Flowser Documentation

Release 0.1.1

Simon Pantzare

October 08, 2012

CONTENTS

Flowser is a high-level interface for Amazon Simple Workflow on top of Boto.

CHAPTER

ONE

CONTENTS

1.1 Arithmetic Example

1.1.1 Utilities

Ids for workflows and activities are created behind the scenes by inspecting inputs. `get_id_from_input` beneath assumes that the input has an "id" key and appends this value to the class name of the workflow or activity:

```
import flowser

@classmethod
def get_id_from_input(cls, input):
    return ".".join([cls.name, input['id']])
```

This function sets version and task list properties on decorated classes. It also makes classes use our `get_id_from_input` function defined earlier:

```
def auto_configured(cls):
    """Class decorator for workflows and activities. """
    cls.version = '1.0.0'
    cls.task_list = '-'.join([cls.name, cls.version])
    cls.get_id_from_input = get_id_from_input
    return cls
```

1.1.2 Domain, Workflow and Activities

Definition of a workflow that performs arithmetic:

```
@auto_configured
class ArithmeticWorkflow(flowser.types.Workflow):
    """Workflow for performing arithmetics. """
    name = 'ArithmeticWorkflow'
    execution_start_to_close_timeout = '600'
    task_start_to_close_timeout = '120'
    child_policy = 'TERMINATE'
```

Definition of two activities. These will be scheduled from arithmetic workflows:

```
@auto_configured
class MultiplyActivity(flowser.types.Activity):
    name = 'MultiplyActivity'
    heartbeat_timeout = '60'
```

```
    schedule_to_close_timeout = '60'
    schedule_to_start_timeout = '60'
    start_to_close_timeout = '60'

@auto_configured
class SumActivity(flowser.types.Activity):
    name = 'SumActivity'
    heartbeat_timeout = '60'
    schedule_to_close_timeout = '60'
    schedule_to_start_timeout = '60'
    start_to_close_timeout = '60'
```

Definition of a math domain:

```
class MathDomain(flowser.Domain):
    name = 'math'
    workflow_types = [ArithmeticWorkflow]
    activity_types = [MultiplyActivity, SumActivity]
```

Register domain, workflows and activities:

```
import boto
domain = MathDomain(boto.connect_swf())
domain.register()
```

Implementations of worker threads for our workflow and activities:

```
import threading

class ArithmeticWorkflowDecider(threading.Thread):

    def run(self):
        op_to_activity = {
                'multiply': MultiplyActivity,
                'sum': SumActivity,
                }
        for task in domain.decisions(ArithmeticWorkflow):
            is_new = len(task.filter('DecisionTaskScheduled')) == 1
            if is_new:
                # Schedule tasks from "operations" input.
                for op_id, op, input in task.start_input['operations']:
                    activity = op_to_activity[op]
                    task.schedule(activity, {
                        'id': task.start_input['id'],
                        'operation': [op_id, input],
                        })
                task.complete()
            else:
                op_ids = set([x[0] for x in task.start_input['operations']])
                results = {}
                for ev in task.filter('ActivityTaskCompleted'):
                    result = ev.attrs['result']
                    results[result[0]] = result[1]
                result_ids = set(results.keys())

                got_all = op_ids == result_ids
                if got_all:
                    # Got results for all operations. Complete workflow.
                    task.workflow_execution.complete(results)
```

```
                else:
                    task.complete()


class WorkerThread(threading.Thread):

    def run(self):
        for task in domain.activities(self.activity_class):
            self.handle_task(task)


class MultiplyWorker(WorkerThread):

    activity_class = MultiplyActivity

    def handle_task(self, task):
        op = task.input['operation']
        result = reduce(lambda a, b: a * b, op[1])
        task.complete([op[0], result])


class SumWorker(WorkerThread):

    activity_class = SumActivity

    def handle_task(self, task):
        op = task.input['operation']
        task.complete([op[0], sum(op[1])])
```

1.1.3 Starting a Workflow Execution

Start workers and an execution:

```
MultiplyWorker().start()
SumWorker().start()
ArithmeticWorkflowDecider().start()

import uuid
arithmetic_input = {
    'id': str(uuid.uuid4()),
    'operations': [
        ['mult_id', 'multiply', [1, 2, 3]],
        ['sum_id', 'sum', [1, 2, 3, 4]],
        ],
    }
domain.start(ArithmeticWorkflow, arithmetic_input)
```

1.2 API Reference

1.2.1 flowser.domain

class `flowser.domain.`**`Domain`**(*conn*)
Represents a Simple Workflow domain.

Subclasses must set a `name` property. They may also set a `retention_period` property (defaults to '30').

