seddy

Release 0.1.0rc0

Laurie O

Mar 24, 2020

CONTENTS

Multi-workflow SWF decider and workflow management service.

CHAPTER

ONE

INSTALLATION

```
pip3 install seddy
```

CHAPTER

TWO

DOCUMENTATION

2.1 seddy

2.1.1 seddy.decisions

SWF decisions building.

class seddy.decisions.**DecisionsBuilder**(*workflow: seddy.decisions._base.Workflow, task: Dict[str, Any]*)

Bases: object

SWF decision builder.

Parameters

- **workflow** – workflow specification
- **task** – decision task

abstract build_decisions()

Build decisions from workflow history.

class seddy.decisions.**Workflow**(*name: str*, *version: str*, *description: str = None*)

Bases: object

SWF workflow specification.

Parameters

- **name** – workflow name
- **version** – workflow version

abstract property decisions_builder

classmethod from_spec(*spec: Dict[str, Any]*)

Construct workflow type from specification.

Parameters **spec** – workflow specification

make_decisions(*task: Dict[str, Any]*) → List[Dict[str, Any]]

Build decisions from workflow history.

Parameters **task** – decision task

Returns workflow decisions

setup()

Set up workflow specification.

Useful for pre-calculation or other initialisation.

abstract property spec_type

class seddy.decisions.**DAGBuilder**(*workflow: seddy.decisions._dag.DAGWorkflow*, *task*)
Bases: seddy.decisions._base.DecisionsBuilder

SWF decision builder from DAG-type workflow specification.

build_decisions()
Build decisions from workflow history.

class seddy.decisions.**DAGWorkflow**(*name*, *version*, *task_specs: List[Dict[str, Any]]*, *description=None*)
Bases: seddy.decisions._base.Workflow

Dag-type SWF workflow specification.

Parameters

- **name** – workflow name
- **version** – workflow version
- **task_specs** – DAG task specifications

decisions_builder
alias of *DAGBuilder*

classmethod from_spec(*spec*)
Construct workflow type from specification.

Parameters **spec** – workflow specification

setup()
Set up workflow specification.

Useful for pre-calculation or other initialisation.

spec_type = 'dag'

2.1.2 seddy.decider

SWF decider.

class seddy.decider.**Decider**(*workflows: List[seddy.decisions._base.Workflow]*, *domain: str*, *task_list: str*)
Bases: object

SWF decider.

Parameters

- **workflows** – decider workflows
- **domain** – SWF domain to poll in
- **task_list** – SWF decider task-list

client
SWF client

Type botocore.client.BaseClient

identity
name of decider to poll as

Type str

run()
Run decider.

seddy.decider.**run_app**(*workflows_spec_json: pathlib.Path*, *domain: str*, *task_list: str*)
Run decider application.

Parameters

- **workflows_spec_json** – workflows specifications JSON
- **domain** – SWF domain
- **task_list** – SWF decider task-list

2.1.3 seddy.registration

SWF workflow registration.

seddy.registration.**list_workflows**(*domain: str*, *client*) → List[Tuple[str, str]]
List all workflows in SWF, including registered and deprecated.

Parameters

- **domain** – domain to list workflows of
- **client** (*botocore.client.BaseClient*) – SWF client

Returns names and versions of workflows in SWF

seddy.registration.**register_workflow**(*workflow: seddy.decisions._base.Workflow*, *domain: str*, *client*)
Register a workflow with SWF.

Parameters

- **workflow** – specification of workflow to register
- **domain** – domain to register workflow in
- **client** (*botocore.client.BaseClient*) – SWF client

seddy.registration.**register_workflows**(*workflows: List[seddy.decisions._base.Workflow]*, *domain: str*, *skip_existing: bool = False*)
Register workflows with SWF.

Parameters

- **workflows** – specifications of workflows to register
- **domain** – domain to register workflows in
- **skip_existing** – check for and skip existing workflows

seddy.registration.**run_app**(*workflows_spec_json: pathlib.Path*, *domain: str*, *skip_existing: bool = False*)
Run decider application.

Parameters

- **workflows_spec_json** – workflows specifications JSON
- **domain** – SWF domain
- **skip_existing** – check for and skip existing workflows

Multi-workflow SWF decider and workflow management service.

2.2 Command-line application

seddy provides a command-line interface for the as-built production service. The interface documentation can be accessed with:

```
seddy -h
```

2.3 SWF decider tutorial

Running an SWF decider for a virtual AWS.

We'll use moto, a tool which mocks out SWF.

