Monitoring & Streams

Teacher Version

Software Architecture

April 4, 2022

Brae Webb

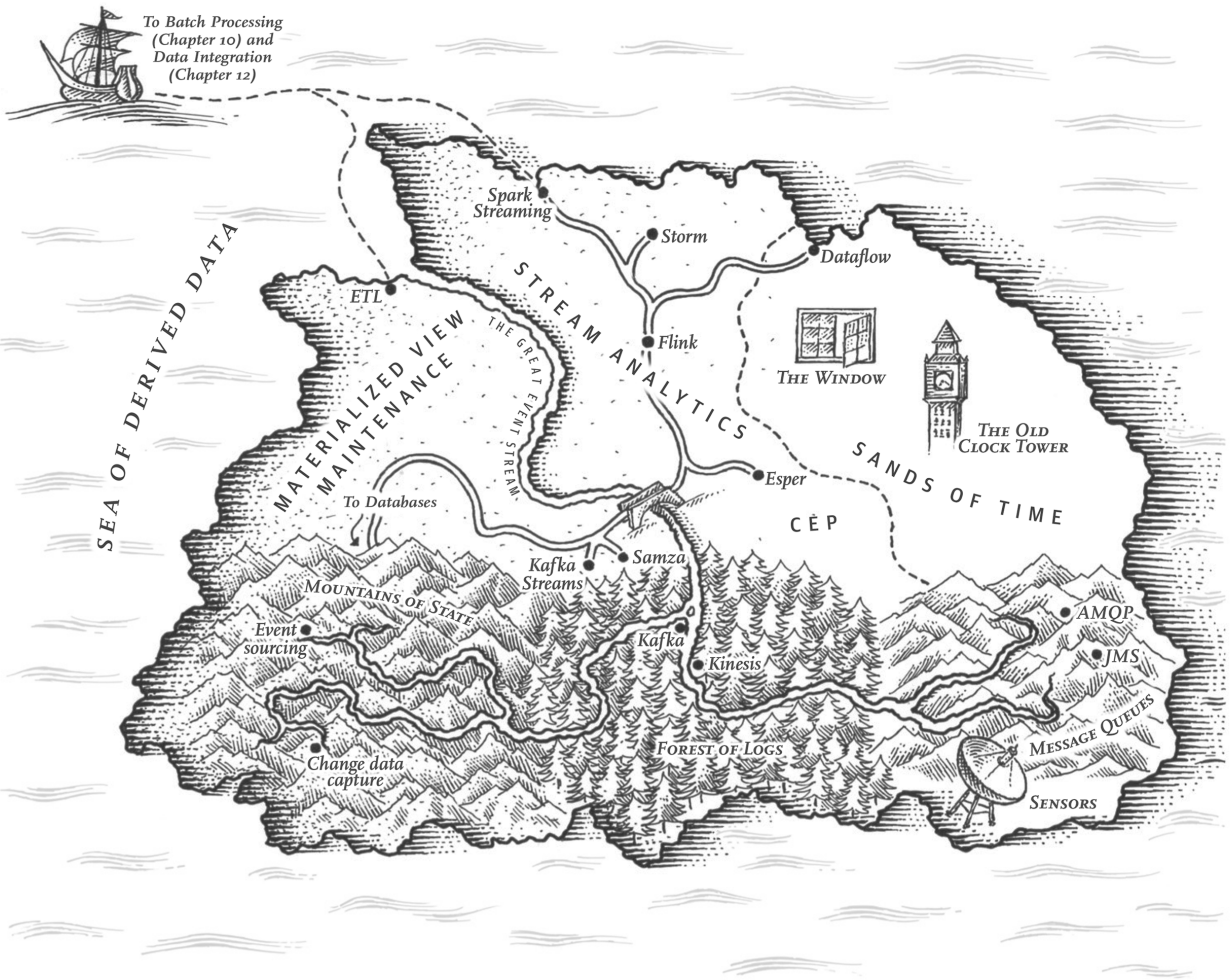


Figure 1: A map of stream processing [1].

1 This Week

This week our goal is to:

- investigate the various options to perform health-checks on services;
- explore the CloudWatch dashboard to monitor our services; and
- run a application which processes requests asynchronously using SQS queues.

2 Health-checks

Health-checks are a way to determine whether or not a service is healthy. They are a core component to developing reliable and scalable systems. Health-checks are utilized by systems that manage collections of service instances, such as Kubernetes, Docker-compose, and of course, AWS Auto-scaling Groups.

In the context of AWS, health-checks serve help load balancers route traffic only to healthy instances. Additionally health-checks can instruct auto-scaling groups to spin down unhealthy instances and replace them with new healthy instances.

