shpkpr

Release 1.0.0

Jun 02, 2017

Contents

shpkpr is a tool for controlling and observing applications/tasks running on Marathon and Chronos. shpkpr is designed to provide a simple command-line interface to Marathon and Chronos (similiar to the `heroku` command-line tool) for use both manually and with CI tools like jenkins.

- Free software: MIT license
- Documentation: https://shpkpr.readthedocs.org.

CHAPTER 1

Features

- List/show detailed application info
- Deploy applications (using Jinja2 templates)
- Zero-downtime application deploys when used with Marathon-LB
- List/show detailed cron task info
- Deploy cron tasks (using Jinja2 templates)

CHAPTER 2

Contents

Installation

Using Pip

At the command line:

```
$ pip install shpkpr
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv shpkpr
$ pip install shpkpr
```

If you prefer to use shpkpr from git (**WARNING:** May be unstable), then:

```
$ pip install -e git+https://github.com/shopkeep/shpkpr.git#egg=shpkpr
```

Once installed, shpkpr should be available on your `$PATH`:

```
$ shpkpr --marathon_url=http://marathon.mydomain.com:8080 show
```

Using Docker

shpkpr is also available in a prebuilt Docker image if you'd prefer to run it in a container. A list of the available tags can be found on the Docker hub:

```
$ docker pull shopkeep/shpkpr:v1.0.0
$ docker pull shopkeep/shpkpr:master
```

Once the image is downloaded, you can use shpkpr with `docker run`:

```
$ docker run -ti shopkeep/shpkpr:master shpkpr --marathon_url=http://marathon.
↪mydomain.com:8080 show
```

A simple way to avoid having to repeat the long `docker run` command is to use a bash alias:

```
$ alias shpkpr="docker run -ti -e SHPKPR_MARATHON_URL=http://marathon.mydomain.
↪com:8080 shopkeep/shpkpr:master shpkpr"
$ shpkpr show
```

Usage

shpkpr is mostly self-documenting, and documentation for any command can be viewed by passing the `--help` parameter, e.g.:

```
$ shpkpr --help
Usage: shpkpr [OPTIONS] COMMAND [ARGS]...

  A tool to manage applications running on Marathon.

Options:
  --marathon_url TEXT      URL of the Marathon API to use.
  --help                   Show this message and exit.

Commands:
  deploy  Deploy application from template.
  show    Show application details.
```

For more detailed usage instructions use the sections linked below:

Configuring shpkpr

Environment Variables

All configuration options and arguments are fully documented in the `--help` screen for each command, however, `shpkpr` additionally allows all options to be specified in environment variables with the prefix `SHPKPR_`.

For example, to avoid having to specify the Marathon API URL with each command, you could do:

```
$ export SHPKPR_MARATHON_URL=http://my.marathon.install.mydomain.com:8080
$ shpkpr list
myapp1
my-other-app
yet-another-app
```

Which is functionally equivalent to:

```
$ shpkpr --marathon_url=http://my.marathon.install.mydomain.com:8080 list
myapp1
my-other-app
yet-another-app
```

Options specified on the command line will always take precedence over those in the environment.

Showing Application Details

`shpkpr show` allows you to view details of one or all applications that are currently deployed to Marathon:

```
$ shpkpr show --help
Usage: shpkpr show [OPTIONS]

  Shows detailed information for one or all applications.

Options:
--output-format [json|yaml]  Serialisation format to use when printing
                             application data to stdout.
-a, --application TEXT       ID/name of the application to scale.
--marathon_url TEXT          URL of the Marathon API to use.  [required]
--help                       Show this message and exit.
```

Required Configuration

Marathon URL:

URL of the Marathon API to use, e.g. `http://marathon.mydomain.com:8080`

- Environment variable: `SHPKPR_MARATHON_URL`
- Command-line flag: `--marathon_url`

Optional Configuration

Application ID:

ID of the application to show, e.g. `my-app`. If not provided `shpkpr` will return a list of *all* currently deployed applications.

- Environment variable: `SHPKPR_APPLICATION`
- Command-line flag: `--application`
- Default: `None`

Output Format:

Serialisation format used when printing details to stdout, e.g. `yaml`

- Environment variable: `SHPKPR_OUTPUT_FORMAT`
- Command-line flag: `--output-format`
- Default: `json`

