WHITE PAPER

Serverless Reload Triggers with AWS Lambda

for Qlik Sense Enterprise SaaS

Daniel Pilla – Principal Analytics Platform Architect

TABLE OF CONTENTS

SUMMARY

- Trigger application reloads in Qlik Sense Enterprise SaaS remotely using AWS Lambda.
- Lambda supports both scheduled and event-driven triggers through AWS services like EventBridge and API Gateway.
- This paper explores the scheduled component using EventBridge, however it provides the Lambda function that could be triggered by any means.

INTRODUCTION

In most enterprises, there is a need for some type of third-party automation/scheduling. One of those areas is the triggering of application reloads. Qlik SaaS allows for basic scheduling of application reloads, but further complexity and integration may be required for an organization. While there are a multitude of technologies which can be used to interact with Qlik SaaS's RESTful APIs, this paper will be focusing on *serverless* execution of application reloads with AWS. Similar functionality (such as Azure Functions) exists in other cloud providers that could also be utilized.

Prerequisites

Basic knowledge of the AWS ecosystem is encouraged for this paper. This is not an instructional paper, but rather an exploration of the art of the possible. Examples will be provided but this is not intended to be a thorough walk-through of AWS specific configuration.

A high-level understanding of the following is encouraged:

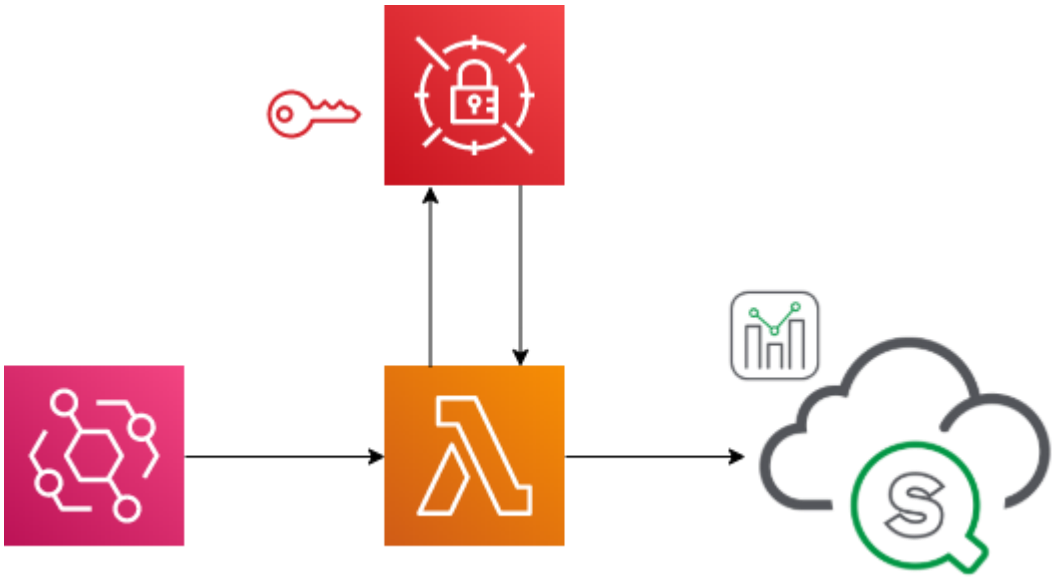
- Qlik SaaS: How to get an API key in Qlik Sense Enterprise SaaS
    - Required to interact with the underlying Qlik SaaS RESTful APIs
- AWS: Lambda

    - Used to execute application refreshes in Qlik SaaS using serverless technology

- AWS: IAM Roles
    - Used to delegate access to the Qlik SaaS API key stored in AWS Secrets Manager
- AWS: EventBridge
    - Used to build execution schedules for the Lambda function(s)
- AWS: Secrets Manager
    - Used to store the API key for Qlik SaaS. This method allows for additional security and reusability as opposed to using hard-coded API keys in the Lambda function(s).

