shub Documentation

Release 2.15.4

Scrapinghub

Feb 12, 2024

CONTENTS

`shub` is the Scrapinghub command line client. It allows you to deploy projects or dependencies, schedule spiders, and retrieve scraped data or logs without leaving the command line.

CHAPTER

ONE

CONTENTS

1.1 Quickstart

1.1.1 Installation

If you have `pip` installed on your system, you can install shub from the Python Package Index:

```
pip install shub
```

We also supply stand-alone binaries. You can find them in our latest GitHub release.

1.1.2 Getting help

To see all available commands, run:

```
shub
```

For help on a specific command, run it with a `--help` flag, e.g.:

```
shub schedule --help
```

1.1.3 Basic usage

Start by logging in:

```
shub login
```

This will save your Scrapinghub API key to a file in your home directory (`~/.scrapinghub.yml`) and is necessary for access to projects associated with your Scrapinghub account. Alternatively, you can set your Scrapinghub API key as an environment variable (`SHUB_APIKEY`), check *an appropriate section* for details.

Next, navigate to a Scrapy project that you wish to upload to Scrapinghub. You can deploy it to Scrapy Cloud via:

```
shub deploy
```

On the first call, this will guide you through a wizard to save your project ID into a YAML file named `scrapinghub.yml`, living next to your `scrapy.cfg`. From anywhere within the project directory tree, you can now deploy via `shub deploy`.

Next, schedule one of your spiders to run on Scrapy Cloud:

```
shub schedule myspider
```

You can watch its log or the scraped items while the spider is running by supplying the job ID:

```
shub log -f 2/34
shub items -f 2/34
```

1.2 Configuration

1.2.1 Where to configure shub

shub is configured via two YAML files:

- `~/.scrapinghub.yml` – this file contains global configuration like your API key. It is automatically created in your home directory when you run `shub login`. You can also change the default location with an environment variable, check an appropriate section below.
- `scrapinghub.yml` – this file contains local configuration like the project ID or the location of your requirements file. It is automatically created in your project directory when you run `shub deploy` for the first time.

All configuration options listed below can be used in both of these configuration files. In case they overlap, the local configuration file will always take precedence over the global one.

1.2.2 Defining target projects

A very basic `scrapinghub.yml`, as generated when you first run `shub deploy`, could look like this:

```
project: 12345
```

This tells shub to deploy to the Scrapy Cloud project `12345` when you run `shub deploy`. Often, you will have multiple projects on Scrapy Cloud, e.g. one for development and one for production. For these cases, you can replace the `project` option with a `projects` dictionary:

```
projects:
  default: 12345
  prod: 33333
```

shub will now deploy to project `12345` when you run `shub deploy`, and deploy to project `33333` when you run `shub deploy prod`.

1.2.3 The configuration options

A deployed project contains more than your Scrapy code. Among other things, it has a version tag, and often has additional package requirements or is bound to a specific Scrapy version. All of these can be configured in `scrapinghub.yml`.

Sometimes the requirements may be different for different target projects, e.g. because you want to run your development project on Scrapy 1.3 but use Scrapy 1.0 for your production project. For these cases some options can be configured either *globally* or *project-specific*.

A *global* configuration option serves as default for all projects. E.g., to set `scrapy:1.3-py3` as default Scrapy Cloud stack, use:

```
projects:
  default: 12345
  prod: 33333

stack: scrapy:1.3-py3
```

If you wish to use the stack *only* for project `12345`, expand its entry in `projects` as follows:

```
projects:
  default:
    id: 12345
    stack: scrapy:1.3-py3
  prod: 33333
```

The following is a list of all available configuration options:

Option	Description	Scope
`requir`	Path to the project's requirements file, and to any additional eggs that should be deployed to Scrapy Cloud. See *Deploying dependencies*.	global default and project-specific
`stack`	Scrapy Cloud stack to use (this is the environment that your project will run in, e.g. the Scrapy version that will be used).	global default and project-specific
`image`	Whether to use a custom Docker image on deploy. See *Deploying custom Docker images*.	global default and project-specific
`versio`	Version tag to use when deploying. This can be an arbitrary string or one of the magic keywords `AUTO` (default), `GIT`, or `HG`. By default, `shub` will auto-detect your version control system and use its branch/commit ID as version.	global only
`apikey`	API key to use for deployments. You will typically not have to touch this setting as it will be configured inside `~/.scrapinghub.yml` in your home directory, via `shub login`.	global only

1.2.4 Configuration via environment variables

Your Scrapinghub API key can be set as an environment variable, it could be useful for noninteractive deploys (e.g. for CI workflow).

On Linux-based systems:

```
SHUB_APIKEY=0bbf4f0f691e0d9378ae00ca7bcf7f0c
```

On Windows:

```
SET SHUB_APIKEY=0bbf4f0f691e0d9378ae00ca7bcf7f0c
```

You can also parametrize global `scrapinghub.yml` file location with `SHUB_GLOBAL_CONFIG` environment variable (default `~/.scrapinghub.yml`).

When working with custom Docker images, please be aware that the tool relies on a set of standard `DOCKER_` prefixed environment variables:

DOCKER_HOST
The URL or Unix socket path used to connect to the Docker API.

DOCKER_API_VERSION
The version of the Docker API running on the host. Defaults to the latest version of the API supported by docker-py.

DOCKER_CERT_PATH
Specify a path to the directory containing the client certificate, client key and CA certificate.

DOCKER_TLS_VERIFY
Enables securing the connection to the API by using TLS and verifying the authenticity of the Docker Host.

1.2.5 Example configurations

Custom requirements file and fixed version information:

```
project: 12345
requirements:
  file: requirements_scrapinghub.txt
version: 0.9.9
```

Custom Scrapy Cloud stack, requirements file and additional private dependencies:

```
project: 12345
stack: scrapy:1.1
requirements:
  file: requirements.txt
  eggs:
    - privatelib.egg
    - path/to/otherlib.egg
```

Using the latest Scrapy 1.3 stack in staging and development, but pinning the production stack to a specific release:

```
projects:
  default: 12345
  staging: 33333
  prod:
    id: 44444
    stack: scrapy:1.3-py3-20170322

stack: scrapy:1.3-py3
```

Using a custom Docker image:

```
projects:
  default: 12345
  prod: 33333

image: true
```

Using a custom Docker image only for the development project:

```
projects:
  default:
    id: 12345
    image: true
  prod: 33333
```

Using a custom Docker image in staging and development, but a Scrapy Cloud stack in production:

```
projects:
  default: 12345
  staging: 33333
  prod:
    id: 44444
    image: false
    stack: scrapy:1.3-py3-20170322

image: true
```

Setting the API key used for deploying:

```
project: 12345
apikey: 0bbf4f0f691e0d9378ae00ca7bcf7f0c
```

1.2.6 Advanced use cases

It is possible to configure multiple API keys:

```
projects:
  default: 123
  otheruser: someoneelse/123

apikeys:
  default: 0bbf4f0f691e0d9378ae00ca7bcf7f0c
  someoneelse: a1aeecc4cd52744730b1ea6cd3e8412a
```

as well as different API endpoints:

```
projects:
  dev: vagrant/3

endpoints:
  vagrant: http://vagrant:3333/api/

apikeys:
  default: 0bbf4f0f691e0d9378ae00ca7bcf7f0c
  vagrant: a1aeecc4cd52744730b1ea6cd3e8412a
```

Global and project-specific requirements. `requirements.txt` is used for projects `prod` and `some`, `requirements-dev.txt` and eggs for `dev`:

```
projects:
  prod: 12345
```

(continues on next page)

(continued from previous page)

```
  dev:
      id: 345
      requirements:
          file: requirements-dev.txt
          eggs:
          - ./egg1.egg
          - ./egg2.egg
  some: 567
requirements:
  file: requirements.txt
stacks:
  default: "scrapy:2.8"
```