Examples

```
$ shpkpr show -a my-app
{
    "args": null,
    "cmd": null,
    "constraints": [
        [
            "hostname",
```

```
            "UNIQUE"
        ]
    ],
    "container": {
        "docker": {
            "forcePullImage": false,
            "image": "my-docker-registry.com/my-app:some-tag",
            "network": "BRIDGE",
            "parameters": [],
            "portMappings": [
                {
                    "containerPort": 4999,
                    "hostPort": 0,
                    "labels": {},
                    "protocol": "tcp",
                    "servicePort": 11070
                }
            ],
            "privileged": false
        },
        "type": "DOCKER",
        "volumes": []
    },
    "cpus": 0.1,
    "deployments": [],
    "env": {
        "DEPLOY_ENVIRONMENT": "staging",
        "NEW_RELIC_ENVIRONMENT": "staging",
    },
    "id": "/my-app",
    "instances": 4,
    "mem": 512,
    "tasks": [
        {
            "appId": "/my-app",
            "healthCheckResults": [
                {
                    "alive": true,
                    "consecutiveFailures": 0,
                    "firstSuccess": "2017-04-19T10:03:59.470Z",
                    "lastFailure": null,
                    "lastFailureCause": null,
                    "lastSuccess": "2017-04-24T13:57:07.387Z",
                    "taskId": "my-app.7b11d1f6-24e7-11e7-83e7-0a9211a8d240"
                }
            ],
            "host": "10.210.84.170",
            "id": "my-app.7b11d1f6-24e7-11e7-83e7-0a9211a8d240",
            "ipAddresses": [
                {
                    "ipAddress": "172.17.0.4",
                    "protocol": "IPv4"
                }
            ],
            "ports": [
                31039
            ],
            "slaveId": "ef806cb6-4ccd-46ae-b507-c47a793e0379-S13",
```

```
            "stagedAt": "2017-04-19T10:03:47.890Z",
            "startedAt": "2017-04-19T10:03:52.157Z",
            "version": "2017-02-23T12:14:50.238Z"
        },
        {
            "appId": "/my-app",
            "healthCheckResults": [
                {
                    "alive": true,
                    "consecutiveFailures": 0,
                    "firstSuccess": "2017-04-19T10:08:29.979Z",
                    "lastFailure": "2017-04-19T10:17:46.053Z",
                    "lastFailureCause": "AskTimeoutException: Ask timed out on␣
↪[Actor[akka://marathon/user/IO-HTTP#-1526159107]] after [5000 ms]",
                    "lastSuccess": "2017-04-24T13:57:07.388Z",
                    "taskId": "my-app.1c8db1d0-24e8-11e7-83e7-0a9211a8d240"
                }
            ],
            "host": "10.210.68.176",
            "id": "my-app.1c8db1d0-24e8-11e7-83e7-0a9211a8d240",
            "ipAddresses": [
                {
                    "ipAddress": "172.17.0.7",
                    "protocol": "IPv4"
                }
            ],
            "ports": [
                31241
            ],
            "slaveId": "ef806cb6-4ccd-46ae-b507-c47a793e0379-S12",
            "stagedAt": "2017-04-19T10:08:18.815Z",
            "startedAt": "2017-04-19T10:08:23.050Z",
            "version": "2017-02-23T12:14:50.238Z"
        },
        {
            "appId": "/my-app",
            "healthCheckResults": [
                {
                    "alive": true,
                    "consecutiveFailures": 0,
                    "firstSuccess": "2017-04-19T10:13:30.585Z",
                    "lastFailure": null,
                    "lastFailureCause": null,
                    "lastSuccess": "2017-04-24T13:57:07.387Z",
                    "taskId": "my-app.d06e4756-24e8-11e7-83e7-0a9211a8d240"
                }
            ],
            "host": "10.210.93.193",
            "id": "my-app.d06e4756-24e8-11e7-83e7-0a9211a8d240",
            "ipAddresses": [
                {
                    "ipAddress": "172.17.0.2",
                    "protocol": "IPv4"
                }
            ],
            "ports": [
                31498
            ],
```

```
            "slaveId": "ef806cb6-4ccd-46ae-b507-c47a793e0379-S20",
            "stagedAt": "2017-04-19T10:13:20.600Z",
            "startedAt": "2017-04-19T10:13:21.469Z",
            "version": "2017-02-23T12:14:50.238Z"
        },
        {
            "appId": "/my-app",
            "healthCheckResults": [
                {
                    "alive": true,
                    "consecutiveFailures": 0,
                    "firstSuccess": "2017-04-19T10:29:52.497Z",
                    "lastFailure": null,
                    "lastFailureCause": null,
                    "lastSuccess": "2017-04-24T13:57:07.387Z",
                    "taskId": "my-app.1ca3a37c-24eb-11e7-83e7-0a9211a8d240"
                }
            ],
            "host": "10.210.59.63",
            "id": "my-app.1ca3a37c-24eb-11e7-83e7-0a9211a8d240",
            "ipAddresses": [
                {
                    "ipAddress": "172.17.0.2",
                    "protocol": "IPv4"
                }
            ],
            "ports": [
                31390
            ],
            "slaveId": "ef806cb6-4ccd-46ae-b507-c47a793e0379-S21",
            "stagedAt": "2017-04-19T10:29:47.450Z",
            "startedAt": "2017-04-19T10:29:48.304Z",
            "version": "2017-02-23T12:14:50.238Z"
        }
    ],
    "tasksRunning": 4,
    "tasksUnhealthy": 0,
    "version": "2017-02-23T12:14:50.238Z"
}
```