3 CloudWatch

CloudWatch is the AWS solution for monitoring services. CloudWatch supports service metrics, logging, and alarms. When working with AWS it is important to understand what CloudWatch can be used for.

> **Info**
>
> For the cloud assignment, we will be be testing that your submission is able to handle an appropriate load. We intend to be give notice of when this testing will occur. The use of CloudWatch to monitor your services and create alarms may give you the ability to manually recover from increased load and perform better in the assignment.

3.1 Metrics

Metrics are at the core of CloudWatch. Metrics track important details about other AWS services often stored as time-series data. For example, EC2 instances store metrics such as CPU and Memory usage. While load balancers record metrics such as the amount of requests and amount of HTTP 200 responses.

Metrics help you to monitor and maintain services running on AWS. When combined with Dashboards they are a powerful way to overview your systems.

3.2 Alarms

CloudWatch alarms can be configured to perform certain actions based on metrics. For example, you may trigger an excessive amount of load balancer requests to increase the size of an auto-scaling group. Or trigger an RDS instance with too little disk space to notify database admin.

```
resource "aws_cloudwatch_metric_alarm" "monitor_alb" {
  comparison_operator = "GreaterThanOrEqualToThreshold"
  metric_name = "RejectedConnectionCount"
  namespace = "AWS/ApplicationELB"
  period = "120"
  statistic = "Sum"
  threshold = "10"

  dimensions = {
    LoadBalancer = aws_lb.balancer.name
  }

  alarm_description = "Increase auto-scaling capacity when 10 requests are rejected
     due to capacity limits"
```

```
  alarm_actions = // increase auto-scaling group capacity.
}

resource "aws_cloudwatch_metric_alarm" "db_free_storage_space_warning" {
  comparison_operator = "LessThanThreshold"
  metric_name = "FreeStorageSpace"
  namespace = "AWS/RDS"
  period = "120"
  statistic = "Sum"
  threshold = 10000000000 // 10 GB
  alarm_description = "Free space dropped below 10 GB"
  alarm_actions = // alert database adims
}
```

The entrypoint for understanding which metrics are available is:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/aws-services-cloudwatch-metr
html

3.3 Logging

CloudWatch offers a logging end-point. It can be utilized to log application data from EC2 instances. For example, if you run an nginx server on an EC2 instance, you can configure CloudWatch to process and store those logs. This feature can be quite helpful and can be extended to the log files of applications you build.

In addition to being processing EC2 log files, CloudWatch comes with built-in logging for some services. For Lambda services it will log print statements or errors. For application load balancer, it will log detailed request information. These logs are incredibly helpful for debugging.

4 Streams

For the teacher

As with last week, spend approximately the first 30 minutes exploring concepts.

4.1 Overview

TODO: Talk about the advantages that streams give, including fan out, fan in, back pressure, isolation

4.2 Data Model

TODO: change to dot points

4.2.1 SPSC

4.2.2 SPMC

4.2.3 MPSC

4.2.4 MPMC

4.3 Technologies

> **Info**
>
> Below we only discuss external stream technologies but it is common to have streams within an application where concurrency is needed. Instead of dealing with concurrency using locking based methods like in CSSE2310, we can use streams to send data to the consumers that need to handle single writer usecases.

4.3.1 AWS SQS

4.3.2 AWS SNS

4.3.3 AWS MQ / Apache ActiveMQ / RabbitMQ

> **Aside**
>
> Not available in the lab environments

4.3.4 AWS MSK (Managed Streaming for Apache Kafka)

> **Aside**
>
> Not available in the lab environments

4.3.5 Redis

> **Aside**
>
> Not available in the lab environments

5 Talking to the Simple Queue Service (SQS)

AWS provides the Simple Queue Service, SQS, which offers a simple and fully managed message queue service. There are two flavours of SQS to be aware of; the standard message queue, and the FIFO message queue. Standard message queues allow for greater scalability by providing higher through-put. However, standard message queues in SQS are not exactly queues, messages are not first in first out, they are best-effort ordered. The second flavour, FIFO message queues, guarantees that messages are First in First Out.

5.1 Terraform

Warning

For terminal examples in this section, lines that begin with a $ indicate a line which you should type while the other lines are example output that you should expect. Not all of the output is captured in the examples to save on space.