Scenario

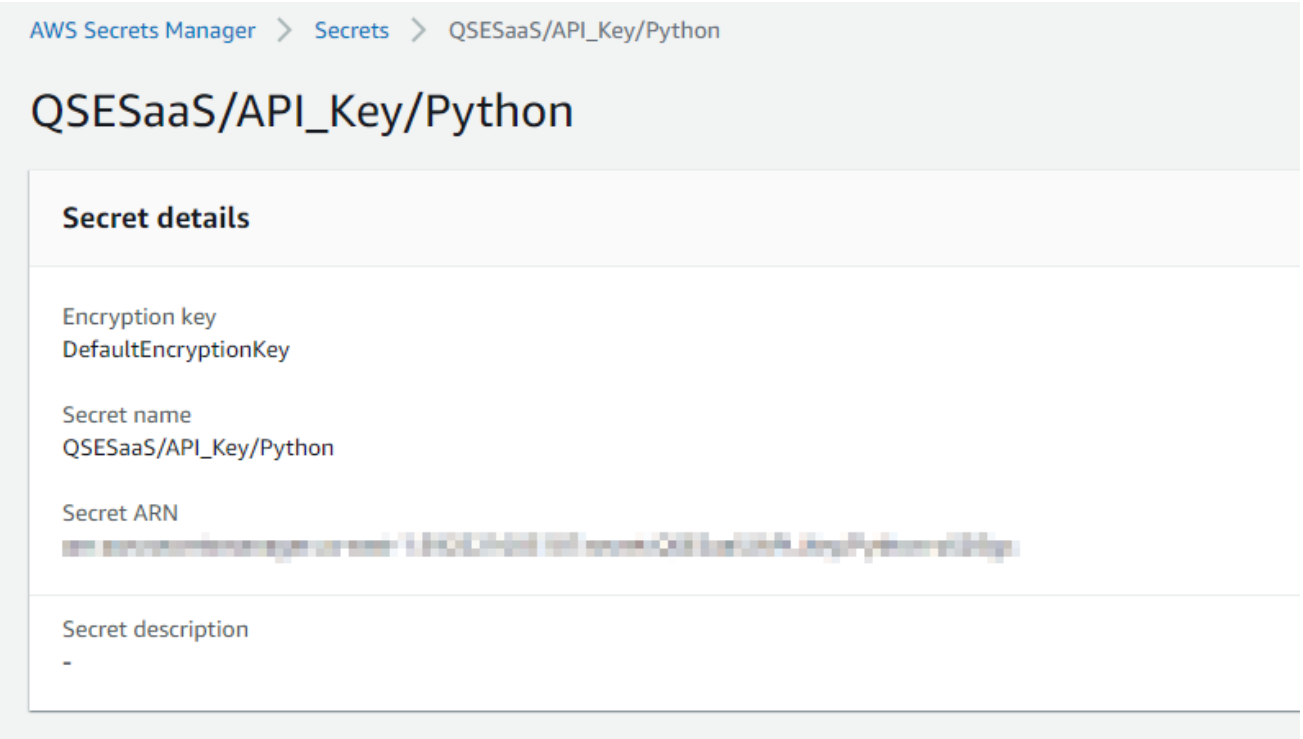
An EventBridge rule will be built so that on a schedule it passes the target application ID to a Lambda function that then calls the Qlik Sense Enterprise SaaS endpoint to reload the application. The Qlik Sense Enterprise SaaS API key is stored securely using Secrets Manager.



Secrets Manager

The example Lambda code in this paper leverages AWS Secrets Manager for the storage of the Qlik Sense Enterprise SaaS API key. Here is an example tutorial if this topic is unfamiliar. The Lambda function will then need the appropriate policy in place to access the secret. Once the secret

is created, AWS will provide sample code in the desired language to fetch it, which the Lambda code later in this paper will leverage.



Here is an example IAM policy that could be applied for the Lambda function (please review internally and do not blindly apply this policy):

