seddy

Release 0.2.0a0

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.decider

SWF decider.

Classes

Decider(workflows_spec_file, domain, ...)	SWF decider.

Exceptions

UnsupportedWorkflow	Decider doesn't support workflow.

Functions

run_app(workflows_spec_file, domain, ...)	Run decider application.

class seddy.decider.**Decider**(*workflows_spec_file: pathlib.Path*, *domain: str*, *task_list: str*, *identity: str = None*)

Bases: object

SWF decider.

Parameters

- **workflows_spec_file** – workflows specifications file path
- **domain** – SWF domain to poll in
- **task_list** – SWF decider task-list
- **identity** – decider identity, default: automatically generated from fully-qualified domain-name and a UUID

Methods

run()	Run decider.

client
SWF client

Type botocore.client.BaseClient

identity
name of decider to poll as

Type str

run()
Run decider.

exception seddy.decider.**UnsupportedWorkflow**
Bases: LookupError

Decider doesn't support workflow.

seddy.decider.**run_app**(*workflows_spec_file: pathlib.Path*, *domain: str*, *task_list: str*, *identity: str = None*)
Run decider application.

Parameters

- **workflows_spec_file** – workflows specifications file path
- **domain** – SWF domain
- **task_list** – SWF decider task-list
- **identity** – decider identity, default: automatically generated

2.1.2 seddy.registration

SWF workflow registration.

Functions

deprecate_workflow(workflow, domain, client)	Deprecate a workflow in SWF.
list_workflows(domain, client)	List all workflows in SWF, including registered and deprecated.
register_workflow(workflow, domain, client)	Register a workflow with SWF.
register_workflows(workflows, domain)	Synchronise workflow registration with SWF.
run_app(workflows_spec_file, domain)	Run registration synchronisation application.
undeprecate_workflow(workflow, domain, client)	Undeprecate a workflow in SWF.

seddy.registration.**deprecate_workflow**(*workflow: seddy._specs._base.Workflow*, *domain: str*, *client*)
Deprecate a workflow in SWF.

Parameters

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

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

Parameters

- **domain** – domain to list workflows of

- **client** (*botocore.client.BaseClient*) – SWF client

Returns names, versions and registration status of workflows in SWF

seddy.registration.**register_workflow**(*workflow: seddy._specs._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._specs._base.Workflow]*, *domain: str*)

Synchronise workflow registration with SWF.

Parameters

- **workflows** – specifications of workflows to register
- **domain** – domain to register workflows in

seddy.registration.**run_app**(*workflows_spec_file: pathlib.Path*, *domain: str*)

Run registration synchronisation application.

Parameters

- **workflows_spec_file** – workflows specifications file path
- **domain** – SWF domain

seddy.registration.**undeprecate_workflow**(*workflow: seddy._specs._base.Workflow*, *domain: str*, *client*)

Undeprecate a workflow in SWF.

Parameters

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

Multi-workflow SWF decider and workflow management service.

Classes

ChildPolicy	Policy for child executions on parent termination.
Registration(active, task_timeout, ...)	Workflow registration configuration.
DecisionsBuilder(workflow, task, Any])	SWF decision builder.
Workflow(name, version, description, ...)	SWF workflow specification.
DAGBuilder(workflow, task)	SWF decision builder from DAG-type workflow specification.
DAGWorkflow(name, version, task_specs, Any]])	Dag-type SWF workflow specification.

Functions

load_workflows(workflows_file)	Load workflows specifications file.

class seddy.**ChildPolicy**

Bases: enum.Enum

Policy for child executions on parent termination.

See also:

StartWorkflowExecution in SWF API documentation

Attributes

ABANDON	str(object='') -> str
REQUEST_CANCEL	str(object='') -> str
TERMINATE	str(object='') -> str

ABANDON = 'ABANDON'

REQUEST_CANCEL = 'REQUEST_CANCEL'

TERMINATE = 'TERMINATE'

class seddy.**Registration**(*active: bool = True*, *task_timeout: Union[int, str] = None*, *execution_timeout: int = None*, *task_list: str = None*, *task_priority: int = None*, *child_policy: seddy._specs._base.ChildPolicy = None*, *lambda_role: str = None*)

Bases: object

Workflow registration configuration.

Parameters

- **active** – registration status, False for deprecated
- **task_timeout** – default decision task time-out (seconds), or "NONE" for unlimited
- **execution_timeout** – default workflow execution time-out (seconds)
- **task_list** – default decision task-list
- **task_priority** – default decision task priority
- **child_policy** – default policy for child executions upon parent execution termination
- **lambda_role** – default IAM role for Lambda access

Methods

from_spec(spec, Any])	Construct registration configuration from specification.

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

Construct registration configuration from specification.

Parameters **spec** – workflow registration configuration specification

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

Bases: object

SWF decision builder.

Parameters

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

Methods

build_decisions()	Build decisions from workflow history.

abstract build_decisions()
Build decisions from workflow history.

class seddy.**Workflow**(*name: str, version: str, description: str = None, registration: seddy._specs._base.Registration = None*)
Bases: object

SWF workflow specification.

Parameters

- **name** – workflow name
- **version** – workflow version
- **registration** – workflow registration configuration

Attributes

decisions_builder
spec_type

Methods

from_spec(spec, Any])	Construct workflow type from specification.
make_decisions(task, Any])	Build decisions from workflow history.
setup()	Set up workflow specification.

