Mocking AWS

FOR DEVELOPMENT PURPOSES (+ MORE)

AWS is great, but…

- Caters for deployment, not development
- No out-of-the box namespacing/isolation
- Can lead to explosion of artifacts, eg. S3 paths, DynamoDB tables, ElastiCache
  - AWS storage costs
  - Bandwidth costs
  - Devops time
- IMO: requires tools and best practices

Best practices used in DBs

- Schema creation scripts
- Data population scripts
- For dev, local deployments of DB instance

…plus others: versioning, archiving, etc

What is fake-aws?

https://github.com/konrads/fake-aws

AWS API compatible mocks for storage services: S3, DynamoDB, ElasticCache (Redis)

Implemented in bash and python.

Includes:

- Schema/data templating
- Schema creation/data population (to both local and remote)
- Local (mocked) Docker containers for S3, DynamoDB, ElasticCache
- Additional Client Docker, containing all the tools required for fake-aws' interaction

Demo – one-off setup

```
# install docker, python 2.7, fake-aws. Optional: jq, redis-cli
# build fake DynamoDB, S3, ElasticCache
fake-aws images build
# setup for local deployment
export dynamodb_host=localhost s3_host=localhost redis_host=localhost
```

Demo – create and populate test data

```
data-gen pcetest resources/aws-templates resources/aws-stage

fake-aws refresh containers resources/aws-stage

# list contents of remote and local

aws dynamodb list-tables

aws dynamodb list-tables --endpoint-url http://$dynamodb_host:7777

aws s3 ls

aws s3 ls  --endpoint-url http://$s3_host:3629
```

Demo – run tests against

```
# no network required
sbt "testOnly com.onzo.pceapi.metrics.*"
sbt "testOnly com.onzo.pceapi.controller.ApplianceCtrlSpec"
sbt "testOnly com.onzo.pceapi.controller.ComparisonSpec"
```

Demo – remote data population

```
# needs network
unset dynamodb_host s3_host
export redis_host=localhost
aws-populate refresh all resources/aws-stage
```