Warning: `moto` v1.13.4 doesn't correctly support SWF. In particular:

- Task-polling returns instantly
- No-task result from task-polling is missing `taskToken`, so `seddy decider` will crash whenever there is no result
- Decision tasks have incorrect value for `previousStartedEventId`, so `seddy decider` will crash after the two decision tasks

These are not issues when using `seddy` for real

2.3.1 Set-up

Install `moto`, `awscli` and `seddy`

```
pip install moto[server] awscli seddy
```

Environment variables

To use `moto`, we need to point the AWS CLI and `seddy` to its server (which we'll start below)

```
export AWS_DEFAULT_REGION=us-east-1
export AWS_SWF_ENDPOINT_URL=http://localhost:5042/
```

2.3.2 Example

Create workflow definitions file

```
{
    "version": "1.0",
    "workflows": [
        {
            "spec_type": "dag",
            "name": "spam",
            "version": "1.0",
            "description": "A workflow with spam, spam, eggs and spam.",
            "registration_defaults": {
                "task_timeout": 5,
                "execution_timeout": 3600,
                "task_list": "coffee"
            },
            "tasks": [
                {
                    "id": "foo",
                    "type": {
                        "name": "spam-foo",
                        "version": "0.3"
                    },
                    "timeout": 10,
                    "task_list": "eggs",
                    "priority": 1
                },
                {
                    "id": "bar",
                    "type": {
                        "name": "spam-foo",
                        "version": "0.4"
                    },
                    "timeout": 10,
                    "task_list": "eggs",
                    "dependencies": ["foo"]
                }
            ]
        },
        {
            "spec_type": "dag",
            "name": "spam",
            "version": "1.1",
            "description": "A workflow with better spam, spam, eggs and spam.",
            "registration_defaults": {
                "task_timeout": 5,
                "execution_timeout": 3600,
                "task_list": "coffee"
            },
            "tasks": [
                {
                    "id": "foo",
                    "type": {
                        "name": "spam-foo",
                        "version": "0.4"
                    },
                    "timeout": 5,
```

(continues on next page)

(continued from previous page)

```
                    "task_list": "eggs",
                    "priority": 1
                },
                {
                    "id": "bar",
                    "type": {
                        "name": "spam-foo",
                        "version": "0.4"
                    },
                    "timeout": 5,
                    "task_list": "eggs",
                    "dependencies": ["foo"]
                }
            ]
        }
    ]
}
```

Start the mock SWF server (in a separate terminal: don't forget *Environment variables*)

```
moto_server swf -p5042
```

Register domain

```
aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf register-domain \
  --name test-domain --workflow-execution-retention-period-in-days 2
```

Register defined workflows with SWF

```
seddy -v register workflows.json test-domain
```

Register referenced activities with SWF

```
aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf register-activity-type \
  --domain test-domain \
  --name spam-foo \
  --activity-version 0.3 \
  --default-task-start-to-close-timeout 20 \
  --default-task-schedule-to-start-timeout 600 \
  --default-task-schedule-to-close-timeout 620 \
  --default-task-heartbeat-timeout 20 \
  --default-task-list name=test-activity-list

aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf register-activity-type \
  --domain test-domain \
  --name spam-foo \
  --activity-version 0.4 \
  --default-task-start-to-close-timeout 20 \
  --default-task-schedule-to-start-timeout 600 \
  --default-task-schedule-to-close-timeout 620 \
  --default-task-heartbeat-timeout 20 \
  --default-task-list name=test-activity-list
```

Start the decider (in a separate terminal: don't forget *Environment variables*)

```
seddy -v decider workflows.json test-domain test-list
```

Start a workflow execution

```
aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf start-workflow-execution \
  --domain test-domain \
  --workflow-id test-wf \
  --workflow-type name=spam,version=1.1 \
  --task-list name=test-list \
  --child-policy ABANDON \
  | python3 -c 'import sys, json; print(json.load(sys.stdin)["runId"])' \
  > /tmp/runid
```

Pretend to be an activity worker

```
aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf poll-for-activity-task \
  --domain test-domain --task-list name=eggs \
  | python3 -c 'import sys, json; print(json.load(sys.stdin)["taskToken"])' \
  > /tmp/tasktoken
aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf respond-activity-task-completed \
  --task-token $(cat /tmp/tasktoken)

aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf poll-for-activity-task \
  --domain test-domain --task-list name=eggs \
  | python3 -c 'import sys, json; print(json.load(sys.stdin)["taskToken"])' \
  > /tmp/tasktoken
aws --endpoint-url $AWS_SWF_ENDPOINT_URL swf respond-activity-task-completed \
  --task-token $(cat /tmp/tasktoken)
```

Check execution status

```
aws --endpoint-url $AWS_SWF_ENDPOINT_URL describe-workflow-execution \
  --domain test-domain --execution workflowId=test-wf,runId=$(cat /tmp/runid)
```

- genindex

PYTHON MODULE INDEX

S

INDEX