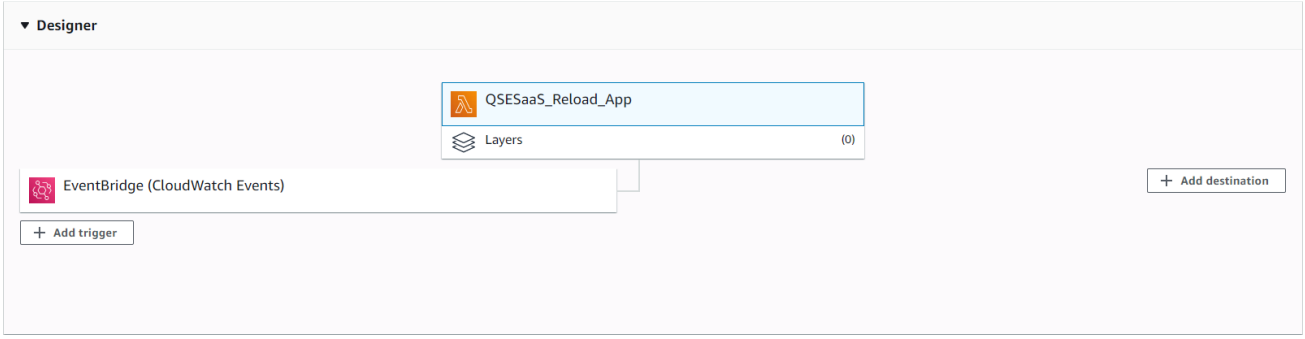
```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetResourcePolicy",
            "secretsmanager:GetSecretValue",
            "secretsmanager:DescribeSecret",
            "secretsmanager:ListSecretVersionIds"
        ],
        "Resource": "<SECRET ARN>"
    }]
}
```

Lambda Function

This section will explore the contents of the example Lambda function. In this scenario, the function is written in Python 3, and is using the Python 3.7 runtime. It leverages the Requests library to work with Qlik Sense Enterprise SaaS's Restful APIs.

Example Lambda Function Code

```python
import json
import os
import boto3
from botocore.exceptions import ClientError
import time
from botocore.vendored import requests

def get_secret():

    # secret name and region name stored to environmental variables
    secret_name = os.environ["secret_name"]
    region_name = os.environ["region_name"]

    # Create a secrets manager client
    session = boto3.session.Session()
    client = session.client(service_name="secretsmanager", region_name=region_name)

    try:
        get_secret_value_response = client.get_secret_value(SecretId=secret_name)
        return 200, json.loads(get_secret_value_response["SecretString"])["key"]
    except ClientError as e:
        statusCode = e.response['ResponseMetadata']['HTTPStatusCode']
        response = e.response['Error']['Message']
        return statusCode, response
    except Exception as e:
        return 500, str(e)

def lambda_handler(event, context):

    try:
        # Grab the api key from Secrets Manager
        statusCode, response = get_secret()
        if statusCode != 200:
            raise Exception(response)

        # Grab the FQDN from the environmental variable
        fqdn = os.environ["FQDN"]

        # Grab the app id from the EventBridge event
        appId = event["appId"]

        # Setup a session and add headers
        s = requests.Session()
        s.headers.update({"Authorization": "Bearer " + response})

        # Check if the app exists
        r = s.get(fqdn + "/api/v1/apps/" + appId)
        statusCode = r.status_code

        if statusCode == 200:
            # Reload target app and grab the reload id and status
            r = s.post(fqdn + "/api/v1/reloads", data=json.dumps({"appId": appId}))
            statusCode = r.status_code

            # Check if the reload task was created (201)
            if statusCode == 201:
                rjson = r.json()

                # Grab the reload status and id
                reloadStatus, reloadId = rjson["status"], rjson["id"]

                # Poll every 3 seconds until the reload has started (or quickly finished)
                while True:
                    time.sleep(3)
                    r = s.get(fqdn + "/api/v1/reloads/" + reloadId)
                    statusCode = r.status_code

                    if statusCode == 200:
                        reloadStatus = r.json()["status"]
                        print(reloadStatus)

                        if reloadStatus in ("RELOADING", "SUCCEEDED", "FAILED"):
                            body = "The application began the reload process with status: " + reloadStatus
                            break
                    else:
                        body = "Something went wrong while checking the reload status."
                        break
            else:
                body = "Something went wrong while creating the reload."
        elif statusCode == 404:
            body = "The target application does not exist."
        else:
            body = "Something went wrong."
        return {"statusCode": statusCode, "body": body}
    except Exception as e:
        return {"statusCode": statusCode, "body": str(e)}
```



The function itself creates a new reload entity with the target application ID (providing the app ID is valid), then it queries that newly created reload entity and will loop until it finds a value of 'RELOADING', 'SUCCEEDED', or 'FAILED', whichever comes first during the polling interval (typically 'RELOADING'). As Lambda is billed by CPU time, it would not be wise to have the function wait around for it to finish. This approach ensures that the application reload request has been successfully executed before closing (or has finished/failed very quickly). The poll wait time is hardcoded at every 3 seconds.

