

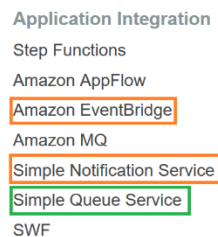
## Module 18

### Amazon Simple Notification Service (SNS)

#### Overview

In Module 15 we looked at a queuing service known as SQS. Queues are widely used in cloud applications. For example in Project 2, two queues are used ferry messages back and forth between two disconnected systems: a cloud Lambda and a local network database. The most important characteristic of queues is that they decouple systems from each other.

Two other important services under the *Application Integration* family of services are Amazon EventBridge and Simple Notification Service (SNS) (Figure 1). We will look at both in this module.



**Figure 1:** Application integration services in AWS.

#### Amazon EventBridge

So far you have already used event-based execution in AWS. For example, you know that you can configure a Lambda with different event triggers. One example we have repeatedly used is how an S3 event can trigger a Lambda to execute. Another example we used is configure a Lambda to be triggered by a message that arrives at a queue (Project 2). In both cases we were able to link such services with minimal fuss using a few clicks. This is so because AWS realized that these are common scenarios and created the events for us out of the box - All we had to do is just configure them.

However, AWS cannot foresee and build all possible event triggers that customers may need in their custom applications. Instead, AWS created the infrastructure to allow us to create our own event triggers to accomplish event-driven execution. This mechanism allows integration between systems without tightly coupling them. The way EventsBridge works is that a component of a system puts messages in an event bus. Any other component can subscribe and get notified of messages.

Let's look at some example where we might need to execute some code in response to some situation:

CloudWatch comes with default metrics. For example there is a metric that tells you how many times a Lambda function was called. There is another metric that tells you the number of invocation errors, etc. CloudWatch also allows you to create custom metrics. Built-in metrics (e.g., Invocations, Errors count,

etc.) are generic in nature and do not necessarily capture important variables your application is monitoring or working with. Custom metrics, however, allow you to emit metrics related to important variables in your applications. CloudWatch also allows you to configure alarms - when an alarm goes off, you can configure CloudWatch to send a notification (e.g., email, text message, etc.). This is good if what we want is to just alert an engineer about a situation that needs close monitoring or a fix, or to enter a bug ticket in a bug tracking system. Sometimes, however, we might have a need to do something more than just notification. What if we want an alarm to say trigger some code to run? – maybe code of some other Lambda function. For instance, I might have this requirement in my application: when the number of login attempts by customers to my service exceed 100 per second, launch 2 new EC2 instances and configure them to take on part of the load. I can write code in a Lambda function that uses the EC2 SDK to launch 2 instances. But how do I make this code execute when the “greater-than-100-logins-per-second alarm” goes off? This is where EventBridge comes in.

*“Amazon EventBridge is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as AWS Lambda. You can set up routing rules to determine where to send your data to build application architectures that react in real time to all of your data sources. EventBridge allows you to build event driven architectures, which are loosely coupled and distributed.”* (From the EventBridge [User Guide](#)).

**We will not be able to do a full hands-on EventBridge exercise because the AWS Academy account does not provide us the needed privileges to do it.**

**Next time you are working on a cloud application and you have a need to have an event/situation triggers something else, if you don’t find a built-in event notification, then keep EventBridge in mind and investigate if it can accomplish what you want.**

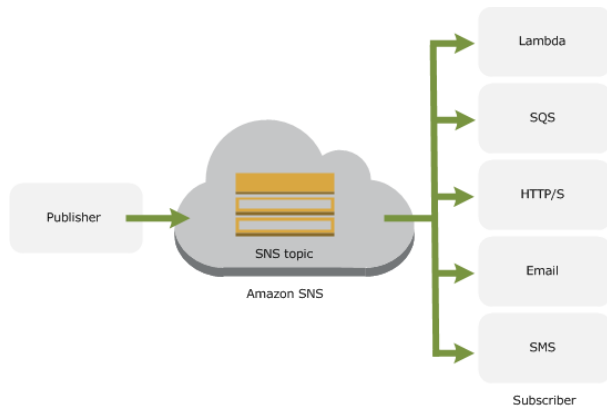
## Simple Notification Service (SNS)

SNS allows you to simulate event-driven systems integration and notification via a Publish/Subscribe mechanism.

