Domovoi Documentation

Release 0.0.1

Andrey Kislyuk

Jul 05, 2018

Contents

Domovoi is an extension to AWS Chalice to handle AWS Lambda event sources other than HTTP requests through API Gateway. Domovoi lets you easily configure and deploy a Lambda function to run on a schedule or in response to a variety of events like an SNS or SQS message, S3 event, or custom state machine transition:

```
import json, boto3, domovoi

app = domovoi.Domovoi()

@app.scheduled_function("cron(0 18 ? * MON-FRI *)")
def foo(event, context):
    context.log("foo invoked at 06:00pm (UTC) every Mon-Fri")
    return dict(result=True)

@app.scheduled_function("rate(1 minute)")
def bar(event, context):
    context.log("bar invoked once a minute")
    boto3.resource("sns").create_topic(Name="bartender").publish(Message=json.dumps({
↪"beer": 1}))
    return dict(result="Work work work")

@app.sns_topic_subscriber("bartender")
def tend(event, context):
    message = json.loads(event["Records"][0]["Sns"]["Message"])
    context.log(dict(beer="Quadrupel", quantity=message["beer"]))

@app.sqs_queue_subscriber("my_queue", batch_size=64)
def process_queue_messages(event, context):
    message = json.loads(event["Records"][0]["body"])
    message_attributes = event["Records"][0]["messageAttributes"]
    # SQS messages are deleted upon successful exit, requeued otherwise.
    # See https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html

@app.cloudwatch_event_handler(source=["aws.ecs"])
def monitor_ecs_events(event, context):
    message = json.loads(event["Records"][0]["Sns"]["Message"])
    context.log("Got an event from ECS: {}".format(message))

@app.s3_event_handler(bucket="myS3bucket", events=["s3:ObjectCreated:*"], prefix="foo
↪", suffix=".bar")
def monitor_s3(event, context):
    message = json.loads(event["Records"][0]["Sns"]["Message"])
    context.log("Got an event from S3: {}".format(message))

# Set use_sns=False to subscribe your Lambda directly to S3 events without forwrading␣
↪them through an SNS topic.
# That approach has fewer moving parts, but you can only subscribe one Lambda␣
↪function to events in a given S3 bucket.
@app.s3_event_handler(bucket="myS3bucket", events=["s3:ObjectCreated:*"], prefix="foo
↪", suffix=".bar", use_sns=False)
def monitor_s3(event, context):
    message = json.loads(event["Records"][0]["Sns"]["Message"])
    context.log("Got an event from S3: {}".format(message))

# DynamoDB event format: https://docs.aws.amazon.com/lambda/latest/dg/eventsources.
↪html#eventsources-ddb-update
@app.dynamodb_stream_handler(table_name="MyDynamoTable", batch_size=200)
def handle_dynamodb_stream(event, context):
    context.log("Got {} events from DynamoDB".format(len(event["Records"])))
```

(continues on next page)

(continued from previous page)

```
    context.log("First event: {}".format(event["Records"][0]["dynamodb"]))

# Use the following command to log a CloudWatch Logs message that will trigger this␣
↪handler:
# python -c'import watchtower as w, logging as l; L=l.getLogger(); L.addHandler(w.
↪CloudWatchLogHandler()); L.error(dict(x=8))'
# See http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/FilterAndPatternSyntax.
↪html for the filter pattern syntax
@app.cloudwatch_logs_sub_filter_handler(log_group_name="watchtower", filter_pattern="{
↪$.x = 8}")
def monitor_cloudwatch_logs(event, context):
    print("Got a CWL subscription filter event:", event)

# See http://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-
↪language.html
# See the "AWS Step Functions state machines" section below for a complete example of␣
↪setting up a state machine.
@app.step_function_task(state_name="Worker", state_machine_definition=state_machine)
def worker(event, context):
    return {"result": event["input"] + 1, "my_state": context.stepfunctions_task_name}
```

CHAPTER 1

Installation

```
pip install domovoi
```

CHAPTER 2

Usage

First-time setup:

```
domovoi new-project
```

- Edit the Domovoi app entry point in `app.py` using examples above.
- Edit the IAM policy for your Lambda function in `my_project/.chalice/policy-dev.json` to add any permissions it needs.
- Deploy the event handlers:

```
domovoi deploy
```

To stage files into the deployment package, use a `domovoilib` directory in your project where you would use `chalicelib` in Chalice. For example, `my_project/domovoilib/rds_cert.pem` becomes `/var/task/domovoilib/rds_cert.pem` with your function executing in `/var/task/app.py` with `/var/task` as the working directory. See the Chalice docs for more information on how to set up Chalice configuration.

2.1 Supported event types

See Supported Event Sources for an overview of event sources that can be used to trigger Lambda functions. Domovoi supports the following event sources:

- SNS subscriptions
- SQS queues (sample event)
- CloudWatch Events rule targets, including CloudWatch Scheduled Events (see CloudWatch Events Event Examples for a list of event types supported by CloudWatch Events)
- S3 events
- AWS Step Functions state machine tasks
- CloudWatch Logs filter subscriptions

- DynamoDB stream events

Possible future event sources to support:

- Kinesis stream events
- SES (email) events

2.2 AWS Step Functions state machines

Domovoi supports AWS Lambda integration with AWS Step Functions. Step Functions state machines can be started using the StartExecution method or the API Gateway Step Functions integration.

See the domovoi/examples directory for examples of Domovoi `app.py` apps using a state machine, including a loop that restarts the Lambda when it's about to hit its execution time limit, and a threadpool pattern that divides work between multiple Lambdas.

When creating a Step Functions State Machine driven Domovoi daemon Lambda, the State Machine assumes the same IAM role as the Lambda itself. To allow the State Machine to invoke the Lambda, edit the IAM policy (under your app directory, in `.chalice/policy-dev.json`) to include a statement allowing the "lambda:InvokeFunction" action on all resources, or on the ARN of the Lambda itself.

2.3 Configuration

2.3.1 Dead Letter Queues

To enable your Lambda function to forward failed invocation notifications to dead letter queuees, set the configuration key `dead_letter_queue_target_arn` in the file `.chalice/config.json` to the target DLQ ARN. For example:

```
{
  "app_name": "my_app",
  ...
  "dead_letter_queue_target_arn": "arn:aws:sns:us-east-1:123456789012:my-dlq"
}
```

You may need to update your Lambda IAM policy (`.chalice/policy-dev.json`) to give your Lambda access to SNS or SQS.

2.3.2 Concurrency Reservations

For high volume Lambda invocations in accounts with multiple Lambdas, you may need to set per-function concurrency limits to partition the overall concurrency quota and prevent one set of Lambdas from overloading another. In Domovoi, you can do so by setting the configuration key `reserved_concurrent_executions` in the file `.chalice/config.json` to the desired concurrency reservation. For example:

```
{
  "app_name": "my_app",
  ...
  "reserved_concurrent_executions": 500
}
```

CHAPTER 3

Links

- Project home page (GitHub)
- Documentation (Read the Docs)
- Package distribution (PyPI)
- Change log

3.1 Bugs

Please report bugs, issues, feature requests, etc. on GitHub.

CHAPTER 4

License

Licensed under the terms of the Apache License, Version 2.0. 

CHAPTER 5

API documentation

CHAPTER 6

Table of Contents