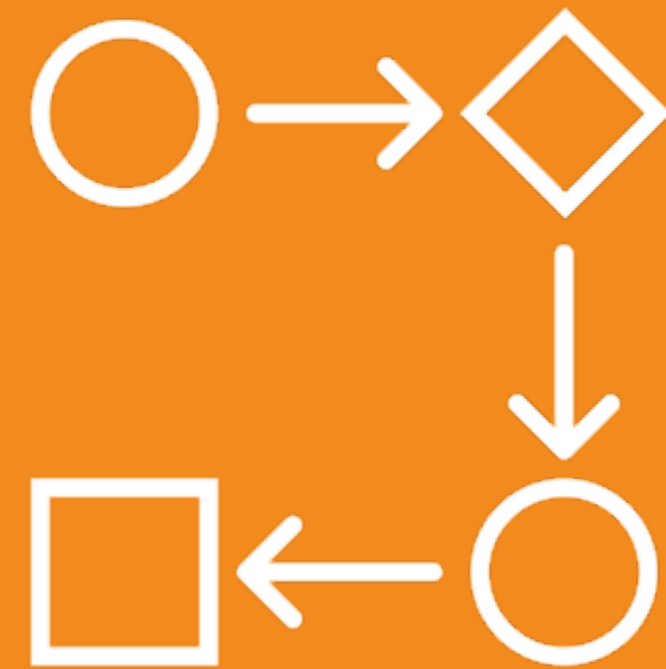


CF SUMMIT 2021

# Scalable, Responsive Apps and Services with Queues and Pub/Sub Mechanisms

BRIAN KLAAS

# Workflows





# Buy Now with 1-Click

Click Button

Validate Request

Check Inventory

Process Payment

Find Closest Warehouse

Create Pick Ticket

Calculate Ship Date

Send Confirmation Email

Success View

Click Button

Success  
View

Nothing's faster  
than work you don't have to wait for.

CF SUMMIT 2021

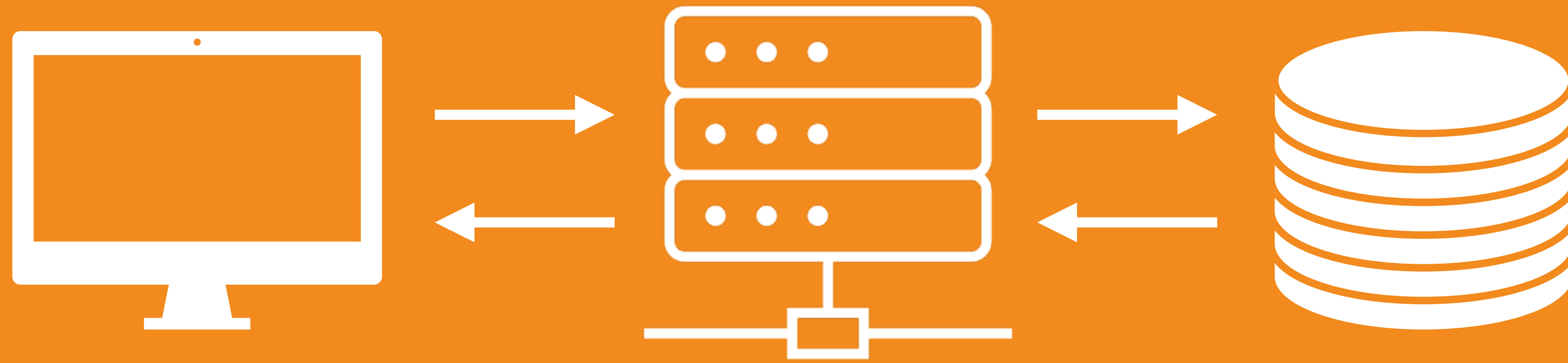
# Scalable, Responsive Apps and Services with Queues and Pub/Sub Mechanisms

BRIAN KLAAS

Request/Response







We've got to wait around for some ill-defined future point.

# runAsync() in ColdFusion 2018+

```
runAsync(validateRequest)
  .then(checkInventory)
  .then(processPayment)
  .then(findClosestWarehouse)
  .then(createPickTicket)
  .then(calculateShipDate);
  .then(sendEmailConfirmation);
```

```
var clientView = runAsync(validateRequest)
    .then(processPayment)
    .then(sendEmailConfirmation);
```

```
var warehouseProcess = runAsync(findClosestWarehouse)
    .then(checkInventory)
    .then(createPickTicket)
    .then(calculateShipDate);
    .then(sendShipmentDateEmail);
```

Error handling

Retries

Throttling

New business requirements

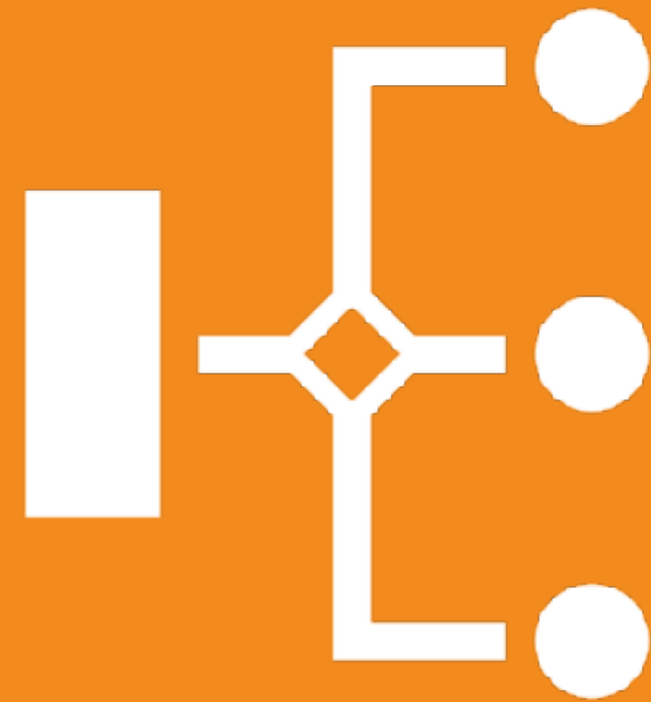
Linear, linked flow encourages  
brittle architectures

Linear, linked flow blocks  
your ability to scale



How can we do better?

Event-driven



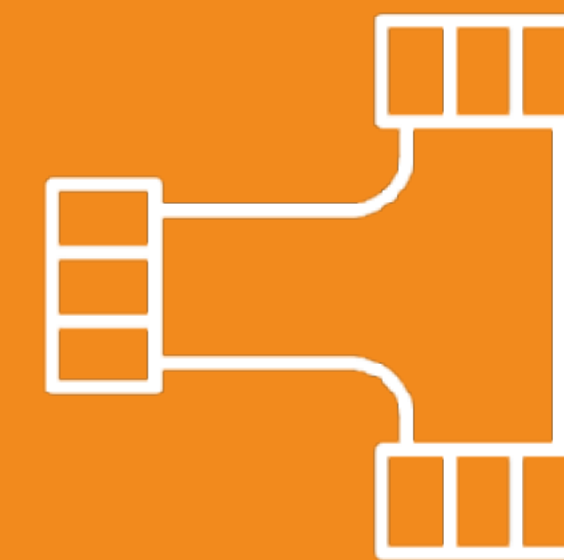
Something happens.  
Code responds.

