actingwebdemo Documentation

Release latest

Apr 22, 2021

Contents

This is a python and AWS Lambda/ElasticBeanstlak implementation showcasing the REST-based ActingWeb distributed micro-services model. It is both a demo application implementing the micro-services model as described in the specification, and it serves as the reference implementation for the ActingWeb REST protocol specification for how such micro-services interact.

CHAPTER 1

License

CHAPTER 2

Getting Started

This actingwebdemo application is a full ActingWeb demo that uses the python library actingweb.

The actingweb library currently supports AWS Dynamodb and this application is deployed to AWS as Lambda function and can be found at https://demo.actingweb.io. There are also config files for ElasticBeanstalk (see below).

Basically, there are only two files in this application: application.py uses the flask framework to map all the endpoints required by an ActingWeb app and set up handlers for each. This is done through a simplified request object that shows how a Flask request is mapped into request elements forward to the ActingWeb framework to handle the requests and through a handler object that processes the responses from the ActingWeb framework and maps back into Flask responses. It is very easy to replace Flask with any framework of your choice

The on_aw.py file implements the on_aw.OnAWBase() class. By overriding methods in this class, you can plug into the framework and do extra things as part of the requests. The empty class in this demo does nothing extra.

2.1 Extending the demo

You can modify all the templates in the templates directory. Most of the logic you can change can be found in on_aw.py. This file has empty methods for all relevant ActingWeb endpoints where you can adapt functionality.

2.2 Running locally/Docker

You don't have to deploy to AWS to test the app. There is a docker-compose.yml file in the repo that brings up both a local version of dynamodb and the actingwebdemo app. The docker image uses alpine as best practice to reduce the footprint of the container, as well as a production quality web server (uWSGI). Also note the use of pipenv as package manager. If you update Pipfile, make sure to run pipenv update‘

1. *docker-compose up -d*
2. Go to http://localhost:5000

You can also use ngrok.io or similar to expose the app on a public URL, remember to change the app URL in docker-compose.yml. See start-ngrok.sh for how to start up ngrok.

2.3 Running tests

If you use ngrok.io (or deploy to AWS), you can use the Runscope tests found in the tests directory. Just sign-up at runscope.com and import the test suites. The Basic test suite tests all actor creation and properties functionality, while the trust suite also tests trust relationships between actors, and the subscription suite tests subscriptions between actors with trust relationships. Thus, if basic test suite fails, all will fail, and if trust test suite fails, subscription test suite will also fail. The attributes test relies on the devtest endpoint to validate the internal attributes functionality to store attributes on an actor that are not exposed through properties or any other ActingWeb endpoint.

2.4 AWS Lambda

You can deploy the app to AWS Lamda in three simple steps. There is a serverless.yml file with the config you need.

1. Install Serverless
2. Edit serverless.yml APP_HOST_FQDN to use your domain (or AWS domain, see 4.) and region if you prefer another
3. Run *sls deploy*
4. (if using AWS allocated domain) Use the long domain name AWS assigns the lambda and go to #2 above

2.5 AWS Elastic Beanstalk

You can also deploy to Elastic Beanstalk:

0. Delete the config.yml in .elasticbeanstalk (it's just for your reference)
1. Install Elastic Beanstalk CLI

2. Edit .ebextensions/options.config to set the hostname you are going to deploy to, region that matches, and the protocol (use http:// if you don't want to set up a certificate just for testing)

2. Run *eb init*, set region and AWS credentials, create a new app (your new app), and select Docker, latest version

3. Run *eb create* to create an environment (e.g. dev, prod etc of your app). Remember to match the CNAME prefix with the prefix of the hostname in options.config (the rest is based on region)

4. Deploy with *eb deploy*
5. Run *eb open* to open the app in the browser

2.6 Use the library for your own projects

For how to use and extend the library, see the ActingWeb repository

CHAPTER 3

CHANGELOG

3.1 Aug 22, 2020

- Put on_aw.py in src/
- Add code to show how the app can authenticate against Google to access Google services
- Fix issue where 403 Forbidden was not correctly returned
- Upgrade serverless dependencies
- Upgrade to python3.7
- Added warning on AWS API Gateway not allowing login on www
- Improve templates

3.2 Nov 21, 2018

- Update to use actingweb v2.5.0

3.3 Oct 22, 2018

- Updated to use python3, lambda, and dynamodb
- Deployed to https://demo.actingweb.io

3.4 Nov 12, 2017

- Forked as separate demo app from actingweb library

- Deployed to https://actingwebdemo.greger.io as a test app