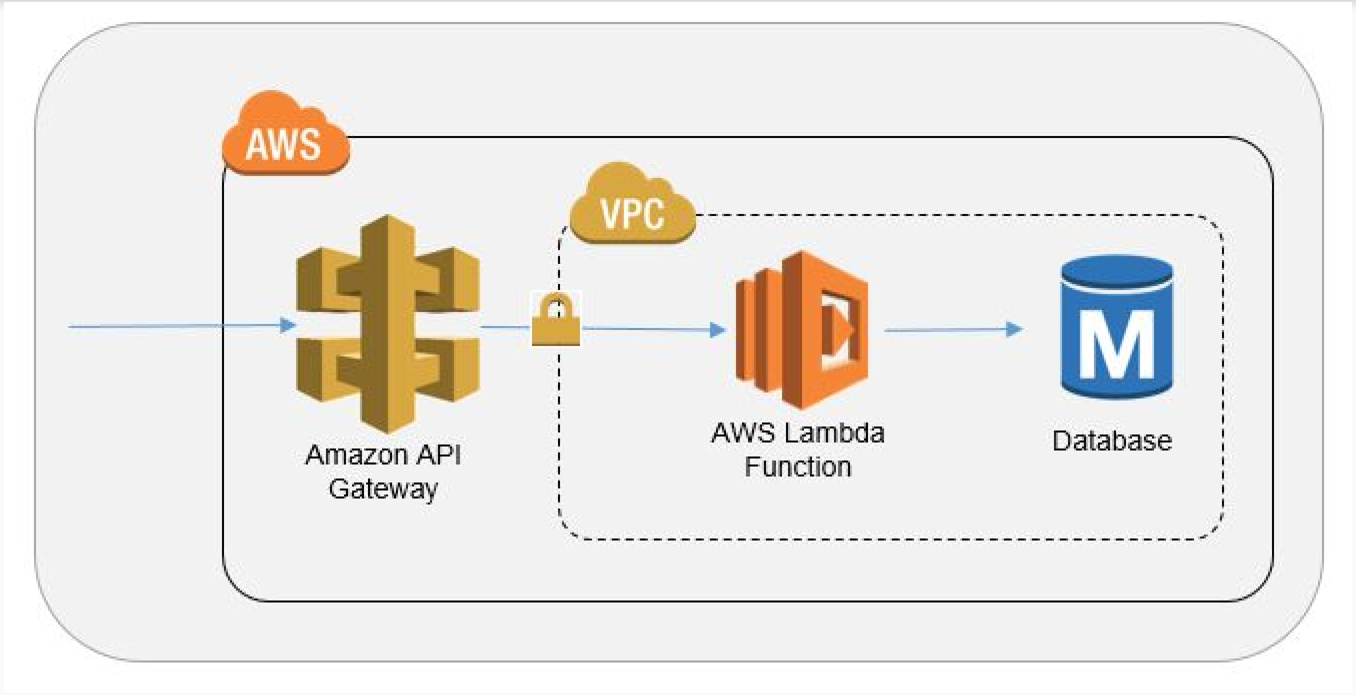
BSDS Assignment 3

Alec, Yang

Let’s go **Serverless!**

Overall Architecture



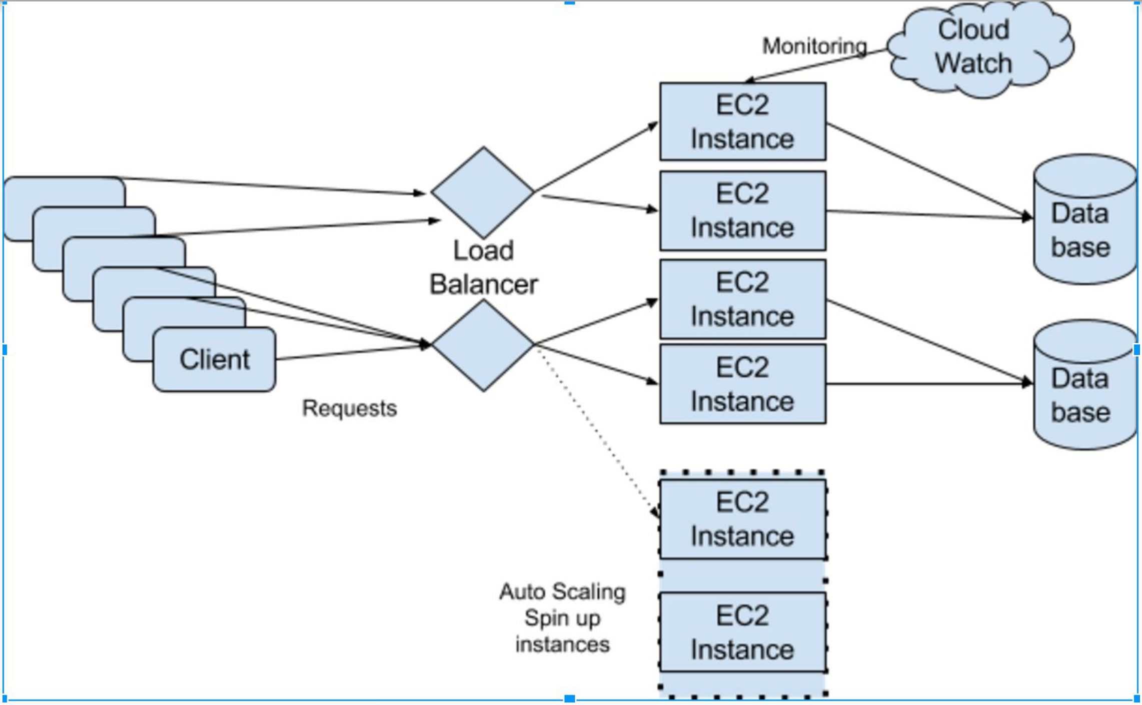
Major Components

1. API Gateway: for creating and managing APIs;
2. Lambda: executing backend business logic;