# Database triggers

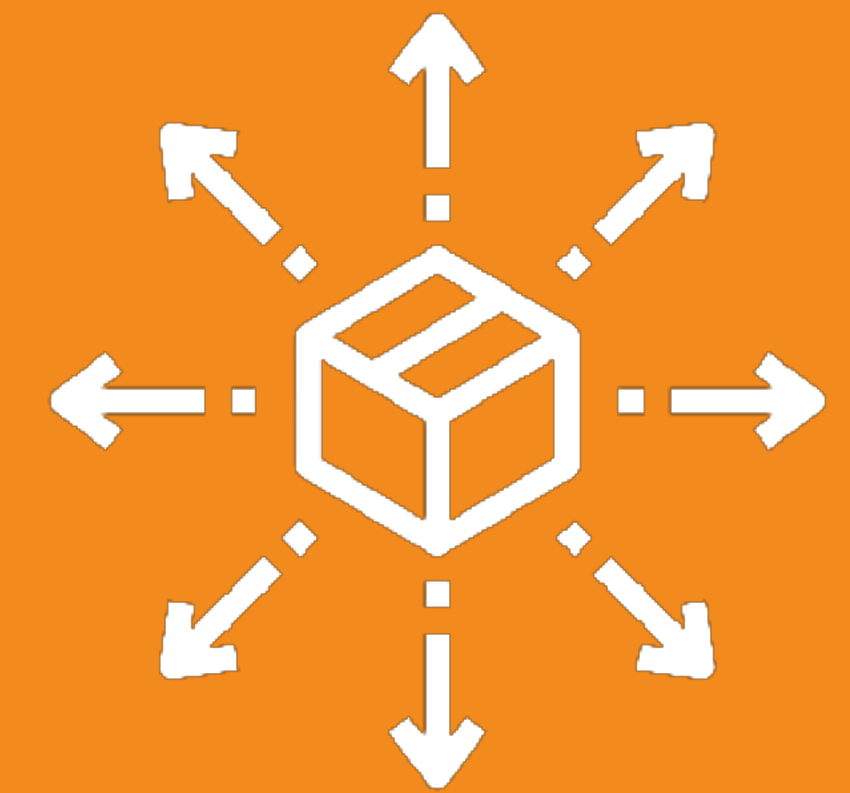


SQL

Event-driven = automatic plumbing



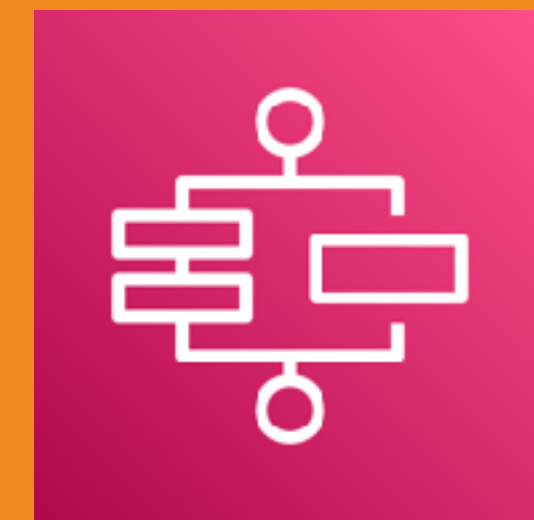
Event-driven = easy fan-out



How do you know it's done?



# Orchestration tools





Orchestration = additional complexity



Something simpler?

# Pub/Sub + Queues



SNS = Simple Notification Service



SNS = Pub/Sub



SNS = One Publisher,  
Many Subscribers



# SNS Subscribers:

https endpoints (including CFML)

Email

Phone number (SMS)

SQS queue

Lambda

Kinesis Firehose

Pinpoint application



# SNS subscribers can:

Filter on specific criteria

Retry on delivery to https endpoints

Specify DLQ on completely failed delivery



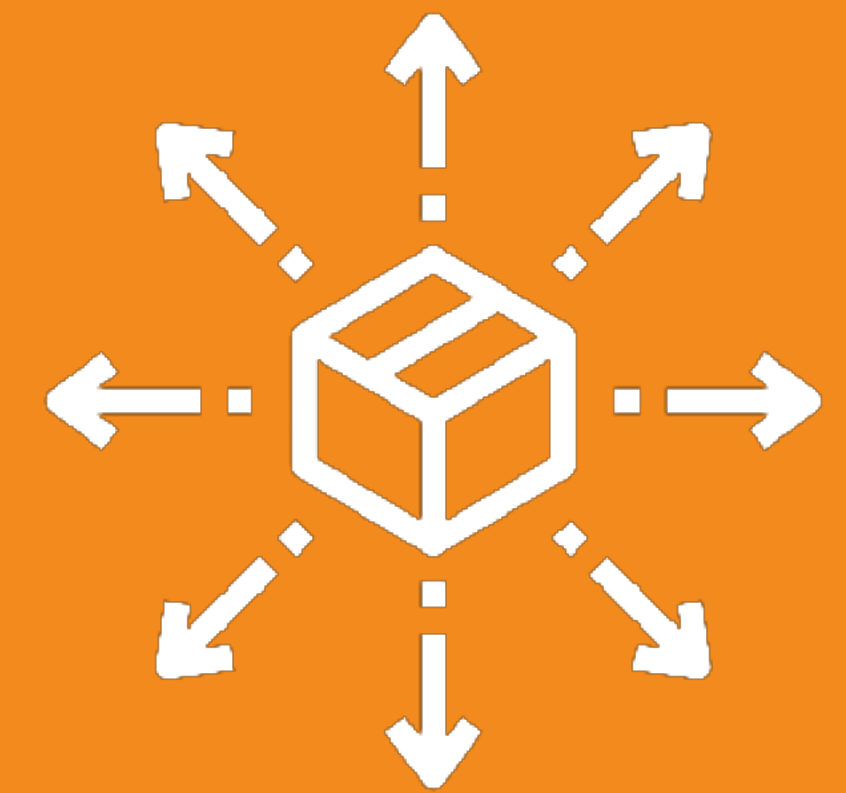


```
snsService= getCloudService(awsCredentials, {"serviceName" : "SNS"});  
topic = snsService.createTopic(topicName);  
msgBody = {"customer": 123, "orderId": 456, "amount": "78.90"};  
topic.publish(msgBody);
```

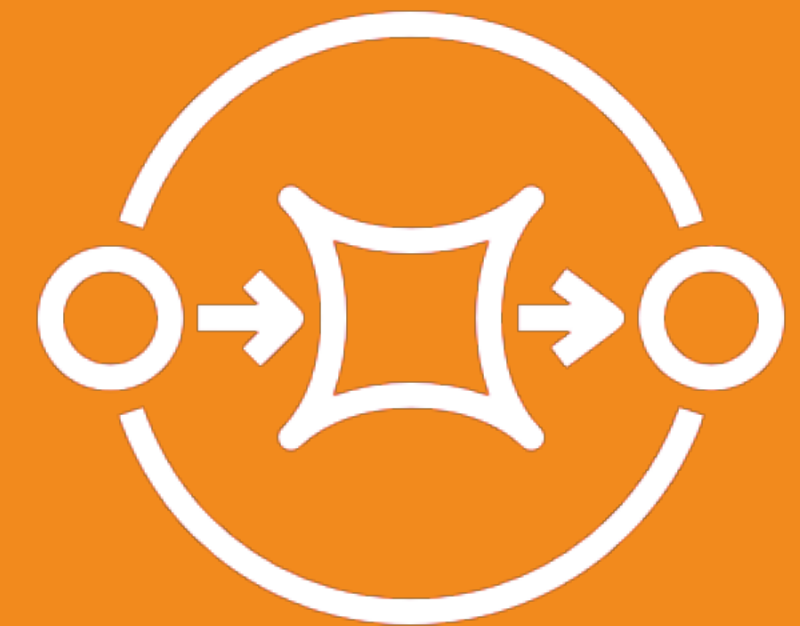
[github.com/brianklaas/awsplaybox](https://github.com/brianklaas/awsplaybox)