The script leverages three Lambda environmental variables including:

- FQDN
    - Key: FQDN
    - Value: https://<your_tenant>.<region>.qlikcloud.com
- Region Name (of the secret)
    - Key: region_name
    - Value: <aws region> (e.g. us-east-1)
- Secret Name (of the secret)
    - Key: secret_name
    - Value: <secret name>

Environment variables (3)

The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
FQDN	[illegible]
region_name	us-east-1
secret_name	QSESaaS/API_Key/Python

The script also takes it's input from the EventBridge rule in the form of the following JSON example:

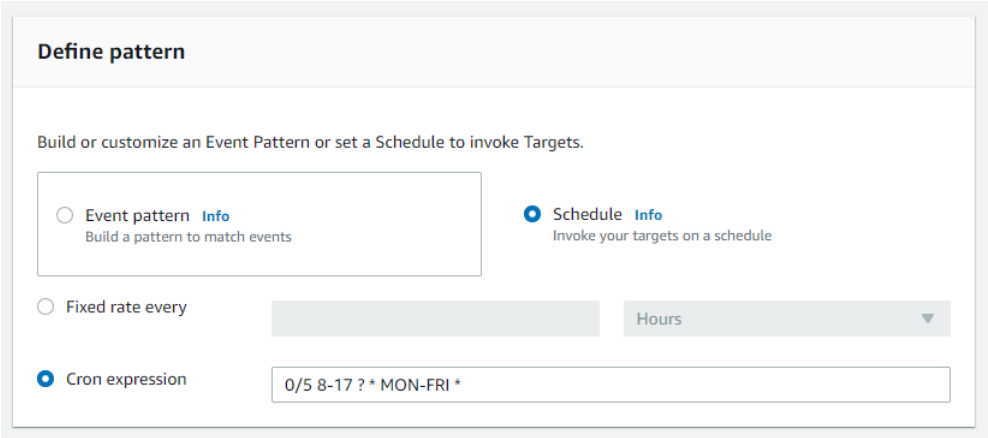
```
{ "appId" : "<GUID>" }
```

Knowing that the Lambda function requires the *appId* param to run, ensure a Lambda test scenario is setup with the above input, including a valid application ID, as it will require that param to execute properly.

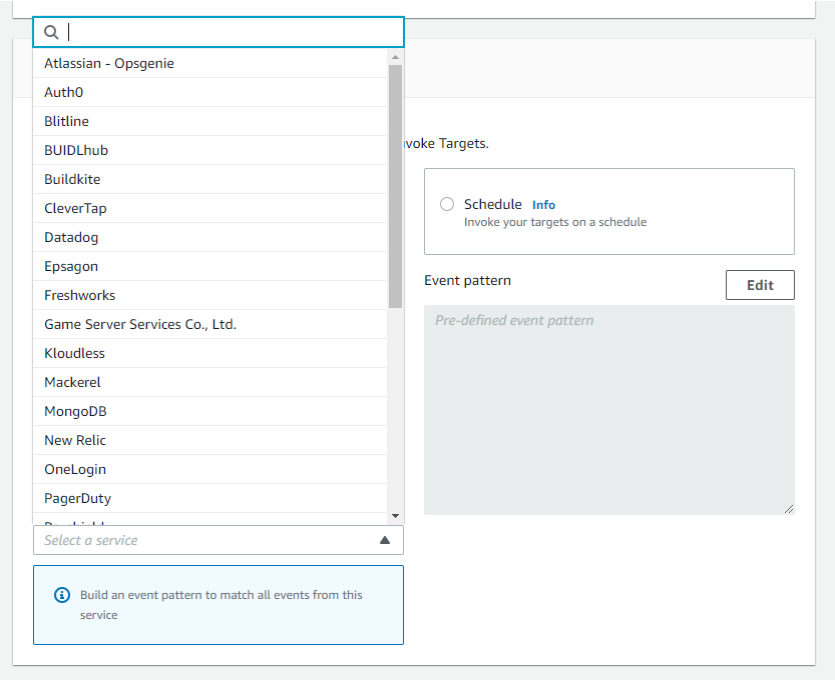
EventBridge

In this scenario, the EventBridge rule is simply going to trigger the Lambda function on a schedule passing a hardcoded application ID. EventBridge supports basic scheduling via the UI (e.g. every 10 minutes), as well as cron based scheduled for more complex scenarios.