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

SWF decision builder from DAG-type workflow specification. **Methods**

build_decisions()	Build decisions from workflow history.

build_decisions()
Build decisions from workflow history.

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

Dag-type SWF workflow specification.

Parameters

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

Classes

decisions_builder	SWF decision builder from DAG-type workflow specification.

Methods

setup()	Set up workflow specification.

Attributes

spec_type	str(object='') -> str

decisions_builder
alias of *DAGBuilder*

setup()
Set up workflow specification.

Useful for pre-calculation or other initialisation.

spec_type = 'dag'

seddy.**load_workflows**(*workflows_file: pathlib.Path*) → List[seddy._specs._base.Workflow]
Load workflows specifications file.

Determines load method from the file suffix. Supported file types:

- JSON
- YAML

Parameters **workflows_file** – workflows specifications file path

Returns workflows specifications

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
```

2.2.1 Docker

Instead of installing *seddy* locally, you can use our pre-built Docker image

```
docker run -v /path/to/workflow/file/parent:/seddy-data seddy -h
```

2.3 Data exchange in executions

See also:

Data exchange SWF documentation

seddy assumes all workflow input and output is JSON-serialisable, and will be manipulated as such according to task IDs. The task ID is used as key for the task input from the workflow input, and the task ID is used as the key to place the task's result input the workflow result.

For example, a workflow with task IDs "task1", "task2", "task3" and "task4" could have execution input:

```
{"task1": "spam", "task2": {"a": 42, "b": null}, "task4": null}
```

And execution result:

```
{"task1": "eggs", "task3": null, "task4": {"c": [1, 2]}}
```

Note that a task won't recieve input if it's not provided, and a task won't have a corresponding entry in the workflow result if the task doesn't provide a result.

To get an arbitrary string as input or result, simply provide a JSON string, eg `"foo: bar"`.

2.4 Workflows specifications

Workflows specified in a workflows specs file can have different specification types. The supported types follow:

2.4.1 DAG-type workflow specification

A DAG (directed acyclic graph) workflow is a series of tasks that are scheduled to run after their dependencies have finished.

Specification

A DAG-type workflow (element of `workflows`) has specification

- **spec_type** (*string*): specification type, must be `dag`
- **name**, **version**, **description** and **registration**: see *Common specification*
- **tasks** (*array[object]*): array of workflow activity tasks to be run during execution, see ScheduleActivityTaskDecisionAttributes
  - **id** (*string*): task ID, must be unique within a workflow execution and without `:`, `/`, `|`, `arn` or any control character
  - **type**: activity type, with **name** (*str*, activity name) and **version** (*str*, activity version)
  - **heartbeat** (*int or "NONE"*): optional, task heartbeat time-out (seconds), or `"NONE"` for unlimited
  - **timeout** (*int*): optional, task time-out (seconds), or `"None"` for unlimited
  - **task_list** (*string*): optional, task-list to schedule task on
  - **priority** (*int*): optional, task priority
  - **dependencies** (*array[string]*): optional, IDs of task's dependents

Example

```
spec_type: dag
name: spam
version: "1.0"
description: A workflow with spam, spam, eggs and spam.
registration:
  active: true
  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
```

2.4.2 Specification

The workflow file has structure

- **version** (*string*): workflow specifications file version
- **workflows** (*array*): workflows' specifications

2.4.3 Common specification

A workflow (element of `workflows`) has common specification

- **spec_type** (*string*): specification type
- **name** (*string*): workflow name
- **version** (*string*): workflow version
- **description** (*string*): optional, workflow description
- **registration** (*object*): optional, specifies workflow registration status default configuration, see RegisterWorkflowType
  - **active** (*boolean*): optional (default: true), intended workflow registration status (mark as false to deprecate a workflow)
  - **task_timeout** (*int or "NONE"*): optional, default decision task time-out (seconds), `"NONE"` for unlimited
  - **execution_timeout** (*int*): optional, default workflow execution time-out (seconds)
  - **task_list** (*string*): optional, default decision task-list, see task lists
  - **task_priority** (*int*): optional, default decision task-list, see setting task priority
  - **child_policy** (*string*): optional, default decision task-list, see child workflows
  - **lambda_role** (*string*): optional, default IAM role for Lambda access, see using Lambda tasks

Example

```
spec_type: test
name: spam
version: "1.0"
description: A workflow with spam, spam, eggs and spam.
registration:
  active: true
  task_timeout: 5
  execution_timeout: 3600
  task_list: coffee
  task_priority: 2
  child_policy: TERMINATE
  lambda_role: arn:aws:iam::spam:role/eggs
```

2.5 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

2.5.1 Set-up

Install Moto, AWS CLI, PyYaml and seddy

```
pip install moto[server] awscli pyyaml 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/
```

2.5.2 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:
      active: true
      task_timeout: 5
      execution_timeout: 3600
      task_list: coffee
    tasks:
      - id: foo
        type:
          name: spam-foo
          version: "0.3"
        timeout: 10
```

(continues on next page)

(continued from previous page)

```
        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:
      active: true
      task_timeout: 5
      execution_timeout: 3600
      task_list: coffee
    tasks:
      - id: foo
        type:
          name: spam-foo
          version: "0.4"
        timeout: 5
        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.yml 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.yml 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