To register types, `workflow_types` and `activity_types` need to be set. They should be lists of `types.Workflow` and `types.Activity` subclasses.

activities(*t*)
High-level interface to iterate over activity tasks.

This method polls for new tasks of the given type indefinitely.

Parameters **t** – Subclass of `types.Type`.

activity_types = None

decisions(*t*)
High-level interface to iterate over decision tasks.

This method polls for new tasks of the given type indefinitely.

Parameters **t** – Subclass of `types.Type`.

register(*raise_exists=False*)
Register domain and associated types on AWS.

retention_period = '30'

start(*t*, *input*)
Start execution.

Internally, this method creates an instance of `t` and calls its `start` method with the given input.

Parameters **t** – Subclass of `types.Type`.

workflow_types = None

1.2.2 flowser.types

class `flowser.types.`**Activity**(*domain*)
Base class for activity types.

Subclasses must set `name`, `task_list` and `version` properties and implement a `schedule` class method.

heartbeat_timeout = '3600'

classmethod schedule(*input*, *control=None*)
Called from subclasses' `schedule` class method.

schedule_to_close_timeout = '3600'

schedule_to_start_timeout = '3600'

start_to_close_timeout = '3600'

class `flowser.types.`**Type**(*domain*)
Base class for Simple Workflow types (activities, workflows).

Subclasses must set `name`, `version` and `task_list` properties. They must also implement `get_id_from_input`.

static get_id_from_input(*input*)
Get id from input.

This class method is used to get unique workflow and activity ids.

A typical implementation may retrieve and id field from the input and append it to the class's task list.

Parameters **input** – Input used when starting and scheduling workflows and tasks.

class `flowser.types.`**`Workflow`**(*domain*)
Base class for workflow types.

Subclasses must set `name` and `task_list` properties and implement a `start` method and a `start_child` class method.

`child_policy` = **'TERMINATE'**

`default_filter_tag` = **None**

`default_tag_list` = **None**

`execution_start_to_close_timeout` = **'600'**

classmethod `start_child`(*input*, *control=None*)
Start child workflow execution.

`input` is serialized and a workflow id is generated from it using `get_id_from_input`.

`task_start_to_close_timeout` = **'120'**

1.2.3 flowser.tasks

class `flowser.tasks.`**`Activity`**(*result*, *caller*)
Wrapper for "PollForActivityTask" results.

See http://docs.amazonwebservices.com/amazonswf/latest/apireference/API_PollForActivityTask.html.

`cancel`(*details=None*)

`complete`(*result=None*)

`fail`(*details=None*, *reason=None*)

class `flowser.tasks.`**`ActivityType`**(*result*)
Wrapper for the API data type.

See http://docs.amazonwebservices.com/amazonswf/latest/apireference/API_ActivityType.html.

class `flowser.tasks.`**`Decision`**(*result*, *caller*)
Wrapper for "PollForDecisionTask" results.

See http://docs.amazonwebservices.com/amazonswf/latest/apireference/API_PollForDecisionTask.html.

This class assumes that history events are in reverse order (most recent first).

`complete`(*context=None*)

`events`

`fail`(*details=None*, *reason=None*)

`filter`(*event_type*)

`mark`(*name*, *details=None*)
Adds a RecordMarker decision.

`most_recent`(*event_type*)

`schedule`(*activity_type*, **args*, ***kwargs*)
Schedule activity.

Internally, this method calls the schedule classmethod on the activity type with the given args and kwargs.

Parameters **activity_type** – Subclass of `types.Activity`.

start_child(*workflow_type*, **args*, ***kwargs*)
Start child workflow.

Internally, this method calls the start_child classmethod on the workflow type with the given args and kwargs.

Parameters **workflow_type** – Subclass of `types.Workflow`.

start_input
Get start input as a python object.

This method iterates over the event history to find the WorkflowExecutionStarted event and unserializes its input attribute. The result is cached.

class `flowser.tasks.`**WorkflowExecution**(*result*, *caller*)
Wrapper for the API data type.

See http://docs.amazonwebservices.com/amazonswf/latest/apireference/API_WorkflowExecution.html.

abandon(*details=None*, *reason=None*)

complete(*result*, *context=None*)
Complete workflow execution.

This can only be called from a decision task.

request_cancel()

signal(*name*, *input=None*)

terminate(*details=None*, *reason=None*)

terminate_request_cancel(*details=None*, *reason=None*)

class `flowser.tasks.`**WorkflowType**(*result*)
Wrapper for the API data type.

See http://docs.amazonwebservices.com/amazonswf/latest/apireference/API_WorkflowType.html.

1.2.4 flowser.exceptions

exception `flowser.exceptions.`**EmptyTaskPollResult**

exception `flowser.exceptions.`**Error**

exception `flowser.exceptions.`**LastPage**

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

f