Simple Notification Service (SNS)

18. Simple Notification Service (SNS)

Example:

Messages are sent into the queue by Application A and they are processed by Application B. If Application B fails, Application A doesn't experience any disruption. Messages being sent can still be sent to the queue and will remain there until they are eventually processed.

This is loosely coupled. This is what we strive to achieve with architectures on AWS. And this brings me to two AWS services that can assist in this regard. Amazon Simple Queue Service or SQS and Amazon Simple Notification Service or SNS. But before I dive into those two services, let me just order a to-go coffee on our cafe website. Done. All right, well, that's great. I should get a message when that order is ready.

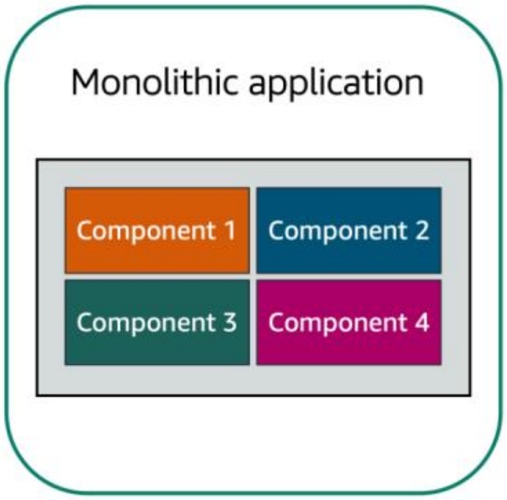
First up, let's discuss Amazon SQS. SQS allows you to send, store, and receive messages between software components at any volume. This is without losing messages or requiring other services to be available. Think of messages as our coffee orders and the order board as an SQS queue. Messages have the person's name, coffee order, and time they ordered. The data contained within a message is called a payload, and it's protected until delivery. SQS queues are where messages are placed until they are processed. And AWS manages the underlying infrastructure for you to host those queues. These scale automatically, are reliable, and are easy to configure and use.

Now, Amazon SNS is similar in that it is used to send out messages to services, but it can also send out notifications to end users. It does this in a different way called a publish/subscribe or pub/sub model. This means that you can create something called an SNS topic which is just a channel for messages to be delivered. You then configure subscribers to that topic and finally publish messages for those subscribers. In practice, that means you can send one message to a topic which will then fan out to all the subscribers in a single go. These subscribers can also be endpoints such as SQS queues, AWS Lambda functions, and HTTPS or HTTP web hooks.

Additionally, SNS can be used to fan out notifications to end users using mobile push, SMS, and email. Taking this back to our coffee shop, we could send out a notification when a customer's order is ready. This could be a simple SMS to let them know to pick it up or even a mobile push.

In fact, it looks like my phone just received a message. Looks like my order is ready.

Monolithic applications and microservices

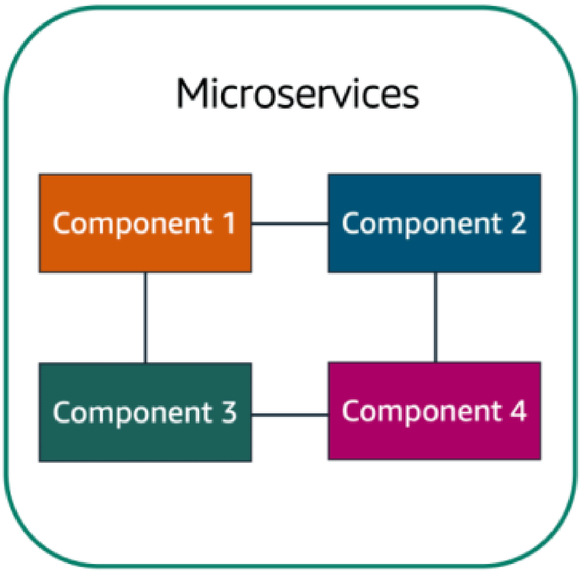


Applications are made of multiple components. The components communicate with each other to transmit data, fulfill requests, and keep the application running.

Suppose that you have an application with tightly coupled components. These components might include databases, servers, the user interface, business logic, and so on. This type of architecture can be considered a **monolithic application**.

In this approach to application architecture, if a single component fails, other components fail, and possibly the entire application fails.

To help maintain application availability when a single component fails, you can design your application through a **microservices** approach.



In a microservices approach, application components are loosely coupled. In this case, if a single component fails, the other components continue to work because they are communicating with each other. The loose coupling prevents the entire application from failing.