1.3 Deploying projects and dependencies

1.3.1 Deploying projects

To deploy a Scrapy project to Scrapy Cloud, navigate into the project's folder and run:

```
shub deploy [TARGET]
```

where `[TARGET]` is either a project name defined in `scrapinghub.yml` or a numerical Scrapinghub project ID. If you have configured a default target in your `scrapinghub.yml`, you can leave out the parameter completely:

```
$ shub deploy
Packing version 3af023e-master
Deploying to Scrapy Cloud project "12345"
{"status": "ok", "project": 12345, "version": "3af023e-master", "spiders": 1}
Run your spiders at: https://app.zyte.com/p/12345/
```

You can also deploy your project from a Python egg, or build one without deploying:

```
$ shub deploy --egg egg_name --version 1.0.0
Using egg: egg_name
Deploying to Scrapy Cloud project "12345"
{"status": "ok", "project": 12345, "version": "1.0.0", "spiders": 1}
Run your spiders at: https://app.zyte.com/p/12345/
```

```
$ shub deploy --build-egg egg_name
Writing egg to egg_name
```

1.3.2 Deploying dependencies

Sometimes your project will depend on third party libraries that are not available on Scrapy Cloud. You can easily upload these by specifying a requirements file:

```
# project_directory/scrapinghub.yml

projects:
  default: 12345
  prod: 33333

requirements:
  file: requirements.txt
```

Note that this requirements file is an *extension* of the Scrapy Cloud stack, and therefore should not contain packages that are already part of the stack, such as `scrapy`.

In case you use pipenv you may also specify a `Pipfile`:

```
# project_directory/scrapinghub.yml

projects:
  default: 12345
  prod: 33333

requirements:
  file: Pipfile
```

In this case the `Pipfile` must be locked and `pipenv` available in the environment.

Note: To install pipenv tool, use `pip install pipenv` or check its documentation.

A requirements.txt file will be created out of the `Pipfile` so like the requirements file above, it should not contain packages that are already part of the stack.

If you use Poetry you can specify your `pyproject.toml`:

```
# project_directory/scrapinghub.yml

projects:
  default: 12345
  prod: 33333

requirements:
  file: pyproject.toml
```

A `poetry.lock` file must be available, that will be used for determining the full requirements.

Note: Poetry is a tool for dependency management and packaging in Python.

When your dependencies cannot be specified in a requirements file, e.g. because they are not publicly available, you can supply them as Python eggs:

```
# project_directory/scrapinghub.yml

projects:
  default: 12345
  prod: 33333

requirements:
  file: requirements.txt
  eggs:
    - privatelib.egg
    - path/to/otherlib.egg
```

Alternatively, if you cannot or don't want to supply Python eggs, you can also build your own Docker image to be used on Scrapy Cloud. See *Deploying custom Docker images*.

1.3.3 Choosing a Scrapy Cloud stack

You can specify the Scrapy Cloud stack to deploy your spider to by adding a `stack` entry to your configuration:

```
# project_directory/scrapinghub.yml

projects:
  default: 12345
stack: scrapy:1.3-py3
```

It is also possible to define the stack per project for advanced use cases:

```
# project_directory/scrapinghub.yml

projects:
  default:
    id: 12345
    stack: scrapy:1.3-py3
  prod: 33333  # will use Scrapinghub's default stack
```

1.4 Scheduling jobs and fetching job data

shub allows you to schedule a spider run from the command line:

```
shub schedule SPIDER
```

where `SPIDER` should match the spider's name. By default, shub will schedule the spider in your default project (as defined in `scrapinghub.yml`). You may also explicitly specify the project to use:

```
shub schedule project_alias_or_id/SPIDER
```

You can supply spider arguments and job-specific settings through the `-a` and `-s` options:

```
$ shub schedule myspider -a ARG1=VALUE -a ARG2=VALUE
Spider myspider scheduled, job ID: 12345/2/15
Watch the log on the command line:
```

(continues on next page)