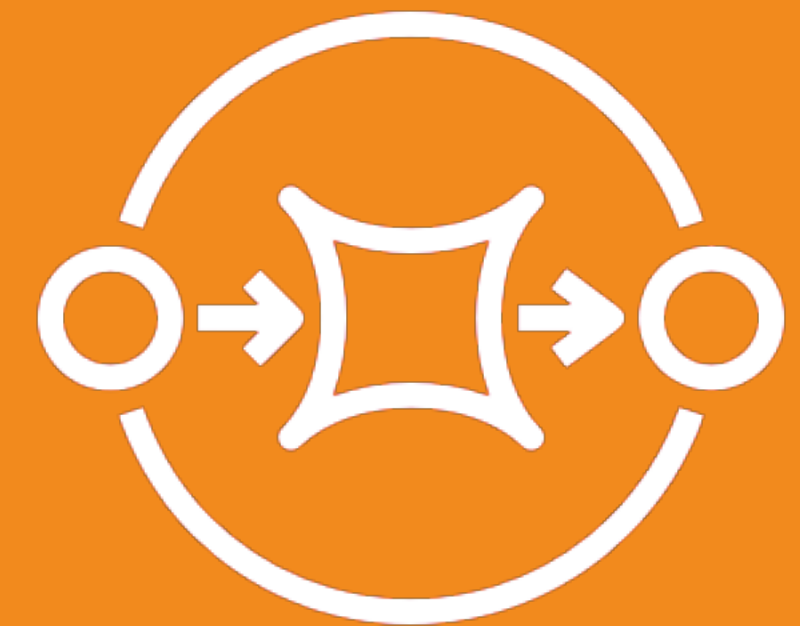
SNS = Fan-out



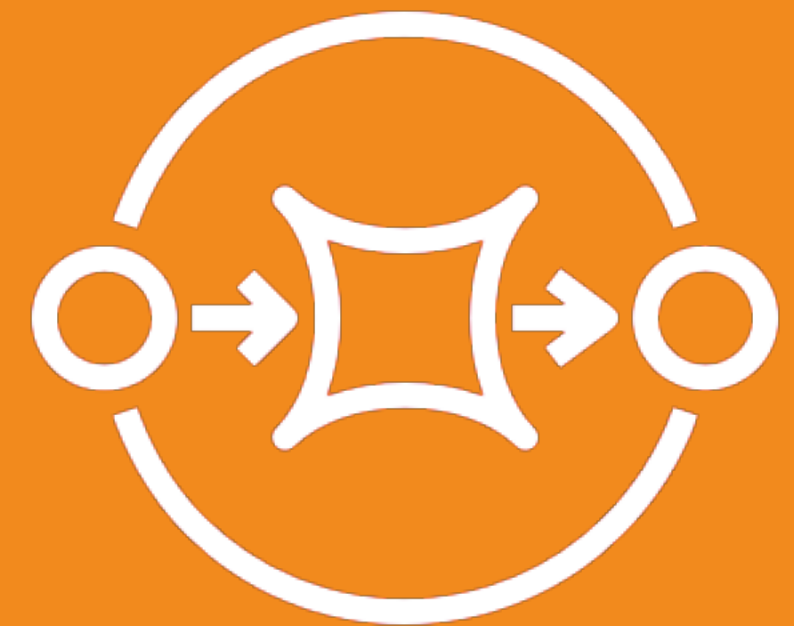
SQS = Simple Queue Service

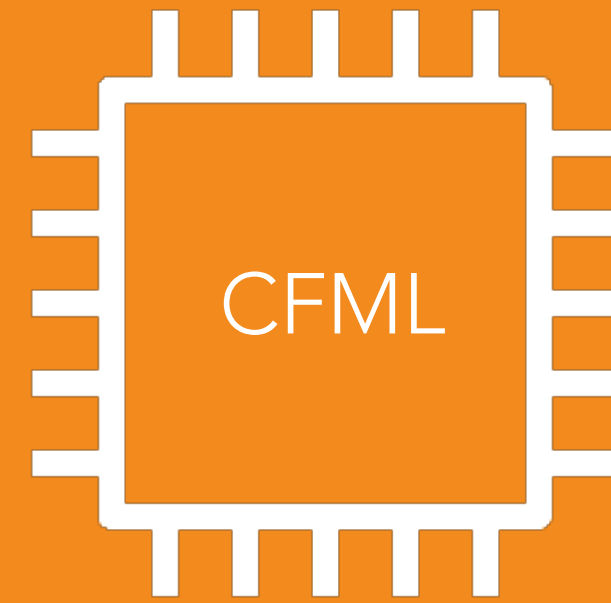
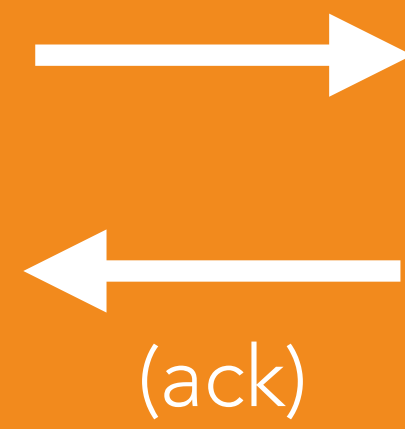


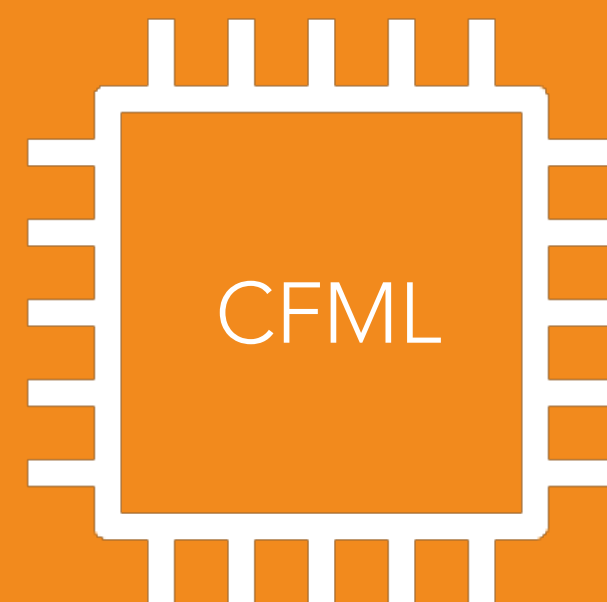
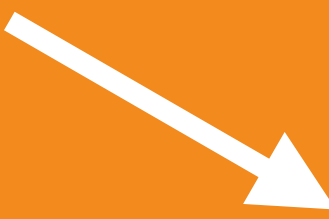
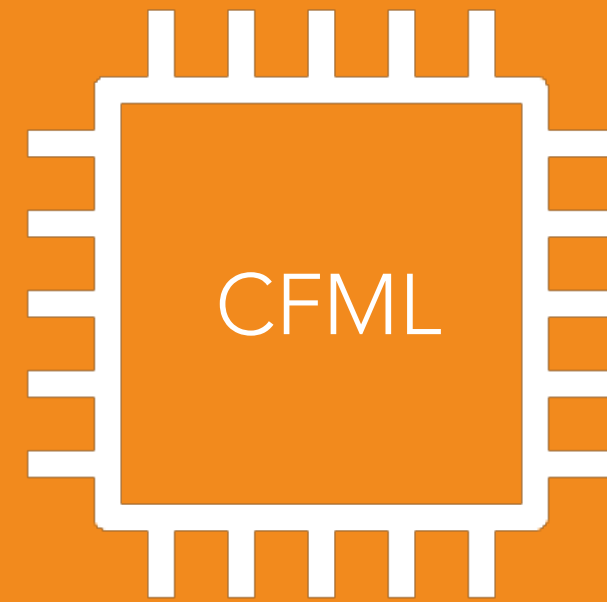
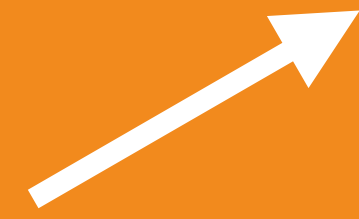
SQS = Stack of messages