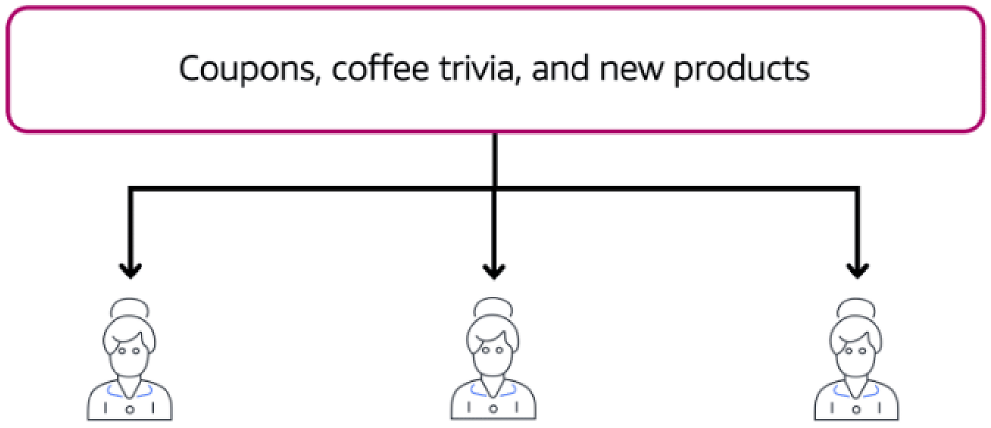
When designing applications on AWS, you can take a microservices approach with services and components that fulfill different functions. Two services facilitate application integration: Amazon Simple Notification Service (Amazon SNS) and Amazon Simple Queue Service (Amazon SQS).

Amazon Simple Notification Service (Amazon SNS)

Amazon Simple Notification Service (Amazon SNS) is a publish/subscribe service. Using Amazon SNS topics, a publisher publishes messages to subscribers. This is similar to the coffee shop; the cashier provides coffee orders to the barista who makes the drinks.

Step 1

Publishing updates from a single topic

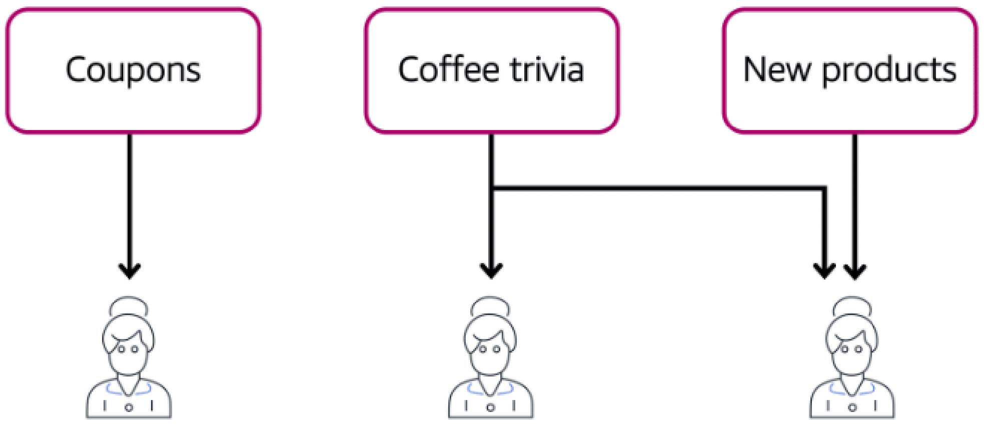


Suppose that the coffee shop has a single newsletter that includes updates from all areas of its business. It includes topics such as coupons, coffee trivia, and new products. All of these topics are grouped because this is a single newsletter. All customers who subscribe to the newsletter receive updates about coupons, coffee trivia, and new products.

After a while, some customers express that they would prefer to receive separate newsletters for only the specific topics that interest them. The coffee shop owners decide to try this approach.

Step 2

Publishing updates from multiple topics



Now, instead of having a single newsletter for all topics, the coffee shop has broken it up into three separate newsletters. Each newsletter is devoted to a specific topic: coupons, coffee trivia, and new products.

Subscribers will now receive updates immediately for only the specific topics to which they have subscribed.

It is possible for subscribers to subscribe to a single topic or to multiple topics. For example, the first customer subscribes to only the coupons topic, and the second subscriber subscribes to only the coffee trivia topic. The third customer subscribes to both the coffee trivia and new products topics.