genisys-connector-docker Documentation

Release 0.1.0

Anthony Lapenna

September 18, 2015

Contents

This component allows Genisys to communicate with a Docker engine.

CHAPTER 1

Contents

1.1 Setup

1.1.1 Docker

A public Docker image is available and can be used to start the component:

```
$ docker run -p "7051:7051" cyberdynesystems/genisys-connector-docker:latest
```

Do not forget to map the port 7051 of the container to a specific port on the Docker host.

Overriding the configuration

You can map your own configuration file in the container file system:

```
$ docker run -p "7051:7051" -v "/path/to/config/genisys-connector.yml:/app/genisys-connector.yml" cyb
```

1.1.2 From sources

Requirements

Ensure you have *python* >= 3.4 and *git* installed on your system.

Installation

Clone this repository and install the dependencies using **pip**:

```
$ git clone https://github.com/cyberdyne-corp/genisys-connector-docker && cd genisys-connector-docke
$ pip install -r requirements.txt
```

Start

Start the connector:

```
$ python main.py
```

1.2 Configuration

1.2.1 Configuration file

The configuration file *genisys-connector.yml* is written in YAML format.

```
connector:
  # Server address to bind to.
  bind: 127.0.0.1

  # Application port
  port: 7051

  # Path to compute definitions
  service_file: ./services.py

docker:
  # URL of the Docker server
  url: unix://var/run/docker.sock
```

Connector section

This section is related to the connector configuration.

bind

The server address to bind to.

port

The port that will be used to communicate with the connector via HTTP.

service_file

A python file that defines an optional list of services that will be loaded by the connector during startup.

See *Service definition* below for more information on the format of the file.

Docker section

This section is related to the Docker engine.

url

The URL of the Docker engine.

Can be either a path to the Docker engine socket or an URL to the Docker API.

1.2.2 Service definition

In order to manage Docker containers for a service, the adapter provides a simple service definition format to declare how to manage containers associated with a service.

A service definition looks like:

```
myService = {
    "name": "myService"
    "image": "docker_image:tag",
    "command": "command",
    "environment": {
            "ENV_VARIABLE_A": "value",
            "ENV_VARIABLE_B": "value",
    },
    "ports": ['8080']
}
```

A service definition must include a *name* and an *image*, it may optionally provide a *command*, an *environment* hash and an array of *ports*.

The *image* field is used to start a container, if the *command* field has been specified the container will be started using that command. The image tag **must** be specified.

The *image* field is also used when stopping containers, the image is used as reference to search in running containers.

The *environment* field is used to inject environment variables into the containers associated with the service.

The *ports* field is used to expose a list of ports on the Docker host.

An optional *service_file* (see *service_file*) can be used to define services using the format defined above. These definitions will be loaded during the connector startup.

1.3 HTTP API

The connector exposes a HTTP API. It can be used to perform CRUD actions on services and also to trigger remote procedure calls to manage containers.

NOTE: The examples use the httpie CLI to query the API.

1.3.1 Service HTTP endpoint

The following endpoints are exposed:

- */service*: List service definitions or register a new service definition
- */service/<service_name>*: Retrieve or update a service definition
- */service/<service_name>/scale*: Ensure a number of containers are running for a service
- */service/<service_name>/status*: Return the number of running resources for a service

`/service`

This endpoint is used to list service definitions or to create a new service definition.

It supports the following methods: POST and GET.

When hitting the endpoint with a GET, it returns a JSON body like this:

```
{
  "myServiceA" = {
        "name": "myServiceA"
        "image": "docker_image:tag",
        "command": "command",
        "environment": {
                "ENV_VARIABLE_A": "value",
                "ENV_VARIABLE_B": "value",
        },
        "ports": ['8080']
  },
  "myServiceB" = {
        "name": "myServiceB"
        "image": "docker_image:tag",
        "ports": ['5000', '5001'],
        "command": null,
        "environment": null,
  }
}
```

When hitting the endpoint with a POST, it expects a JSON request body that must look like:

```
{
  "name": "service_name",
  "image": "docker_image:tag",
  "command": "command",
  "environment": {
        "ENV_VARIABLE_A": "value",
        "ENV_VARIABLE_B": "value",
  },
  "ports": ['port_number']
}
```

The *name* and *image* fields are mandatory.

The *name* field is used to identify the service.

The *image* field specifies the reference of the container image used when creating/stopping containers. The image tag must be included.

The *command* field specifies which command should be used when starting a container.

The *environment* field is used to inject environment variables into the containers associated with the service.

The *ports* field is used to expose a list of ports on the Docker host.

Example:

```
$ http POST :7051/service name="helloworld" image="tutum/hello-world:latest" ports:='["8080", "8081"]
```

/service/<service_name>

This endpoint is used to retrieve a service definition or to update it.

It supports the following methods: PUT and GET.

When hitting the endpoint with a GET, it returns a JSON body like this:

```
{
  "image": "tutum/hello-world:latest",
```

```
  "name": "helloworld"
  "command": null,
  "environment": null,
  "ports": null,
}
```

When hitting the endpoint with a PUT, it expects a JSON request body that must look like:

```
{
  "image": "tutum/hello-world:latest",
  "command": "/run.sh",
  "environment": {
        "VAR_A": "value"
  },
  "ports": ['8080'],
}
```

The *image* field is mandatory.

The *image* field specifies the image to use when starting/killing containers. The image tag must be included.

The *command* field specifies which command should be used when starting a container.

The *environment* field is used to inject environment variables into the containers associated with the service.

The *ports* field is used to expose a list of ports on the Docker host.

Example:

```
$ http PUT :7051/service/helloworld image="panamax/hello-world-php:latest" command="/run.sh" ports:=
```

/service/<service_name>/scale

This endpoint is used to ensure that a specific number of containers associated to a service are running.

It expects a JSON request body to be POST.

The request body must look like:

```
{
  "number": number_of_containers,
}
```

The *number* field is mandatory.

Example:

```
$ http POST :7051/service/helloworld/scale number=3
```

/service/<service_name>/status

This endpoint returns the number of running resources for a service managed by this connector.

When hitting the endpoint with a GET, it returns a JSON body like this:

```
{
  "running_resources": number_of_running_resources,
}
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search