(continued from previous page)

```
    shub log -f 2/15
or print items as they are being scraped:
    shub items -f 2/15
or watch it running in Scrapinghub's web interface:
    https://app.zyte.com/p/12345/job/2/15
```

```
$ shub schedule 33333/myspider -s LOG_LEVEL=DEBUG
Spider myspider scheduled, job ID: 33333/2/15
Watch the log on the command line:
    shub log -f 2/15
or print items as they are being scraped:
    shub items -f 2/15
or watch it running in Scrapinghub's web interface:
    https://app.zyte.com/p/33333/job/2/15
```

You can also specify the amount of Scrapy Cloud units (`-u`) and the priority (`-p`):

```
$ shub schedule myspider -p 3 -u 3
Spider myspider scheduled, job ID: 12345/2/16
Watch the log on the command line:
    shub log -f 2/16
or print items as they are being scraped:
    shub items -f 2/16
or watch it running in Scrapinghub's web interface:
    https://app.zyte.com/p/12345/job/2/16
```

shub provides commands to retrieve log entries, scraped items, or requests from jobs. If the job is still running, you can provide the `-f` (follow) option to receive live updates:

```
$ shub log -f 2/15
2016-01-02 16:38:35 INFO Log opened.
2016-01-02 16:38:35 INFO [scrapy.log] Scrapy 1.0.3.post6+g2d688cd started
...
# shub will keep updating the log until the job finishes or you hit CTRL+C
```

```
$ shub items 2/15
{"name": "Example product", description": "Example description"}
{"name": "Another product", description": "Another description"}
```

```
$ shub requests 1/1/1
{"status": 200, "fp": "1ff11f1543809f1dbd714e3501d8f460b92a7a95", "rs": 138137, "_key":
↪"1/1/1/0", "url": "http://blog.scrapinghub.com", "time": 1449834387621, "duration":␣
↪238, "method": "GET"}
{"status": 200, "fp": "418a0964a93e139166dbf9b33575f10f31f17a1", "rs": 138137, "_key":
↪"1/1/1/0", "url": "http://blog.scrapinghub.com", "time": 1449834390881, "duration":␣
↪163, "method": "GET"}
```

1.5 Deploying custom Docker images

Note: This feature is currently only available for paying customers.

It's possible to deploy Docker images with spiders to Scrapy Cloud. To be able to run spiders in custom Docker images it's necessary to follow the *Custom images contract* - a set of requirements that image should comply with to be compatible with Scrapy Cloud.

1.5.1 Deployment

This section describes how to build and deploy a custom Docker image to Scrapy Cloud. For all the following steps it's assumed that commands are executed at the root directory of your project.

1. Create Dockerfile

The most important thing you need to be able to build and deploy Docker images is a Dockerfile. Please follow the link if you are not familiar with the concept as it's crucial to understand it while using custom Docker images feature.

If you want to migrate an existing Scrapy project - there's a tool that may help you, please read *this section*. In all other cases you're responsible for writing your own Dockerfile. The resulting Dockerfile should produce a Docker image that follows the *Custom images contract* - follow the link to find an example Dockerfile.

2. Deploy to Scrapy Cloud

Once you have the Dockerfile run the *shub deploy* command to build the Docker image. If there's no *scrapinghub.yml* configuration file at the project root shub will start a wizard that will help to configure the project and will save the configuration file. If you already have *scrapinghub.yml* at the project root please ensure that *image deploy is configured* for the target project. If the target project already exists in the configuration file but images deploy is not configured you can run *shub image build* to build the image for the first time and shub will help you to configure the image repository.

The deploy consists of 3 stages which are described below. Normally *shub deploy* will execute all 3 stages in a single run, but in some cases in might be useful to run those stages separately, so there are commands bundled under *shub image* that allow to execute different stages separately.