First read about what is SNS [here](#).

Then look at [common scenarios](#) where SNS can be used.

Like any Publish/Subscribe pattern (whether in the cloud or not), one of the benefits of a Publish/Subscribe pattern is that it decouples the publisher from the subscribers. The publisher doesn’t need to know how many subscribers there are. And subscribers can come in and leave at any time (Figure 2).



**Figure 2:** Amazon SNS ([source](#)).

## Working with SNS from the Management Console

### Steps

1. Login to the AWS Management Console, and go to the SNS service (Simple Notification Service).
2. In the navigation bar on the left, click **Topics**.
3. Click the **Create topic** button.
4. Choose the **Standard** type.
5. Fill in the Details section as shown below

**Details**

**Type** [Info](#)  
Topic type cannot be modified after topic is created

☐ **FIFO (first-in, first-out)**

- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

☒ **Standard**

- Best-effor
- At-least o
- Highest tr
- Subscripti
- email, mo

**Name**  
  
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (\_).

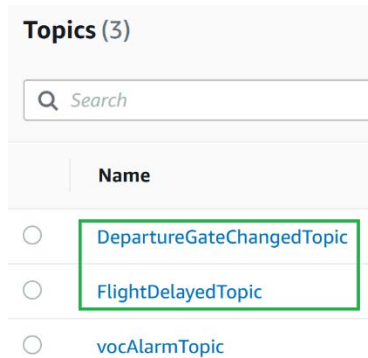
**Display name - optional**  
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are  
  
Maximum 100 characters, including hyphens (-) and underscores (\_).

6. Leave the default for everything else. Click the **Create topic** button.

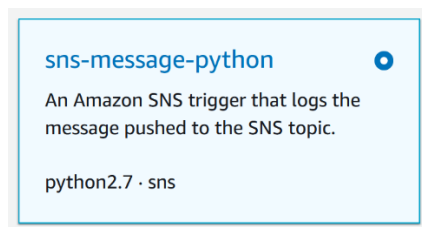
7. In the navigation bar on the left, click **Topics** again and create an additional topic with the following characteristics:

Type: Standard  
Name: DepartureGateChangedTopic  
Display name: Gate Changed

You should now have the two topics you just created:



8. In the AWS Management Console, go to the **Lambda** service, and click the **Create function** button.
9. Choose the **Use a blueprint** option.
10. In the Blueprints Textbox, search for: Keyword : SNS
11. Choose **sns-message-python**, and click the **Configure** button.



12. Fill in the Basic information section as follows:

Function name: FlightDelayedFunction  
Role: choose Lab Role

13. In the SNS trigger section, choose the FlightDelayedTopic.

14. Click the **Create function** button.

Copy the ARN of this function (at top of page). You need this value in step 21 below.

15. In the AWS Management Console, navigate back to the SNS service.
16. Click the **Topics** link on the navigation bar on the left.
17. Click the FlightDelayedTopic.
18. In the Subscriptions section, click the **Create subscription** button.
19. Choose the FlightDelayedTopic ARN
20. For Protocol, choose AWS Lambda.
21. For Endpoint, paste the ARN of the FlightDelayedFunction.
22. Click the **Create subscription** button.
23. Click on Topics again and click the FlightDelayedTopic to open its properties.
24. Click the **Publish message** button.
25. In the Message body Textbox at the bottom, enter this:

```
{  
  "flight" : "AA-221",  
  "destination" : "San Francisco",  
  "scheduled-on" : "6/1/2021 4:30 PM",  
  "delayed-by" : "45 min"  
}
```

26. Click the **Publish message** button.
27. Go back to the FlightDelayedFunction Lambda function and look at the log in CloudWatch.  
Verify that you see the topic you published get printed.