3. DynamoDB: key-value data storage

NO load balancer, NO auto-scaling, even NO EC2 instances!

- It's all taken care of under the hood!

Compare to traditional design



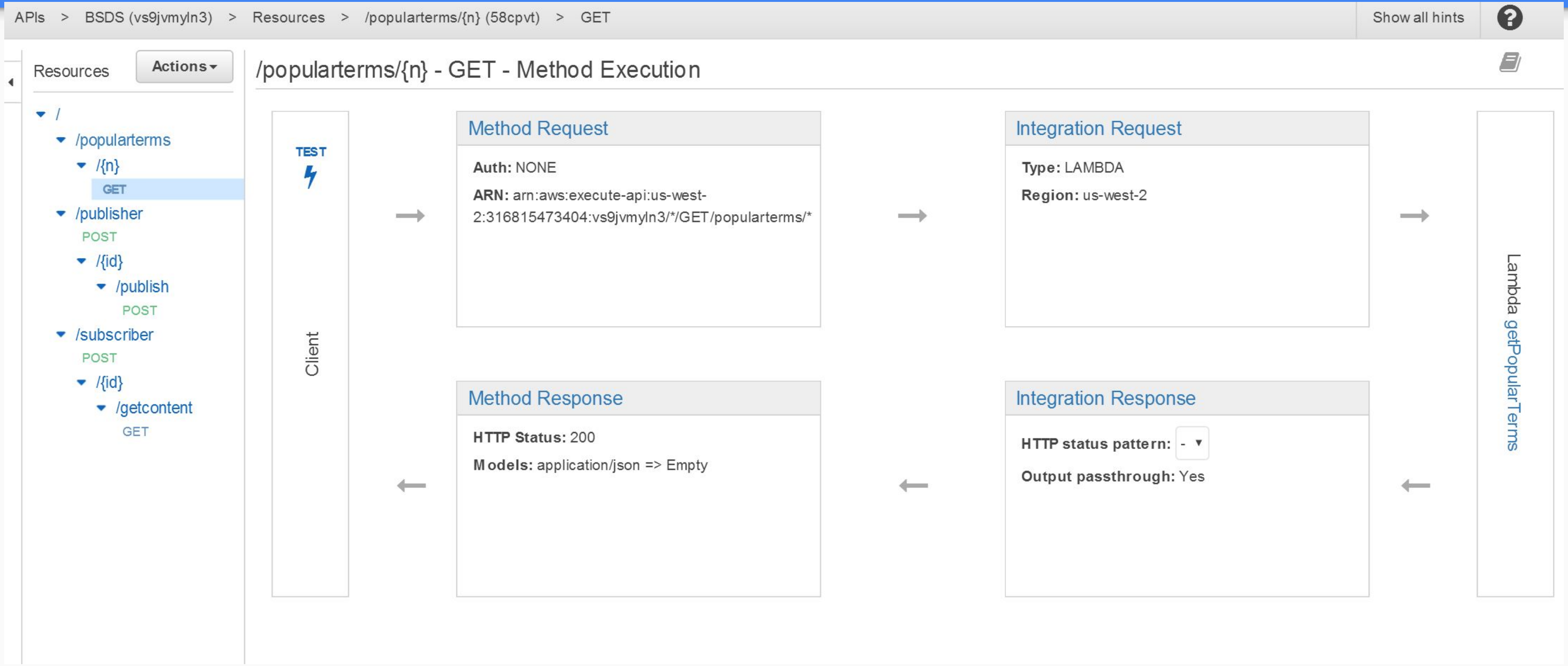
Advantages of using Serverless vs traditional approach

