- `job_settings`
- `enabled_addons`

Organization / project / spider / job settings define appropriate levels of same settings but with different priorities. Enabled addons define Scrapinghub addons specific settings and may have an extended structure.

All the settings should replicate Dash API project `/settings/get.json` endpoint response (except `job_settings` if exists):

```
http -a APIKEY: http://dash.scrapinghub.com/api/settings/get.json project==PROJECTID
```

Note: All environment variables starting from `SHUB_` are reserved for Scrapinghub internal use and shouldn't be used with any other purposes (they will be dropped/replaced on a job start).

2.3 Scrapy entrypoint

A base support wrapper written in Python implementing Custom Images contract to run Scrapy-based python crawlers and scripts on Scrapy Cloud.

Main functions of this wrapper are the following:

- providing `start-crawl` entrypoint
- providing `list-spiders` entrypoint (starting from `0.7.0` version)
- translating system environment variables to Scrapy `crawl` / `list` commands

In fact, there are a lot of different features:

- parsing job data from environment
- processing job args and settings
- running a job with Scrapy
- collecting stats
- advanced logging & error handling
- transparent integration with Scrapinghub storage

- custom scripts support

scrapinghub-entrypoint-scrapy package is available on:

- PyPI
- Github

CHAPTER 3

Indices and tables

- genindex
- modindex
- search