Build

During the build stage Docker image is built from the given Dockerfile. This stage can be manually started with *shub image build* command:

```
$ shub image build
...
The image images.scrapinghub.com/project/XXXXXX:YYYYYY build is completed.
```

In the end of the command, shub will automatically run a few tests to make sure everything is alright for deployment. You can run the test manually after the build:

```
$ shub image test
```

Note: If you want to access Docker build logs you can invoke the command in the verbose mode:

```
$ shub image build -v
```

Push

During the push stage the image is pushed to the repository defined in the *scrapinghub.yml* file. This stage can be manually started with *shub image push* command:

```
$ shub image push
...
The image images.scrapinghub.com/project/XXXXXX:YYYYYY pushed successfully.
```

In the example above, the image was pushed to the default Scrapinghub images registry `images.scrapinghub.com`.

Note: If you want to access Docker push logs you can invoke the command in the verbose mode:

```
$ shub image push -v
```

Deploy

During the deploy stage the image is deployed to the Scrapy Cloud. This stage can be manually started with *shub image deploy* command:

```
$ shub image deploy
...
You can check deploy results later with 'shub image check --id 1'.
Deploy results:
 {'status': 'started'}
 {'project': XXXXXX, 'status': 'ok', 'version': 'YYYYYY', 'spiders': 1}
```

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

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

1.5.2 Create Docker image for existing Scrapy project

If you have an existing Scrapy project and you want to run it using a custom Docker image you'll need to create a Dockerfile for it. There's a *shub image init* command that creates a template Dockerfile, which should be suitable for the majority of the Scrapy projects that run on Scrapy Cloud:

```
$ shub image init
```

If your project has `requirements.txt` file you can easily add it like this:

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

Warning: If you have a Scrapy project but don't want to use the generated Dockerfile or need to use a different base image you may want to install scrapinghub-entrypoint-scrapy Python package inside your image. It is a support layer that passes data from the job to Scrapinghub storage. Otherwise you will need to send data to Scrapinghub storage using HTTP API.

1.5.3 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.

build

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

It reads the target images from the *scrapinghub.yml* file. You should add a section called `images` on it using the following format:

```
projects:
  default: 11111
  prod: 22222
# image deploy is enabled for all targets
image: true
```

Or:

```
projects:
  default:
    id: 12345
    # image deploy is enabled only for default target
    image: true
  prod: 33333
```

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`

`-V/--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 `-V/--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.

-S/--skip-tests

Option to skip testing image with `shub image test` after build.

-v/--verbose

Increase the tool's verbosity.

-f/--file

Use this option to pass a custom Dockerfile name (default is 'PATH/Dockerfile').

Default value: `Dockerfile`

Example:

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

push

This command pushes the image built by the `build` command to the registry (the `default` 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`

-V/--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>`

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

`--insecure`

Use the Docker registry in insecure mode.

`-v/--verbose`

Increase the tool's verbosity.

Most of these options are related with Docker registry authentication. If you don't provide them, shub 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*).

deploy

This command deploys your release image to Scrapy Cloud.

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`

`-V/--version <text>`

The image version that you want to deploy to Scrapy Cloud. 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>

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

--insecure

Use the Docker registry in insecure mode.

--async

Warning: Deploy in asynchronous mode is deprecated.

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`

-v/--verbose

Increase the tool's verbosity.

Example:

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

This command will deploy the image from the `private` target, using user credentials passed as parameters.

upload

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

Example:

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

Options for upload

The `upload` command accepts the same parameters as the *deploy* command, except for `--target`, which can be passed as an argument.

check

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

Warning: Deploy in asynchronous mode is deprecated.

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).

test

This command checks if your local setup meets the requirements for a deployment at Scrapy Cloud. 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.

`--target <text>`

Target name that defines an image that is going to be tested.

Default value: `default`

`-V/--version <text>`

The image version that you want to test. If you provided a custom version to the *deploy*, make sure to provide the same value here.

`-v/--verbose`

Increase the tool's verbosity.

list

This command lists spiders for your project based on the image you built and your project settings in Dash. You can run it right after the *build command* to make sure that all your spiders are found.

Options for list

`--list-targets`

List available targets and exit.

`--target <text>`

Target name that defines an image to get spiders list.