▶	2021-06-01T12:57:29.042-07:00	From SNS: {
▶	2021-06-01T12:57:29.042-07:00	"flight" : "AA-221",
▶	2021-06-01T12:57:29.042-07:00	"destination" : "San Francisco",
▶	2021-06-01T12:57:29.042-07:00	"scheduled-on" : "6/1/2021 4:30 PM",
▶	2021-06-01T12:57:29.042-07:00	"delayed-by" : "45 min"
▶	2021-06-01T12:57:29.042-07:00	}

In the above case we used Lambda as the message protocol. In the next example, we will use SMS.

28. In the AWS Management Console, navigate back to the **SNS** service.
29. Go to Topics, and click the DepartureGateChangedTopic.
30. Click the **Create subscription** button.
31. For Protocol, select SMS.
32. For endpoint, enter your mobile phone number (with the +1 in front of it like +14253334444)
33. Click the **Create subscription** button.
34. Click on Topics again and choose the DepartureGateChangedTopic topic.
35. In the Message body Textbox enter:  

`Departure gate for flight AA-221 was changed to D12.`
36. Click the **Publish message** button.
37. Verify that you received the text message on your phone.
38. Click the Subscriptions link in the navigation bar.
39. Select the subscription with the SMS protocol.
40. Click the **Delete** button. Then click Delete on the confirmation dialog.

## Working with SNS from a Client

### Steps

1. Create a Visual Studio solution named SNSSolution.
2. In the SNSSolution, create a project of type Console App (.NET Core), and name it SNSClient.
3. Add to the project references to the packages AWSSDK.Core and AWSSDK.SimpleNotificationService.

4. Complete the code of the program to do the following:

- a) Ask the airline passenger for a phone number (in this case enter your own cell phone number).

We can assume that this client is an airline phone app that the user has on his/her phone.

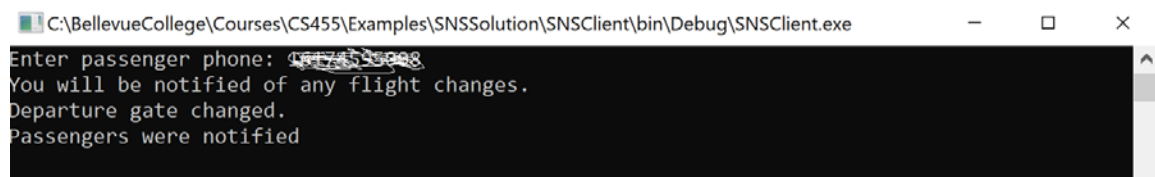
- b) Subscribe with the above number to the DepartureGateChangedTopic (use “sms” for the Protocol)
- c) Inform the user that he/she will be notified of any flight changes.
- d) Simulate that a gate change took place and publish a request to the DepartureGateChangedTopic (in a real scenario this publishing step won’t happen from the user’s app, but from the airline computer system at the airport).
- e) Verify that you received a SMS message of gate changes.

#### HINTS:

Classes you need to use: `AmazonSimpleNotificationServiceClient`, `SubscribeRequest`, `SubscribeResponse`, `PublishRequest`, `PublishResponse`.

You also need your `departureGateChangedTopicArn` which you can store in a string.

The output of your program should look something like Figure 3:



```
C:\BellevueCollege\Courses\CS455\Examples\SNSSolution\SNSClient\bin\Debug\SNSClient.exe
Enter passenger phone: 425-555-9998
You will be notified of any flight changes.
Departure gate changed.
Passengers were notified
```

**Figure 3:** Example output of SNSClient.

#### What to Submit:

Nothing to submit for this module