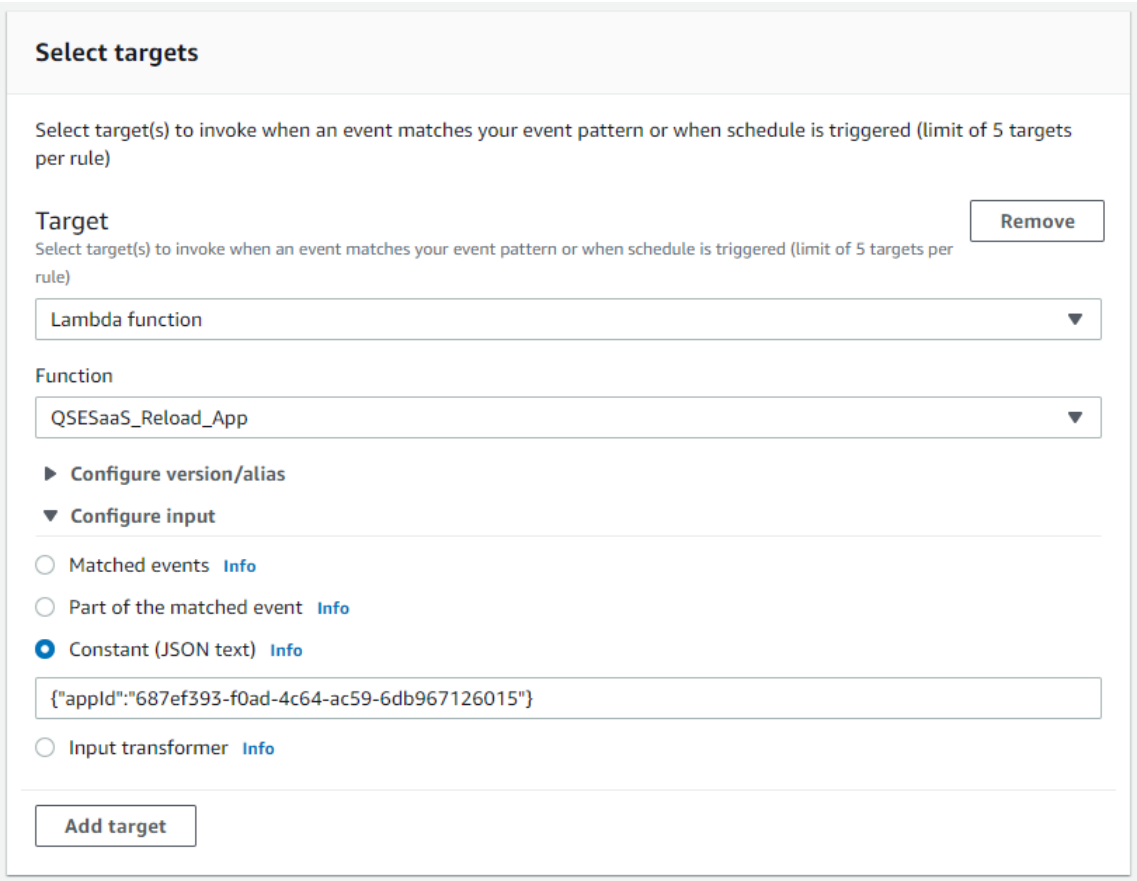
In the sample below, the Cron expression is set to reload every 5 minutes Monday through Friday between 8:00 am and 5:55 pm (UTC). This example is directly from the AWS EventBridge docs found here.



As this article has mentioned, the trigger doesn't have to be a schedule, it could also be sent through from many different sources:



The target in this example will be the Lambda function that was created, and the input will be the hardcoded JSON that specifies the app that should be reloaded. Note that if this was an event-driven trigger from another service, the application ID could be dynamically pulled/derived and provided by the integration with an outside service.



Summary

This paper is meant to be an introductory exploration into serverless triggering of application reloads in Qlik Sense Enterprise SaaS.

About Qlik

Qlik's vision is a data-literate world, where everyone can use data and analytics to improve decision-making and solve their most challenging problems. Qlik provides an end-to-end, real-time data integration and analytics cloud platform to close the gaps between data, insights and action. By transforming data into active intelligence, businesses can drive better decisions, improve revenue and profitability, and optimize customer relationships. Qlik does business in more than 100 countries and serves over 50,000 customers around the world.

qlik.com