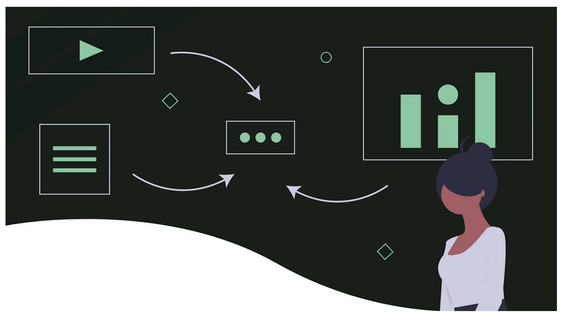
Serverless queues and workers — Designing Lift

Matthieu Napoli · Follow
Published in Serverless Transformation · 4 min read · Apr 30, 2021

This article is part of a series on Lift, an open-source project that simplifies deploying serverless applications.



As introduced in previous articles, **Lift is a Serverless Framework plugin** that simplifies deploying serverless applications.

As we approach the first beta planned for May, let's talk about **queues and workers**:

- How to deploy production-ready queues and workers on AWS today.
- How we plan to simplify that with Lift.

The naive approach

When going serverless, SQS + AWS Lambda is an iconic duo:

- SQS is a queue service that is entirely managed by AWS, scales automatically, and bills by the usage.
- AWS Lambda runs our code and is entirely managed by AWS, scales automatically, and bills by the usage.

The best part: Lambda integrates with SQS natively. We can write "queue workers" on Lambda without having to poll messages from SQS: instead, **our code (the worker) is automatically invoked when a new message (aka job) is pushed into SQS**.

To deploy queues and workers with the Serverless Framework, we need a bit of CloudFormation:

```
# serverless.yml
# ...

functions:
    worker:
        handler: worker.handler
        # Subscribe our function to the SQS queue
        -   sqs:
                arn: !GetAtt ReportGenerationQueue.Arn

resources:
    Resources:
        # This creates an SQS queue
        ReportGenerationQueue:
            Type: AWS::SQS::Queue
```

So far so good! Or at least, it's doable.

Problems with the naive approach

The biggest missing piece here is error handling.

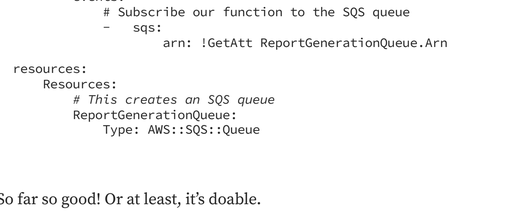
By default, SQS retries failed jobs indefinitely. Imagine your code has a bug (let's pretend that can happen): the job will be retried over and over for days, possibly costing a lot of money and wasting resources.

To deal with errors, we need to set up a "*dead letter queue*" (a queue that stores failed messages) and limit the number of retries.

On top of that, there are many details that needs to be configured, for example:

- We should configure the dead letter queue to store failed messages for 14 days (the maximum).
- We should set up an **alarm** that sends our team an email whenever there are failed messages in the dead letter queue.
- Many other settings need to be fine-tuned: the SQS "visibility timeout", the Lambda max concurrency, message batching, etc.

A production-ready approach



Here is a preview of a `serverless.yml` configuration that includes those best practices:

```
# serverless.yml
# ...

functions:
    worker:
        handler: worker.handler
        timeout: 10 # seconds
        # Scale to maximum 10 worker instances
        reservedConcurrency: 10
        events:
            -   sqs:
                    arn: !GetAtt ReportGenerationQueue.Arn
                    # Process messages without batching
                    batchSize: 1

resources:
    Resources:
        # Create an SQS queue
        ReportGenerationQueue:
            Type: AWS::SQS::Queue
            Properties:
                QueueName: !Sub '${AWS::StackName}-myqueue'
                # 6 times the lambda function's timeout
                VisibilityTimeout: 60
                # Messages will be stored (and retried) up to 2 days
                MessageRetentionPeriod: 172800
                RedrivePolicy:
                    # Jobs will be retried 5 times
                    maxReceiveCount: 5
                    # Send failed messages to the dead letter queue
                    deadLetterTargetArn: !GetAtt DeadLetterQueue.Arn

        # Failed messages will be sent to this queue
        DeadLetterQueue:
            Type: AWS::SQS::Queue
            Properties:
```

```
# Store messages up to 14 days (the max)
# we want time to debug/analyse
MessageRetentionPeriod: 1209600
```

