# Reactors SDK Documentation

***Release 1.0.0a***

**TACC Open Source**

**June 21, 2018**

# Quick Start


**1 Pre-requisites** **3**

**2 Start a Project** **5**
2.1 Let's go! . . . 5

**3 Configure the Project** **7**
3.1 Step 1: Edit config.rc . . . 7
3.2 Step 2: Edit config.yml . . . 7
3.3 Step 3: Create secrets.json . . . 7

**4 Write some code** **9**

**5 Deploy the Reactor** **11**

**6 Validate deployment** **13**

**7 Test by sending a message** **15**

**8 User Guide** **17**

**9 Third-party Webhooks** **19**

**10 Agave API Notifications** **21**

**11 Finite State Machine** **23**

**12 Schedule Actions** **25**

**13 Automate Deployment** **27**

**14 Unit Testing** **29**

**15 RabbitMQ** **31**

**16 AWS SNS** **33**

**17 Getting Help** **35**
17.1 TACC.cloud Slack . . . 35
17.2 Tenant-specific Assistance . . . 35

18 Other Tutorials and Guides 37

19 Use third-party Docker images 39

20 Import from other Serverless systems 41

21 Extend to other languages 43

22 Contribute to Reactors 45

23 Indices and tables 47

Nearly any language can be used to build functions for TACC's Abaco serverless computing platform. This is the documentation for a specific, opinionated approach that embeds a client-side Python2.7/3.6 support library called Reactors in the container that hosts an Abaco function.

Reactors extends the Abaco Actors concept with:

- YAML-based configuration mechanism with environment overrides
- Support for first-class mocking and local-side testing
- Semantic aliases for Actors and other TACC.cloud API assets
- Helper methods for working with integrations like Slack and IFTTT
- Introspection of the actor's platform-level attributes
- Advanced logging with support for redacting sensitive text
- Optimized TACC.cloud API operations

# CHAPTER 1

## Pre-requisites

1. You must have installed and configured the Agave CLI and the Abaco CLI.
2. You should have gone through the **Abaco Quickstart** and be familiar deploying, running, and querying an Actor

# CHAPTER 2

# Start a Project

**Concept:** The Abaco CLI will automatically generate a skeleton for your Reactor that is preconfigured for easy building, testing, and deployment. This same structure lends itself well to adopting continuous integration and unit testing if you should need those practices. The list of templates is limited at present, and constrained to Python language support. This is expected to change in the future. In addition, it is expected that deeper integration with Github and Gitlab will be added to the Reactors workflow.

## 2.1 Let's go!

Run `abaco init`, specifying language (`python2` or `python3` for now) and a URL-safe name. A good rule-of-thumb is to name a Reactors project like one would a Git repository.

```
$ abaco init -l python3 -n hello_world
$ ls hello_world/
Dockerfile       config.yml      reactor.py      requirements.txt
TEMPLATE.md      message.jsonschema  reactor.rc      secrets.json.sample
```

Details on what each project file does are provided in the User Guide.

# CHAPTER 3

# Configure the Project

**Concept:** The `Dockerfile` is a recipe to build the environment where our function will run. The function itself is implemented in `reactor.py`. A Python module built into the base Docker image (`reactors`) works with `config.yml` and `message.jsonschema` to provide declarative configuration and validation. The `requirements.txt` file is used with `pip` in the container image to specify additional Python modules to install. Finally, the Reactors workflow uses `reactor.rc` to specify name, version, and other metadata, and `secrets.json` as a way to pass sensitive information into a function without committing it to the container image.

## 3.1 Step 1: Edit config.rc

Naivigate to the project directory and edit `DOCKER_HUB_ORG` in `config.rc` to reflect either **your** Docker Hub username or an organization where you have push and pull access. For example, if a person with the Docker Hub username `taco` is a member of Docker Hub group `cabana`, they can choose either `taco` or `cabana` as the value for `DOCKER_HUB_ORG`

## 3.2 Step 2: Edit config.yml

Change the config file to read as follows.

```
---
logs:
  level: INFO
  token: ~
dont_reveal: ~
```

## 3.3 Step 3: Create secrets.json

Write a JSON file with the following contents.

```
{"_REACTORS_DONT_REVEAL": "This is a secret"}
```

# CHAPTER 4

## Write some code

**Concept:** An Abaco function is a script or binary that is set as the default command in a container, accepts a message and parameters from environment variables, and can (optionally) make use of a pre-authenticated Agave API client. Functions can be written in any language, but the Reactors Python SDK streamlines these processes and adds support for some experimental platform features.

**Action:** Replace the contents of `reactor.py`

```
from reactors.runtime import Reactor


def main():
    """Main function"""
    r = Reactor()
    r.logger.info("I received: {}".format(r.context['raw_message']))
    r.logger.debug("This is a DEBUG message from actor {}".format(r.uid))
    r.logger.info("This is an INFO message from actor {}".format(r.uid))
    r.logger.warning("This is a warning from actor {}".format(r.uid))

    r.logger.info("Here's that secret value: {}".format(
        r.settings.dont_reveal))

if __name__ == '__main__':
    main()
```

This example illustrates use of the `Reactor` object, specifically, its *settings*, *context*, and *logging* functions. More features and use cases are described in the User Guide and Scenarios sections.