Default value: `default`

`-V/--version <text>`

The image version that you want to use to extract spiders list. If you provided a custom version to the *deploy*, make sure to provide the same value here.

`-s/--silent-mode`

Silent mode to suspend errors in a case if project isn't found for a given target in *scrapinghub.yml*.

`-v/--verbose`

Increase the tool's verbosity.

init

This command helps to migrate existing Scrapy projects to custom Docker images. It generates a `Dockerfile` that can be used later by the *build* or *upload* commands.

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`

`--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 project and exit.

Example:

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

1.5.4 Troubleshooting

Image not found while deploying

If you don't use default Scrapinghub repository - make sure the repository you set in your *scrapinghub.yml* images section exists in the registry. Consider this example:

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

shub will try to deploy the image to http://hub.docker.com/johndoe/scrapy-crawler, since hub.docker.com is the default Docker 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: image johndoe/scrapy-crawler not found',)"}
```

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
```

Or pass it to *upload* command:

```
$ shub image upload --username johndoe --password yourpass
```

Container works locally but fails in Scrapy Cloud

Prior to running `start-crawl` in Scrapy Cloud, some configurations are set to ensure we can run an isolated process. This can lead to issues that are quite hard to debug and find the root cause. To aid in this process, below you will find some steps that are quite similar to what actually runs in Scrapy Cloud.

Run your container in interactive mode with `bash` (or any other terminal that is available). Please replace the 2 occurrences of `<SPIDER-NAME>` with the actual spider that is to run:

```
$ docker run \
-it \
-e SHUB_JOBKEY=123/4/5 \
-e SHUB_JOB_DATA='{
    "_shub_worker": "kumo",
    "api_url": "https://app.zyte.com/api/",
    "auth": "SOME AUTH KEY NOT REQUIRED FOR THIS TEST",
    "deploy_id": 1,
    "key": "123/4/5",
    "pending_time": 1632739881823,
    "priority": 2,
    "project": 123,
    "running_time": 1632739882059,
```

(continues on next page)

(continued from previous page)

```
    "scheduled_by": "some_user",
    "spider": "<SPIDER-NAME>",
    "spider_type": "manual",
    "started_by": "jobrunner",
    "state": "running",
    "tags": [],
    "units": 1,
    "version": "1.0"
}' \
-e SHUB_JOB_ENV='{}' \
-e SHUB_JOB_MEMORY_LIMIT=950 \
-e SHUB_JOB_UID=123 \
-e SHUB_SETTINGS='{
    "deploy_id": 1,
    "enabled_addons": [],
    "job_settings": {},
    "organization_settings": {},
    "project_settings": {},
    "spider_settings": {},
    "status": "ok",
    "version": "1.0"
}' \
-e SHUB_SPIDER=<SPIDER-NAME> \
--net bridge \
--volume=/scrapinghub \
--rm=true \
--name=scrapy-cloud-container \
my-docker-image \
/bin/bash
```

Connect to the container in a new terminal window and open a named pipe to communicate through `sh_scrapy`:

```
$ docker exec -it scrapy-cloud-container /bin/bash
$ mkfifo -m 0600 /dev/scrapinghub
$ chown 65534:65534 /dev/scrapinghub
$ cat /dev/scrapinghub
```

Go back to the first window and start the crawling process:

```
$ export SHUB_FIFO_PATH=/dev/scrapinghub
$ start-crawl
```

Switch back to the second window (the named pipe one) to see the results comming out.

1.6 Custom Images contract

This is a set of requirements that any custom Docker image has 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. This means you have to implement your own scripts following the specification below. You can find example projects written in other frameworks and programming languages in the custom-images-examples repository. The `shub bootstrap` can be used to clone these projects.

1.6.1 Contract statements

1. Docker image should be able to run via `start-crawl` command without arguments. `start-crawl` should be *executable and located on the search path*.

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

   Crawler will be started by unpriviledged user `nobody` in a writable directory `/scrapinghub`. `HOME` environment variable will be set to `/scrapinghub` as well. Beware that this directory is added dynamically when job starts, if Docker image contains this directory - it'll be erased.