Deploying a new application config

`shpkpr deploy` allows you to deploy a new (or changes to an existing) application configuration by rendering and POSTing a JSON template to Marathon:

```
$ shpkpr deploy --help
Usage: shpkpr deploy [OPTIONS] [ENV_PAIRS]...

  Deploy application from template.

Options:
  --force                 Force update even if a deployment is in progress.
  --dry-run               Enables dry-run mode. Shpkpr will not attempt to
                          contact Marathon when this is enabled.
  -t, --template TEXT     Name of the template to use for deployment (default: built-
↪in template).
```

```
  --template_dir TEXT    Base directory in which your templates are stored (default:␣
↪`pwd`).
  -e, --env_prefix TEXT  Prefix used to restrict environment vars used for
                         templating.
  --marathon_url TEXT    URL of the Marathon API to use.  [required]
  --help                 Show this message and exit.
```

Additional template values can be passed on the command line in a KEY=VALUE format. Any number of key/value pairs can be passed. See below for a few examples.

Required Configuration

Marathon URL:

URL of the Marathon API to use, e.g. `http://marathon.mydomain.com:8080`

- Environment variable: `SHPKPR_MARATHON_URL`
- Command-line flag: `--marathon_url`

Optional Configuration

Force:

Using the force flag allows the user to initiate a deployment even if another deployment is currently in progress. This option should only be used in the case of a previous failed deployment as it *may* leave the app in an inconsistent state if anything goes wrong.

- Command-line flag: `--force`

Dry Run:

Using the dry-run flag allows the user to test a deployment before performing it against a live Marathon instance. When enabled, shpkpr will not attempt to contact the configured Marathon server.

- Command-line flag: `--dry-run`

Environment Variable Prefix:

When reading variables from the environment to inject into the template at render time, only those variables which begin with the specified prefix are considered. The prefix is stripped from the variable names before injecting into the template context.

Note: The `env_prefix` in this case controls only the prefix used for collecting environment variables for templating purposes. The `SHPKPR_` prefix is **always** used for regular configuration as documented elsewhere.

- Environment variable: `SHPKPR_ENV_PREFIX`
- Command-line flag: `--env_prefix`
- Default: `SHPKPR_`

Base Template Directory:

Absolute path to the base directory in which templates are stored. By default this is `pwd` but can be overridden to allow reading templates from any location on the filesystem. This setting is useful when using templates not found in `pwd` or controlling exactly how template inheritance should work. The specified directory is passed to a `jinja2.FilesystemLoader` within `shpkpr`.

- Command-line flag: `--template_dir`

JSON Template Name:

Name of the JSON template to use for deployment, e.g. `a/template.json.tmpl` or `my-template.json.tmpl`. This path should always be relative to the template base directory defined by `--template-dir` (`pwd` by default).

- Environment variable: `SHPKPR_TEMPLATE`
- Command-line flag: `--template`

If no template is explicitly provided then the default template is used (`marathon/default/standard.json.tmpl`).

NOTE: When running Marathon >0.13.0 it is possible to deploy multiple applications at the same time either by specifying the `--template` option multiple times, or by using a single template which contains a list of individual applications.

Examples

```
$ cat deploy.json.tmpl
{
  "id": "{{APPLICATION}}",
  "cmd": "{{CMD}}",
  "cpus": {{CPUS|require_float(min=0.0, max=2.0)}},
  "mem": 512,
  "instances": {{INSTANCES|require_int(min=1, max=16)}},
  "labels": {
    "DOMAIN": "{{LABEL_DOMAIN}}",
    "RANDOM_LABEL": "{{RANDOM_LABEL}}"
  }
}

$ export SHPKPR_APPLICATION=my-app
$ export SHPKPR_CPUS=0.1
$ export SHPKPR_INSTANCES=2
$ export SHPKPR_CMD="sleep 60"
$ export SHPKPR_LABEL_DOMAIN=mydomain.com

$ shpkpr deploy -t deploy.json.tmpl RANDOM_LABEL=my_value

# Would result in the following output sent to Marathon
# {
#   "id": "my-app",
#   "cmd": "sleep 60",
#   "cpus": 0.1,
#   "mem": 512,
#   "instances": 2,
#   "labels": {
#     "DOMAIN": "mydomain.com",
#     "RANDOM_LABEL": "my_value"
#   }
# }
```

```
$ cat deploy.json.tmpl
{
  "id": "my-application",
  "cmd": "sleep 60",
  "cpus": 0.1,
```

```
  "mem": 512,
  "instances": 1,
  "labels": {
    {% for k, v in _all_env|filter_items("LABEL_", True) %}
    "{{ k }}": "{{ v }}"{% if loop.last == False %},{% endif %}
    {% endfor %}
  }
}

$ export LABEL_DOMAIN=mydomain.com
$ export LABEL_NODE_TYPE=webserver
$ export LABEL_FAVORITE_ICECREAM_FLAVOR=vanilla

$ shpkpr deploy -t deploy.json.tmpl -e ""

# Would result in the following output sent to Marathon
# {
#   "id": "my-application",
#   "cmd": "sleep 60",
#   "cpus": 0.1,
#   "mem": 512,
#   "instances": 1,
#   "labels": {
#     "DOMAIN": "mydomain.com",
#     "NODE_TYPE": "webserver",
#     "FAVORITE_ICECREAM_FLAVOR": "vanilla"
#   }
# }
```