Today we will be creating and experimenting with the two queue flavours of AWS SQS. We provide the Terraform to create these services below:

Info

If you have forgotten how to get started you will need to run the following commands in a local terminal.

```
$ terraform init
...
$ terraform plan
...
$ terraform apply
...
```

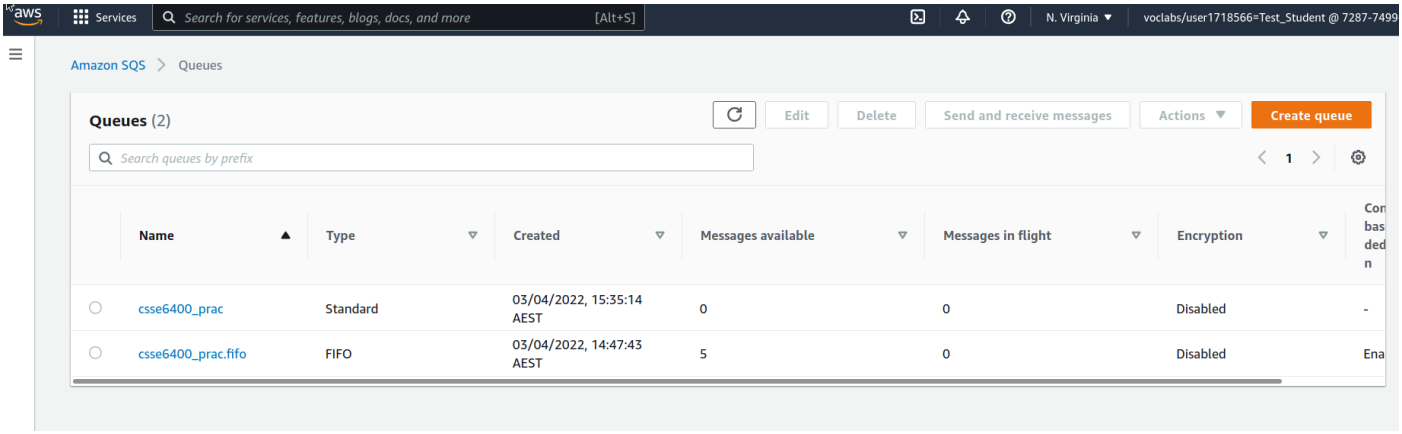
For the teacher

In the below terraform it is pretty simple and relies a lot on defaults. Though the FIFO name needs to have .fifo at the end, make sure to mention this.

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  shared_credentials_file = "./credentials"
}

resource "aws_sqs_queue" "our_first_mailbox" {
  name = "csse6400_prac"
}

resource "aws_sqs_queue" "our_first_fifo" {
  name = "csse6400_prac.fifo"
  fifo_queue = true
```

```
  content_based_deduplication = true
}

output "mailbox" {
    value = aws_sqs_queue.our_first_mailbox.arn
}

output "fifo" {
    value = aws_sqs_queue.our_first_fifo.arn
}
```

Now that we have provisioned the queues we can have a look at them in the AWS Console. In the main AWS dashboard you can search for"SQS" to find these queues. You should reach a page like this:



Like the EC2 and RDS dashboards before it, we can browse the queues settings and the metrics that are gathered about them.

For the teacher

Show the students the Monitoring Panel and Access Policy panel in particular.

- Monitoring: Messages Sent/Received
- Monitoring: Empty Receives - This is the receiver getting nothing from polling. Costs $ in the real world.
- Policy: Not covering today but SQS is a public service, it is protected via AWS credentials. In real use cases you need to configure a Access Policy.

5.2 Queue Cli

We have provided a small docker container for you to use with your queues to see the differnce between the implementations. First we must get our AWS credentials and prep our environment.

With our learner lab grab the AWS credentials but instead of creating a credentials file we will be using environment variables. Make a folder for the practicle.

```
$ mkdir queues && cd queues
```

Now we need to create an environment file for our docker container to read so that it can access AWS. Create a ".env" file in the current directory and edit the contents so that it looks similar to the below: The AWS keys will be from the credentials shown in your lab environment.

```
TERM=xterm-256color
AWS_ACCESS_KEY_ID=...
AWS_SECRET_ACCESS_KEY=...
AWS_SESSION_TOKEN=...
```

```
$ docker run --rm -it --env-file .env ghcr.io/csse6400/queue:main --name "test" --
    client-name "Client 1"


  ----------
 |   \XX/   |
 | T. \/ .T |      University of Queensland
 | XX:  :XX |          Faculty of EAIT
 T L' /\ 'J T
  \  /XX\  /          CSSE6400 Queue Prac
@\_ '____' _/@        csse6400.uqcloud.net
\_X\_ __ _/X_/
 \=/\----/\=/



Unable to find a Queue by this name test
```

If your program shows the above then your ready to head to the next section :).

5.3 SQS Standard

TODO: Talk about what a SQS Standard is in terms of the info above. Dont forget to mention that they arnt a QUEUE.

5.4 SQS FIFO

TODO: Talk about how these are classicalt queues in the sense that First-In First-Out is the correct term for a QUEUE.

References

[1] M. Kleppmann, *Designing Data-Intensive Applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media, Inc., March 2017.