SQS = One publisher,  
one worker

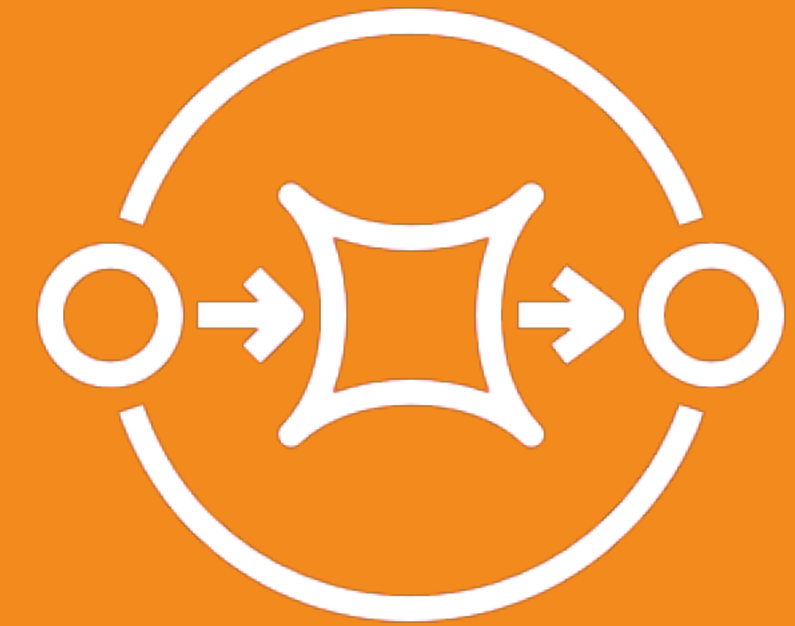






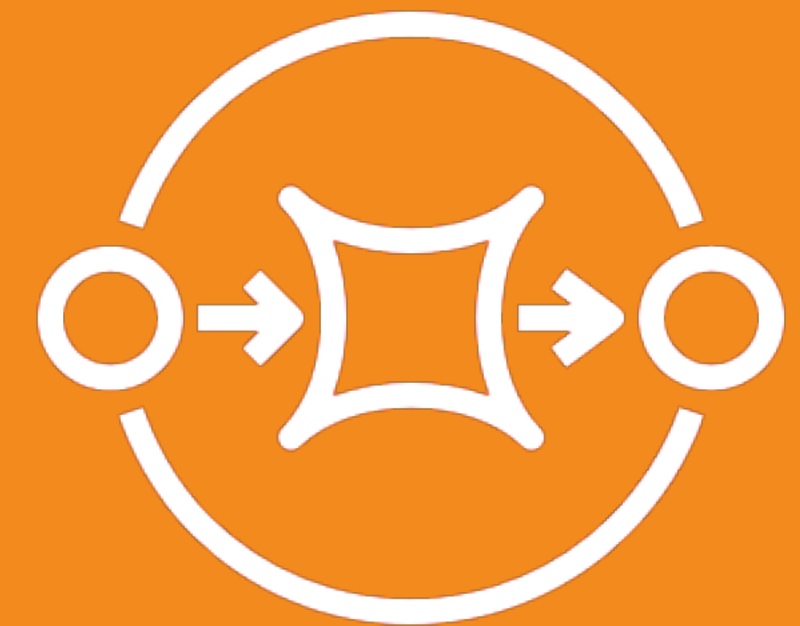
# SQS queues can:

- Perform content-based deduplication
- Retry messages on failed processing
- Specify DLQ on completely failed processing

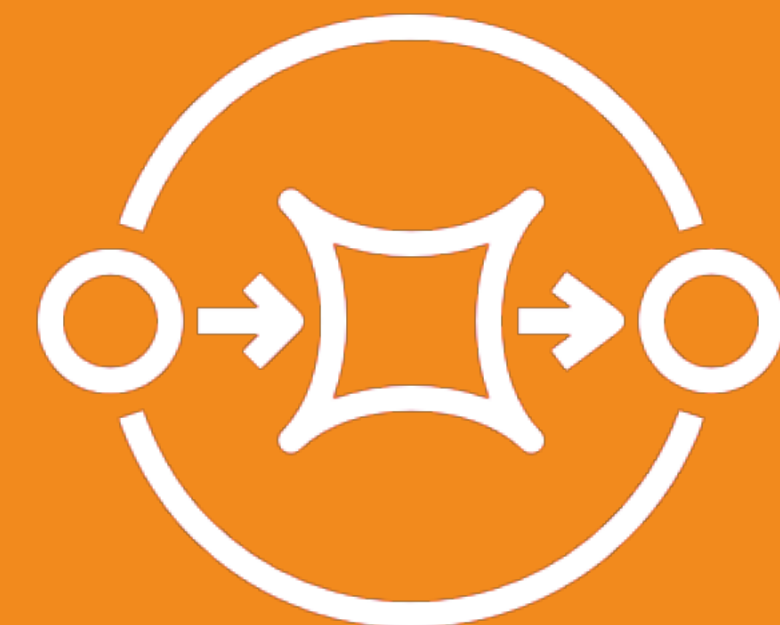




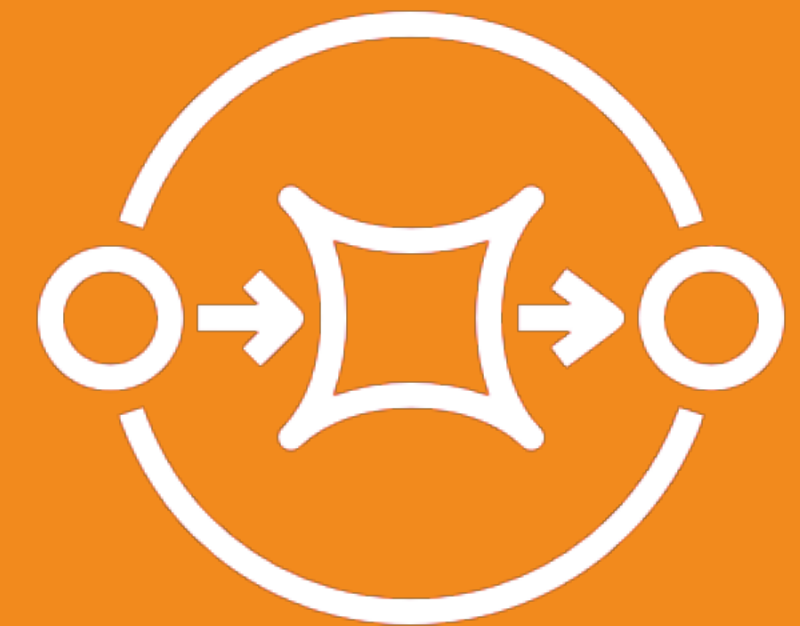
SQS != ordered processing



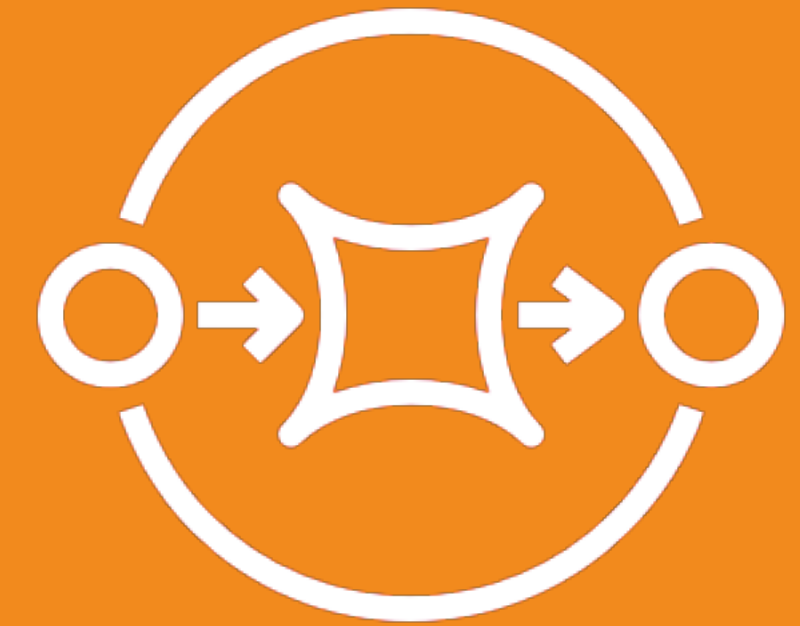
SQS != only-once delivery



SQS requires idempotency



# FIFO queues for order



```
sqsService= getCloudService(awsCredentials, {"serviceName" : "SQS"});
```

```
msgBody = {"customer" : 123, "orderId" : 456, "amount" : "78.90"};
```

```
message = {"messageBody" : msgBody};
```

```
myQueue.sendMessage(message);
```