If we wanted to add an alarm for failed messages, we would need 30 more lines of YAML.

Needless to say:

- this is hard to figure out,
- this is tedious to implement.

We want to make that simpler with Lift.

Serverless queues and workers with Lift

We are currently working on a “Queues” component that can be deployed via `serverless.yml`:

```
# serverless.yml
# ...

queues:
  report-generation:
    # Our Lambda worker function is defined in the "queue" component:
    worker:
      handler: worker.handler
      # Scale to maximum 10 worker instances
      reservedConcurrency: 10
```

As we can see in the example above, there is a new `queues` section that lets us define queues (and their workers). In that instance, we define a `report-generation` queue and its worker function.

On `serverless deploy`, Lift will create:

- A `report-generation` SQS queue configured following best practices.
- A `worker` Lambda function, that will be automatically subscribed to the SQS queue.
- A `report-generation-dlq` SQS “dead letter queue” that will receive failed messages.

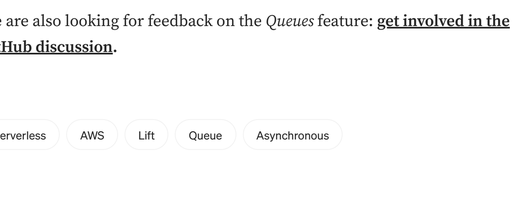
By default, messages are configured to be retried up to 3 times.

We are considering several settings on the component, including setting an email alert on failed messages:

```
# serverless.yml
# ...

queues:
  report-generation:
    alarm: alerting@my-company.com
    worker:
      ...
```

On top of that, Lift also simplifies **pushing messages** into the SQS queue:



- Lambda functions defined in `serverless.yml` will automatically get **IAM permissions** to send messages to the SQS queue.
- To simplify referencing the SQS queue, Lift introduces **a new variable syntax**: `${queues:my-queue.queueUrl}`

Here is a complete example with a *publisher* Lambda function:

```
# serverless.yml
# ...

functions:
  publisher:
    handler: publisher.handler
    environment:
      QUEUE_URL: ${queues:report-generation.queueUrl}

queues:
  report-generation:
    worker:
      handler: worker.handler
```

Deploying queues, static websites and file storage is a small part of what we are working on with Lift.

To follow Lift’s development, **star or watch Lift on GitHub**.

We are also looking for feedback on the *Queues* feature: **get involved in the GitHub discussion**.

Serverless AWS Lift Queue Asynchronous

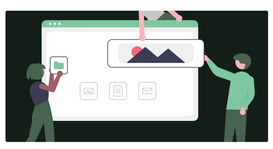
Written by Matthieu Napoli

234 Followers · Writer for Serverless Transformation

Make serverless more accessible

Follow

More from Matthieu Napoli and Serverless Transformation



Matthieu Napoli in Serverless Transformation

Static websites on AWS—Designing Lift

This article is part of a series on Lift, an open-source project that simplifies deploying...

3 min read · Apr 16, 2021

47 1

Xavier Lefèvre in Serverless Transformation

Asynchronous client interaction in AWS Serverless: Polling,...

Event-driven Serverless often requires async update to the client. There are several ways ...

6 min read · Apr 18, 2020

730 7

Xavier Lefèvre in Serverless Transformation

Sarah Hamilton in Serverless Transformation