# CHAPTER 5

## Deploy the Reactor

**Concept:** Functions can be deployed with the `abaco create` CLI command using a Docker image that has been built and pushed to a public registry. This is a very flexible approach, but it requires the authorone to execute the same series of steps each time. The `abaco deploy` command implements a streamlined workflow that, with configuration guidance from `reactor.rc`, automatically builds the image, pushes it, gathers environment variables, and deploys or updates the Reactor.

**Action:** Ensure the image builds correctly with a dry run

```
$ abaco deploy -R

 [INFO]   Build Options: --rm=true --pull
 Sending build context to Docker daemon  10.75kB
 Step 1/1 : FROM sd2e/reactors:python3
 python3: Pulling from sd2e/reactors
 Digest: sha256:789c9057306d618168193c75a6c47ca5c500bc6fcdb60dc30f27f9bf8b1af404
 Status: Image is up to date for sd2e/reactors:python3
 # Executing 5 build triggers
  ---> Using cache
  ---> Using cache
  ---> c06a54dcc66c
 Successfully built c06a54dcc66c
 Successfully tagged taco/hello_world:0.1
 [INFO] Stopping deployment as this was only a dry run!
```

**Action:** Deploy the Reactor

```
$ abaco deploy

 [INFO]   Build Options: --rm=true --pull
 Sending build context to Docker daemon  10.75kB
 Step 1/1 : FROM sd2e/reactors:python3
 python3: Pulling from sd2e/reactors
 Digest: sha256:789c9057306d618168193c75a6c47ca5c500bc6fcdb60dc30f27f9bf8b1af404
 Status: Image is up to date for sd2e/reactors:python3
```

(continues on next page)

(continued from previous page)

```
 # Executing 5 build triggers
  ---> Using cache
  ---> Using cache
  ---> Using cache
  ---> Using cache
  ---> Using cache
  ---> c06a54dcc66c
 Successfully built c06a54dcc66c
 Successfully tagged taco/hello_world:0.1
 The push refers to repository [docker.io/taco/hello_world]
 f9dde2603ec7: Pushed
 87f9719c8a1d: Mounted from sd2e/reactors
 913edbb0371b: Mounted from sd2e/reactors
 0.1: digest: sha256:a944131700e2ae540dc76f2c1c2d72e3909fdfd287b42a505c339ff79615bac7␣
↪size: 7184
 [INFO] Pausing to let Docker Hub register that the repo has been pushed
 [INFO] Reading environment variables from secrets.json
 Successfully deployed actor with ID: e6rkEBlzJ8vG4
```

# CHAPTER 6

## Validate deployment

**Concept:** A Reactor that has been deployed successfully will be accessible via the `actors` API and will report a status of **SUBMITTED** while the function is being deployed, then **READY** when it is prepared to accept messages.

**Action:** List the new actor by its identifier

```
$ abaco ls e6rkEBlzJ8vG4
```

The expected response should resemble this JSON document:


```
{
    "message": "Actor retrieved successfully.",
    "result": {
    "_links": {
      "executions": "https://api.sd2e.org/actors/v2/e6rkEBlzJ8vG4/executions",
      "owner": "https://api.sd2e.org/profiles/v2/taco",
      "self": "https://api.sd2e.org/actors/v2/e6rkEBlzJ8vG4"
    },
    "createTime": "2018-06-21 14:39:16.435800",
    "defaultEnvironment": {
      "_REACTORS_DONT_REVEAL": "This is a secret"
    },
    "description": "",
    "gid": 845005,
    "id": "e6rkEBlzJ8vG4",
    "image": "taco/hello_world:0.1",
    "lastUpdateTime": "2018-06-21 14:39:16.435800",
    "mounts": [
      {
        "container_path": "/work",
        "host_path": "/work",
        "mode": "rw"
      },
      {
        "container_path": "/corral",
```


(continues on next page)

(continued from previous page)


```
        "host_path": "/corral/projects/TACC-Cloud",
        "mode": "rw"
      }
    ],
    "name": "hello_world",
    "owner": "taco",
    "privileged": false,
    "state": {},
    "stateless": false,
    "status": "READY",
    "statusMessage": " ",
    "tasdir": "05201/taco",
    "type": "none",
    "uid": 845005,
    "useContainerUid": false
    },
    "status": "success",
    "version": "0.8.0"
}
```


Note that `result.status` is **READY** - this means the actor is ready to do work. If it reads **SUBMITTED**, deployment is stil in progress. If it reads **ERROR**, a problem has been encountered.

CHAPTER 7

# Test by sending a message

CHAPTER 8

# User Guide

CHAPTER 9

# Third-party Webhooks

CHAPTER 10

# Agave API Notifications

CHAPTER 11

# Finite State Machine

CHAPTER 12

# Schedule Actions

CHAPTER 13

# Automate Deployment

CHAPTER 14

# Unit Testing

CHAPTER 15

# RabbitMQ

CHAPTER 16

# AWS SNS

CHAPTER 17

# Getting Help

## 17.1 TACC.cloud Slack

You are welcome to join the developers and users of TACC.cloud services in TACC.cloud Slack. Helpful channels to join include `#support` and `#announcements`

## 17.2 Tenant-specific Assistance

If you are a user from any of the following organizations, you can get help from additional listed resources.

- CyVerse
- DesignSafe
- Synergistic Discovery and Design (SD2)

CHAPTER 18

# Other Tutorials and Guides

CHAPTER 19

# Use third-party Docker images

CHAPTER 20

# Import from other Serverless systems

CHAPTER 21

# Extend to other languages

CHAPTER 22

# Contribute to Reactors

CHAPTER 23

# Indices and tables

- genindex
- modindex
- search