[github.com/brianklaas/awsplaybox](https://github.com/brianklaas/awsplaybox)

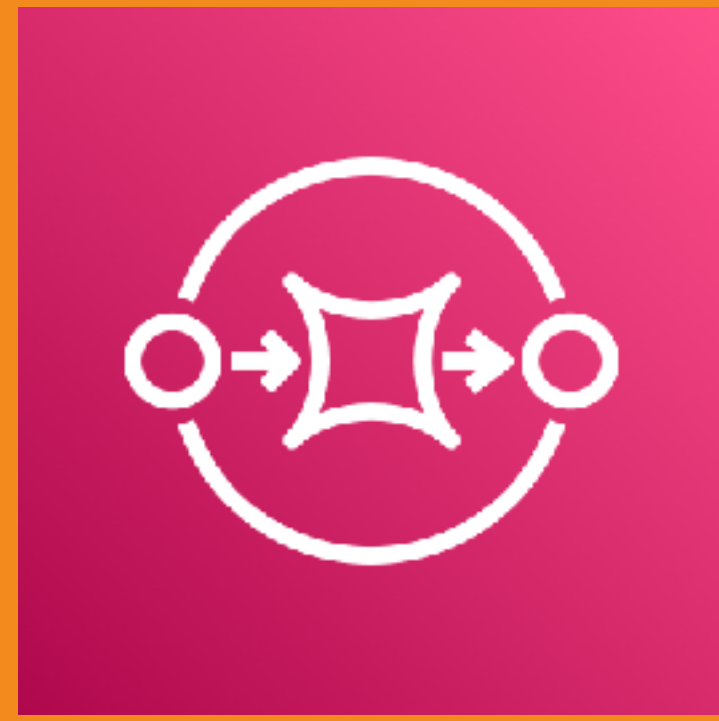


```
sqsService= getCloudService(awsCredentials, {"serviceName" : "SQS"});  
urlOfQueue = sqsService.GetQueueUrl(nameOfQueue);  
messageInfo = sqsService.receiveMessage(urlOfQueue);  
receiptHandle = messageInfo.messages[1].receiptHandle;  
  
// do work  
  
sqsService.deleteMessage(urlOfQueue,receiptHandle);
```





VS



Message Size Limit:

SNS = 256 KiB

SQS = 256 KiB



Message Persistence:

SNS = No

SQS = Yes

Message Ordering:

SNS = Yes

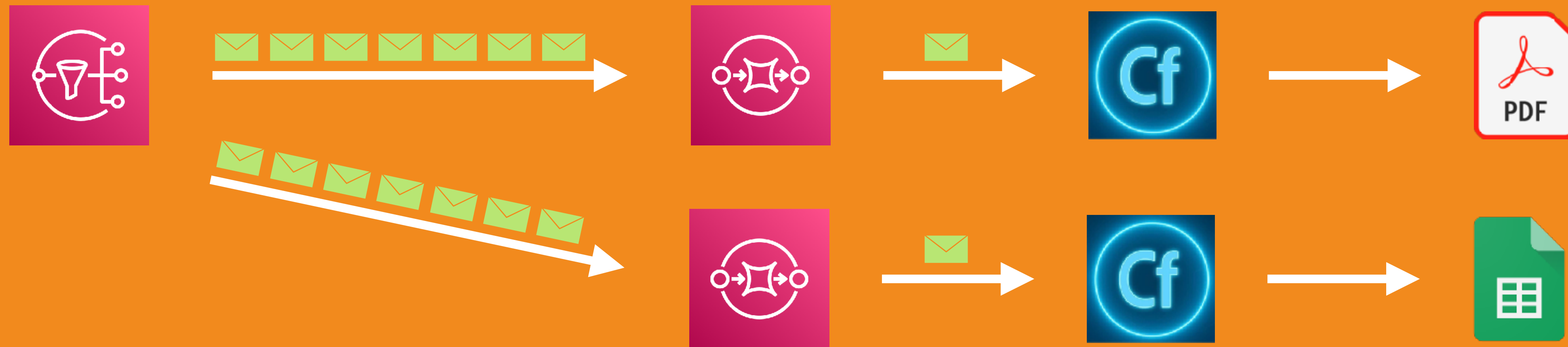
SQS = Yes, when using FIFO queues

Message Filtering:

SNS = Yes

SQS = No

# Common Pattern: SNS in front of SQS



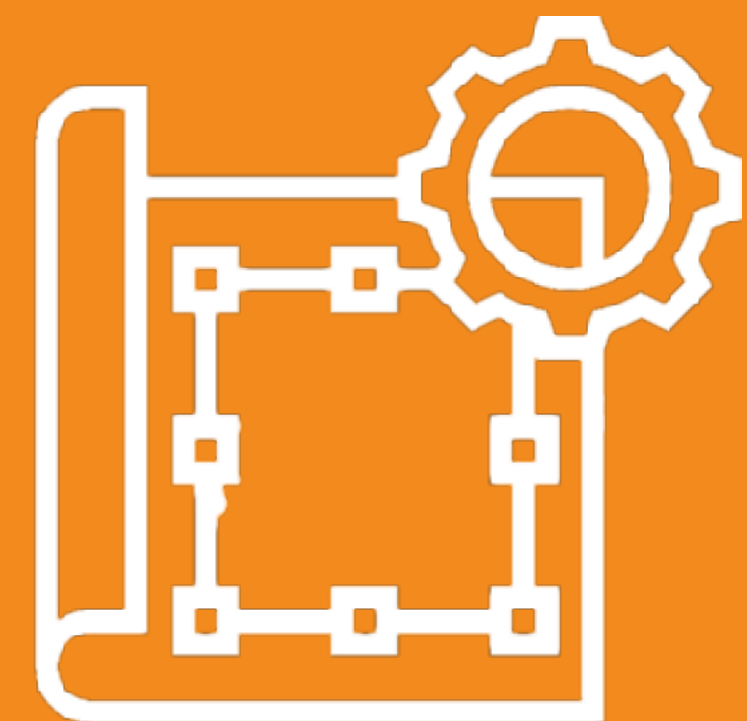
Use cases:

SNS = Fan-out to multiple recipients

SQS = Queuing up work by processors

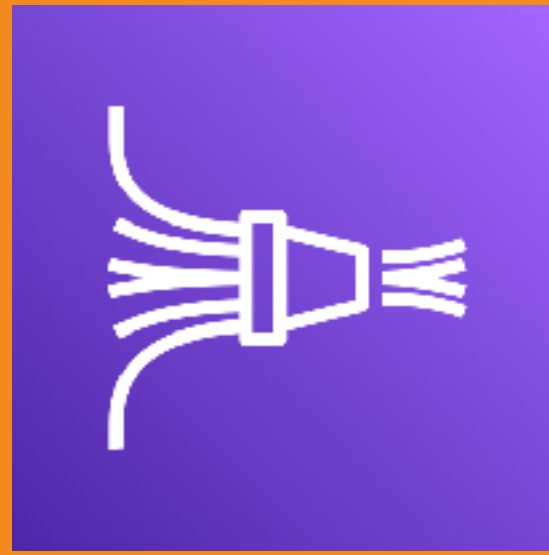
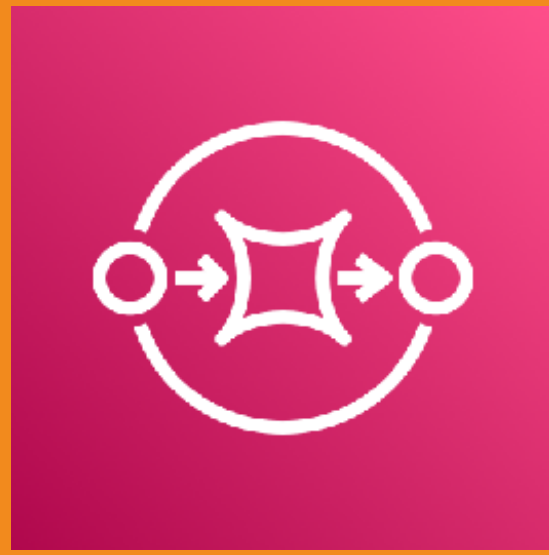
SNS + SQS allows you to scale and parallelize work safely and durably.