Managing Chronos Jobs

`shpkpr cron` Allows one to list, add, delete, update, run and terminate running tasks for Chronos Jobs.

$ shpkpr cron –help Usage: shpkpr cron [OPTIONS] COMMAND [ARGS]...

Manage Chronos Jobs.

Options:

--help	Show this message and exit.
--chronos_url	URL of the Chronos API to use.

Required Configuration

Chronos URL:

URL of the Chronos API to use, e.g. `http://chronos.mydomain.com:4400`

- Environment variable: `SHPKPR_CHRONOS_URL`
- Command-line flag: `--chronos_url`

Examples

```
$ shpkpr cron show --chronos_url chronos.mydomain.com:4400
[
    {
        "name": "job1",
        ...
    },
    {
        "name": "job2",
        ...
    }
]
```

```
$ shpkpr cron show --job-name job1 --chronos_url chronos.mydomain.com:4400
[
    {
        "name": "job1",
        ...
    }
]
```

```
$ shpkpr cron set --chronos_url chronos.mydomain.com:4400 \
  --template_dir /path/to/template/base/path \
  --template job_1_template.json.tmpl
```

```
$ shpkpr cron set --chronos_url chronos.mydomain.com:4400 \
  --template_dir /path/to/template/base/path \
  --template job_1_template.json.tmpl \
  --template job_2_template.json.tmpl
```

```
$ shpkpr cron delete-tasks --chronos_url chronos.mydomain.com:4400 job2
```

```
$ shpkpr cron delete --chronos_url chronos.mydomain.com:4400 job2
```

```
$ shpkpr cron run --chronos_url chronos.mydomain.com:4400 job2
```

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/shopkeep/shpkpr/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Did you install with `pip` or `docker`?

- Are you using a released version or a `git` checkout?
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

shpkpr could always use more documentation, whether as part of the official shpkpr docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/shopkeep/shpkpr/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up `shpkpr` for local development.

1. Fork the `shpkpr` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/shpkpr.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv shpkpr
$ cd shpkpr/
$ pip install --editable .
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the tests, including testing other Python versions with tox:

```
$ tox
```

   To get tox, just pip install it into your virtualenv.

   If you prefer, and have docker installed, you can also run the tests against all supported Python versions:

```
$ make test
```

   You should also run the integration tests against a live marathon instance to manually verify that the tool still works as expected. Set the necessary environment variables as appropriate:

```
# URL of the Marathon API to use
export SHPKPR_MARATHON_URL=http://marathon.somedomain.com:8080

# An application ID to use for testing, this should not exist on
# Marathon prior to running the tests
export SHPKPR_APPLICATION=my-dummy-application-for-testing

# Docker repotag of the container that should be deployed to Marathon.
# This container must be pullable by the mesos cluster; for testing
# purposes it's probably easiest to use a container from the public
# Docker hub.
export SHPKPR_DOCKER_REPOTAG=goexample/outyet:latest

# Port that should be exposed from the Docker container
export SHPKPR_DOCKER_EXPOSED_PORT=8080

# An arbitrary label to be injected into the deploy template. This can
# be any non-empty string for testing.
export SHPKPR_DEPLOY_DOMAIN=somedomain.com
```

   Then:

```
$ tox -e integration
```

   Or:

```
$ make test.integration
```

6. If your changes are user-facing, you should update the documentation to reflect the changes you've made. shpkpr's documentation is built with Sphinx and can be built using the `make`:

```
$ pip install Sphinx
$ make docs
```

   While developing, you can watch the documentation for changes and rebuild as required by installing watchdog:

```
$ pip install watchdog
$ make docs.watch
```

   The built documentation is output to the `_build/html/` folder. The simplest way to view these docs is with Python's built-in static webserver `python -m SimpleHTTPServer`.

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. If applicable, add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.3, 3.4, and 3.5, and for PyPy. Check https://travis-ci.org/shopkeep/shpkpr/pull_requests and make sure that the tests pass for all supported Python versions.