2. Docker image should be able to return its metadata via `shub-image-info` command without arguments. `shub-image-info` should be *executable and located on the search path*. For now only a few fields are supported, and all of them are required:

- `project_type` - a string project type, one of [`scrapy`, `casperjs`, `other`],
- `spiders` - a list of non-empty string spider names.

```
docker run myscrapyimage shub-image-info
{"project_type": "casperjs", "spiders": ["spiderA", "spiderB"]}
```

Note: `shub-image-info` is an extension (and a replacement) for a former `list-spiders` command to provide metadata in a structured form allowing to simplify non-Scrapy development and parametrize custom images in a more configurable way.

The command could also handle optional `--debug` flag by returning debug information about the image inside of an additional `debug` field: a name/version of operation system, installed packages etc. For example, for a Python-based custom image it could be a good idea to include `pip freeze` call results. Data format of the `debug` field is plain text, not structured to keep it simple.

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

Note: The simplest way to place scripts on the search path is to create a symbolic link to the script located in the directory present in the PATH environment variable. Here's an example Dockerfile:

```
FROM python:3
RUN mkdir -p /spiders
WORKDIR /spiders
ADD . /spiders
# Create a symbolic link in /usr/sbin because it's present in the PATH
RUN ln -s /spiders/start-crawl /usr/sbin/start-crawl
RUN ln -s /spiders/shub-image-info /usr/sbin/shub-image-info
```

(continues on next page)

(continued from previous page)

```
# Make scripts executable
RUN chmod +x /spiders/start-crawl /spiders/shub-image-info
```

1.6.2 Environment variables

SHUB_SPIDER

Spider name.

Example:

```
test-spider
```

SHUB_JOBKEY

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

Example:

```
123/45/67
```

SHUB_JOB_DATA

Job arguments, in JSON format.

Example:

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

Some useful fields

Field	Description	Example
key	Job key in format `PROJECT_ID/SPIDER_ID/JOB_ID`	`"1111112/2/2"`
project	Integer project ID	`1111112`
spider	String spider name	`"spider-name"`
job_cmd	List of string arguments for the job	`["--flagA", "--key1=value1"]`
spider_args	Dictionary with spider arguments	`{"arg1": "val1"}`
version	String project version used to run the job	`"version1"`
deploy_id	Integer project deploy ID used to run the job	`253`
units	Amount of units used by the job	`1`
priority	Job priority value	`2`
tags	List of string tags for the job	`["tagA", "tagB"]`
state	Job current state name	`"running"`
pending_time	UNIX timestamp when the job was added, in milliseconds	`1460374516193`
running_time	UNIX timestamp when the job was started, in milliseconds	`1460374557448`
scheduled_by	Username who scheduled the job	`"john"`

If you specified some custom metadata with `meta` field when scheduling the job, the data will also be in the dictionary.

Warning:

`SHUB_JOB_DATA` may contain other undocumented fields. They are for the platform's internal use
and are not part of the contract, i.e. they can appear or be removed anytime.

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).

1.6.3 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 `shub-image-info` entrypoint (starting from `0.11.0` 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

1.6.4 Scrapy addons

If you have Scrapy addons enabled in Dash UI, you may encounter with the similar errors:

```
[sh_scrapy.settings] Addon import error scrapy_pagestorage.PageStorageMiddleware:  No␣
↪module named scrapy_pagestorage
```

As you are in control of managing your Docker image content, you should add all missing packages by yourself to `requirements.txt` file (including dependencies related with the Scrapy addons), or disable corresponding addons in Dash UI.

1.7 Changes

1.7.1 2.15.4 (2024-02-08)

- Support Docker server 25+.

1.7.2 2.15.3 (2024-01-23)

- Fix `shub image deploy` failing on Python 3.8 and 3.9.

1.7.3 2.15.2 (2024-01-17)

- Add support for Python 3.12.
- Remove remnants of Python 2 support.
- Start a changelog.