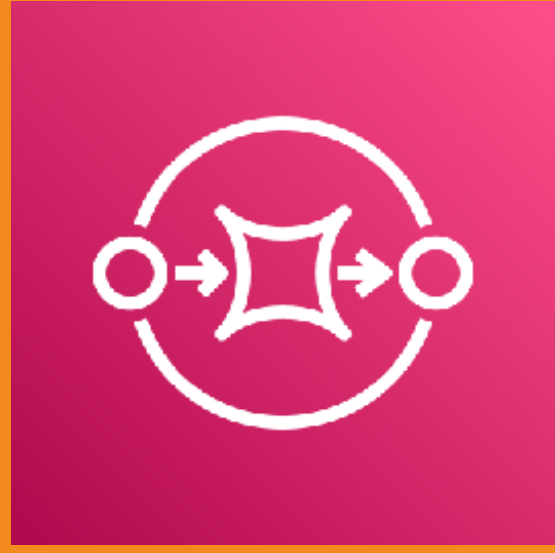
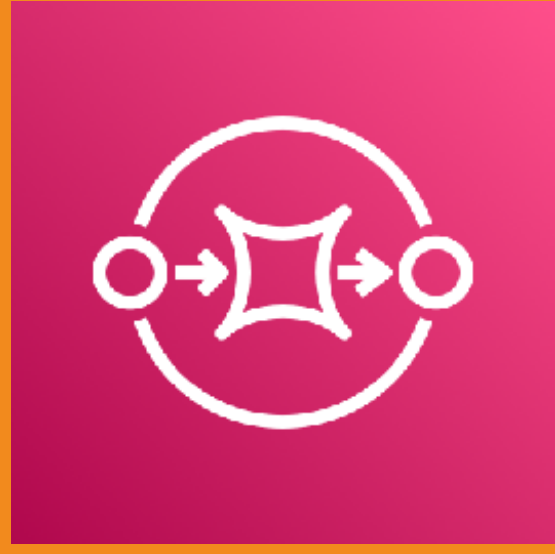
# A Real-World Example









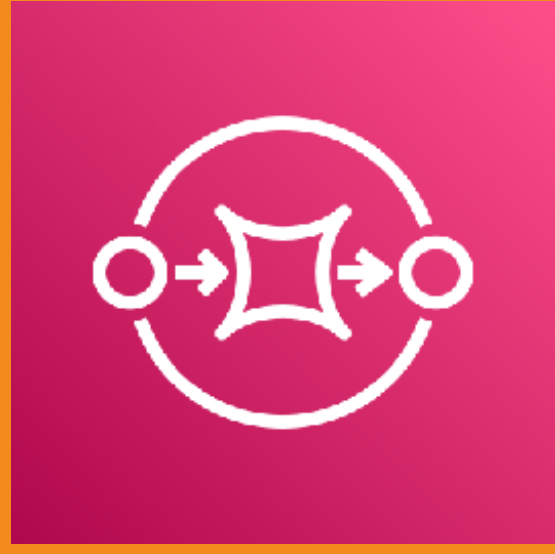


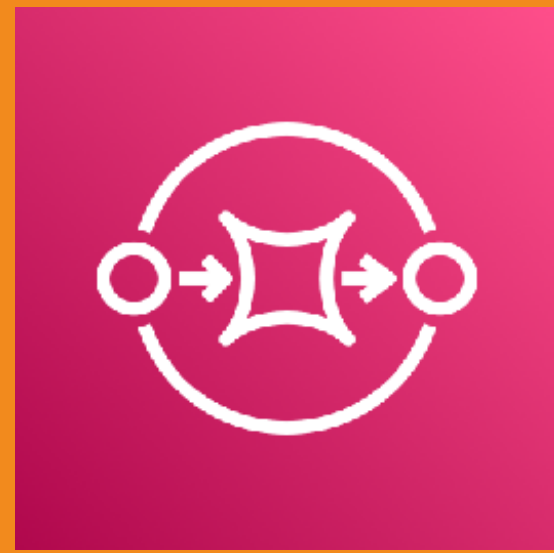


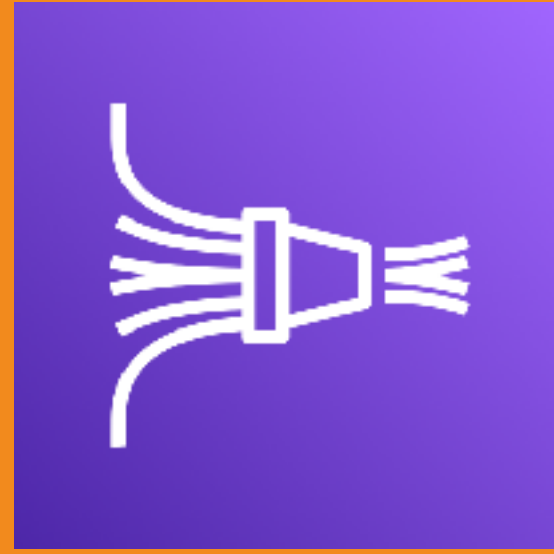
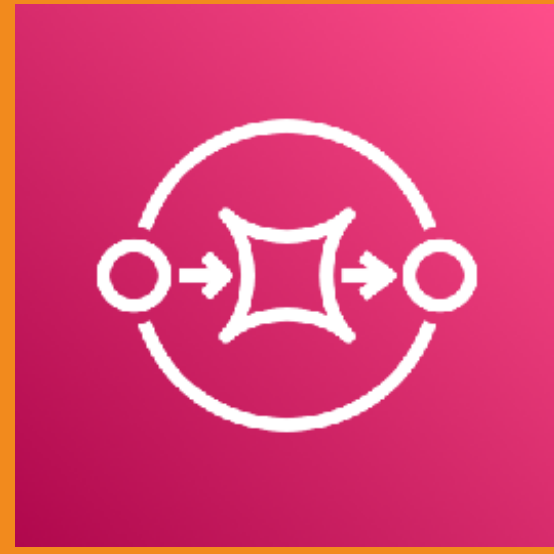
```
snsService= getCloudService(awsCredentials, {"serviceName" : "SNS"});  
topic = snsService.createTopic(topicName);  
msgBody = {"customer": 123, "orderId": 456, "amount": "78.90"};  
topic.publish(msgBody);
```

[github.com/brianklaas/awsplaybox](https://github.com/brianklaas/awsplaybox)











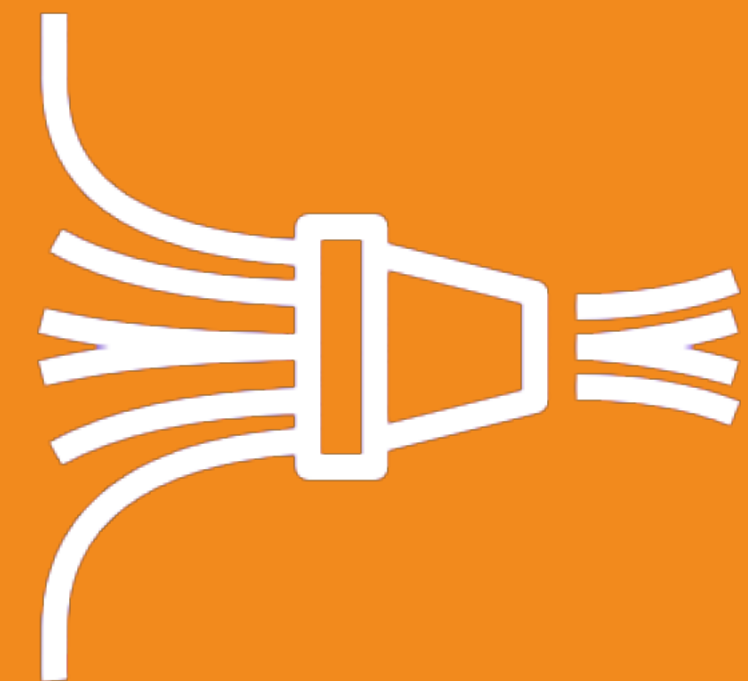
# Kinesis Data Firehose:

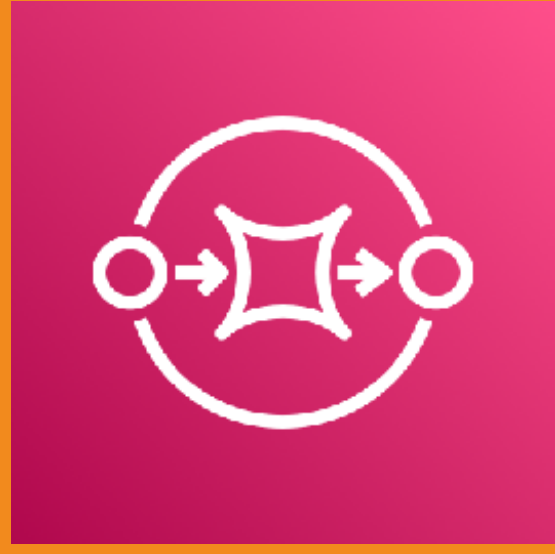
Can ingest up to 500,000 records/second

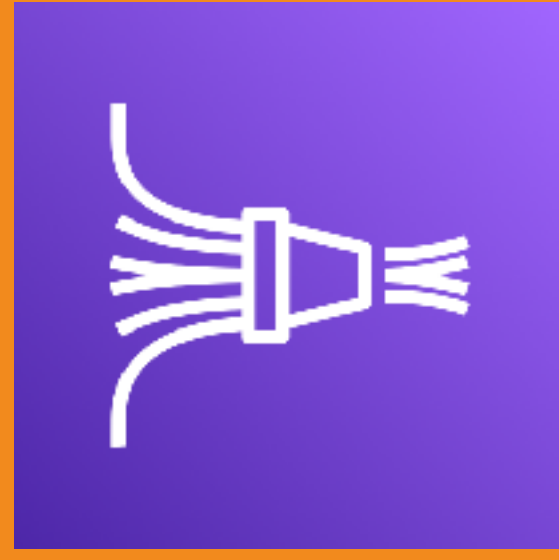
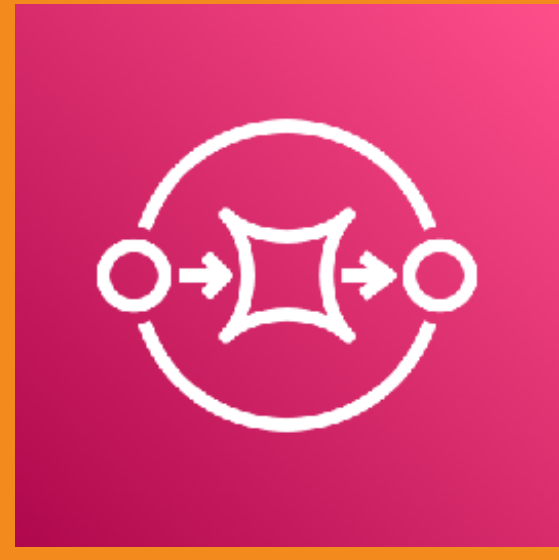
Stores records for up to 7 days

Manages writing to the destination

Can be queried in real-time with Kinesis Data Analytics







# Amazon Athena:

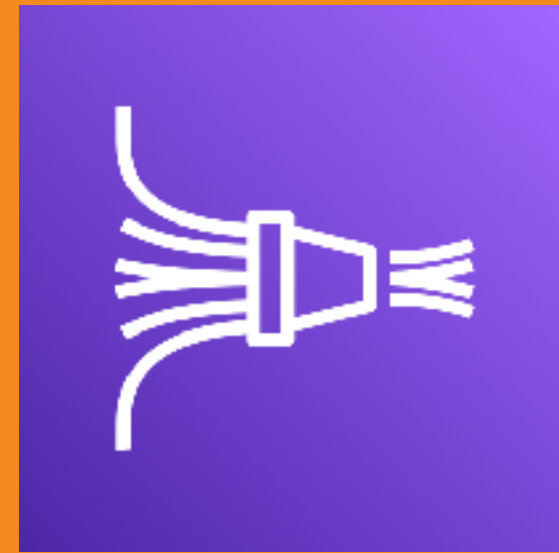
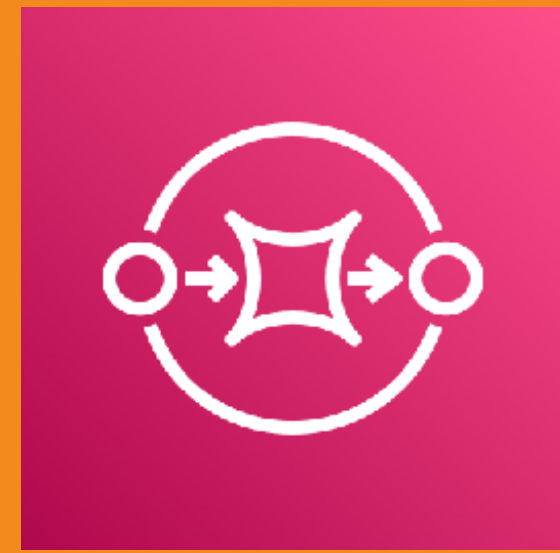
Query files in S3 with SQL

Store in JSON, CSV, or **Parquet** formats

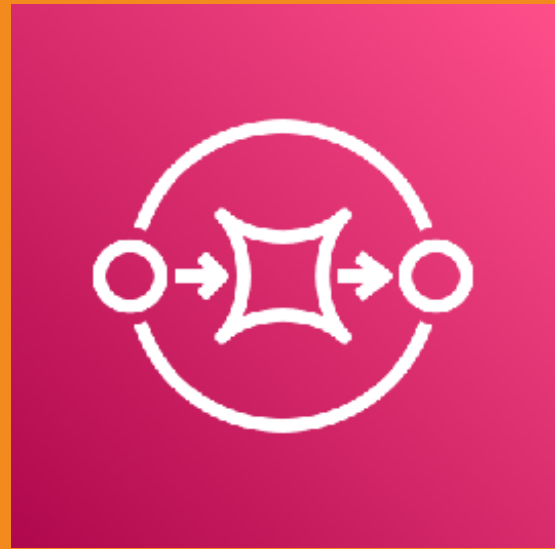
Ad-hoc or repeated queries

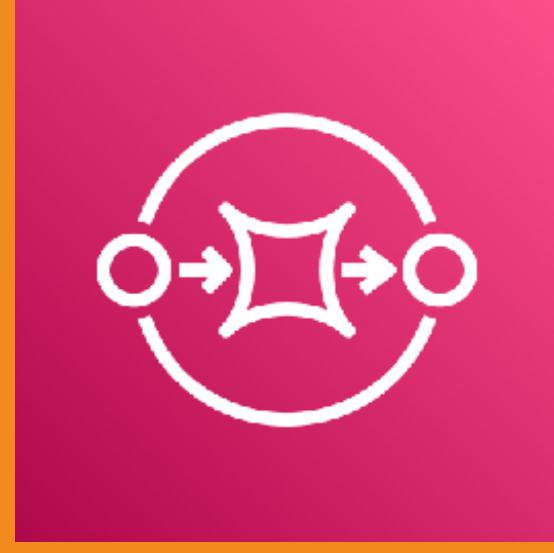
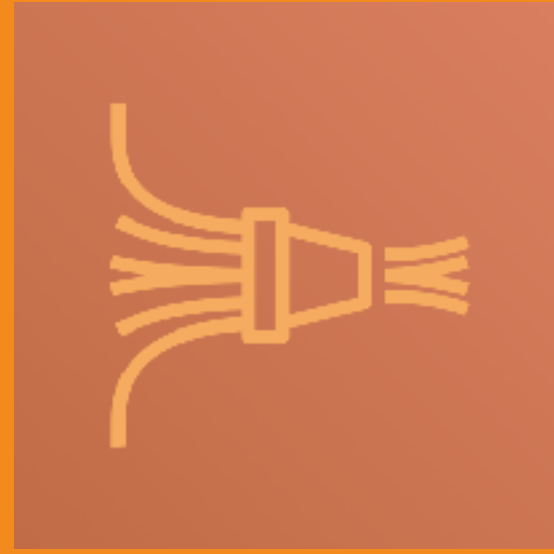
Pay based on amount of data scanned



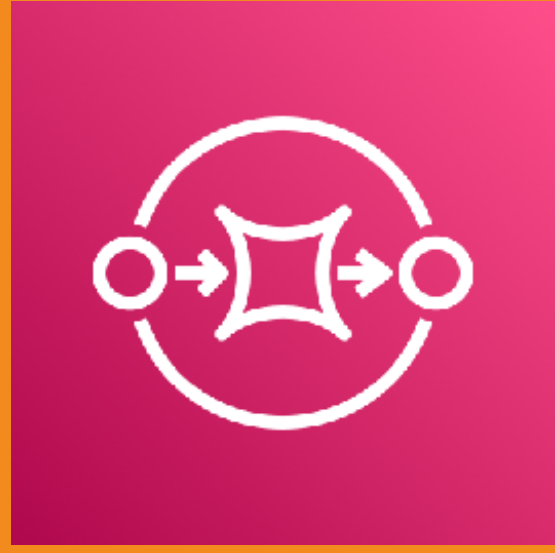
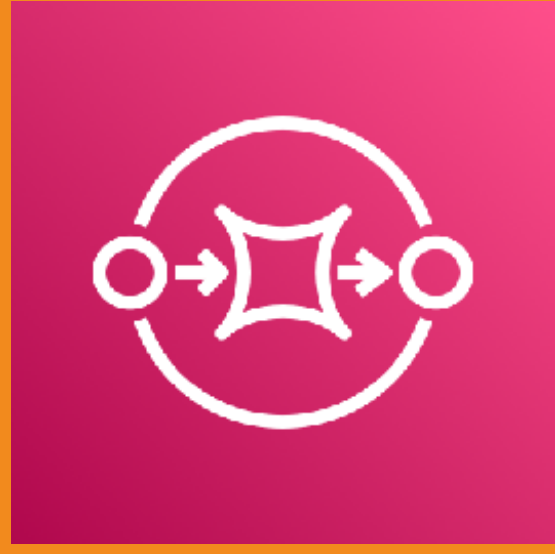








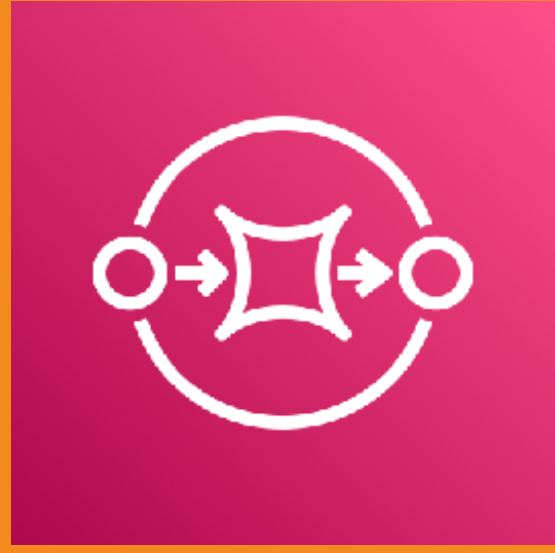
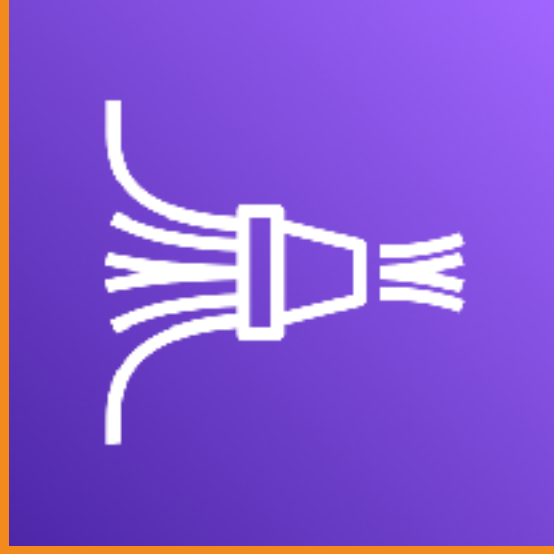
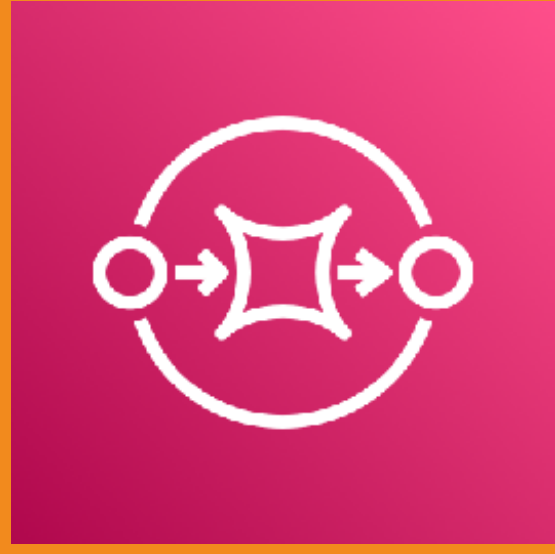




So how much does this cost?

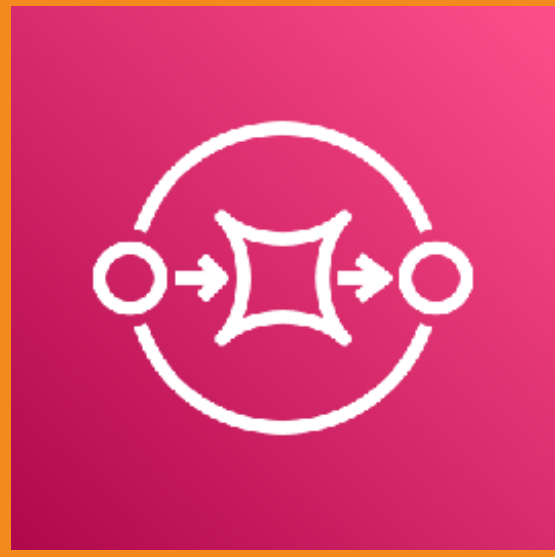
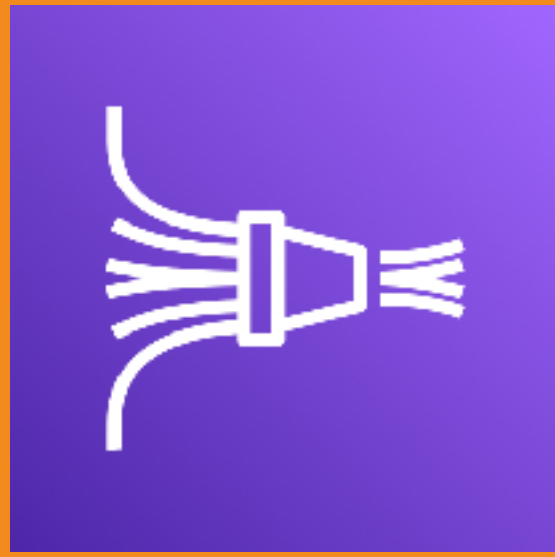
40 million requests = \$30/month





Example





# Safe experimentation!

**Go Do!**

Brian Klaas

@brian\_klaas

Blog: [brianklaas.net](http://brianklaas.net)

[github.com/brianklaas/awsplaybox](https://github.com/brianklaas/awsplaybox)