1. Amazon API Gateway’s integration with AWS Lambda enables user defined code functions to be triggered directly via a user-defined HTTPS request.
2. Regardless of the request volume required, both the API Gateway and Lambda will scale **automatically** to support exactly the needs of your application.

Amazon API Gateway

- “A fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.”
- Provides a HTTPS API front end for clients
- Clients integrate with the APIs using standard HTTPS requests

Amazon API Gateway



APIs > BSDS (vs9jvmyln3) > Resources > /popularterms/{n} (58cpvt) > GET

Show all hints

Resources

- /
  - /popularterms
    - /{n}
      - GET
  - /publisher
    - POST
    - /{id}
      - /publish
        - POST
  - /subscriber
    - POST
    - /{id}
      - /getcontent
        - GET

/popularterms/{n} - GET - Method Execution

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-west-2:316815473404:vs9jvmyln3/*/GET/popularterms/*

Integration Request

Type: LAMBDA

Region: us-west-2

Method Response

HTTP Status: 200

Models: application/json => Empty

Integration Response

HTTP status pattern: -

Output passthrough: Yes

AWS Lambda

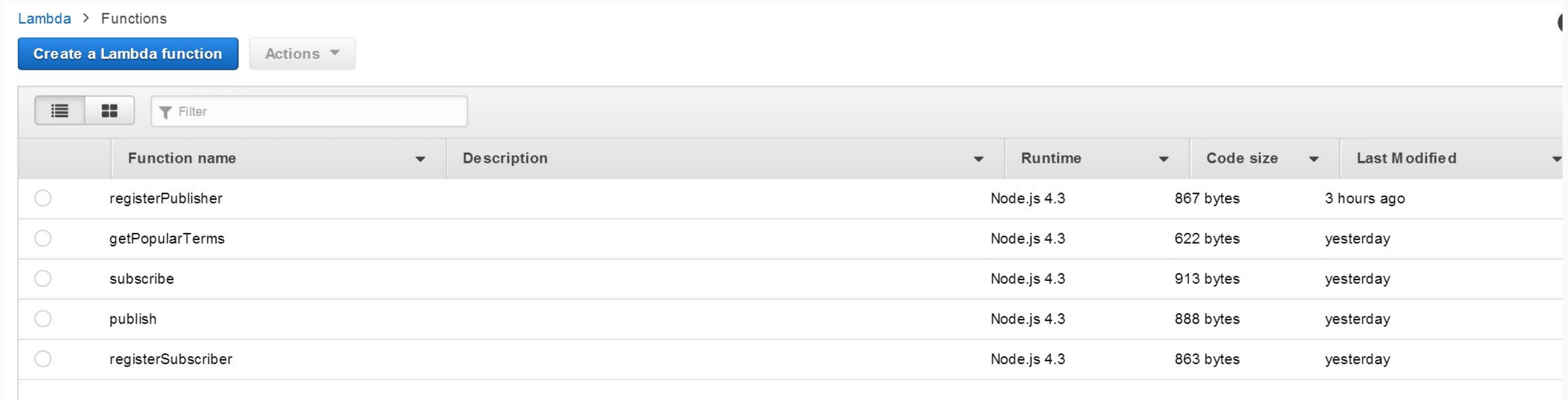
- Run code without provisioning or managing servers
- Can be triggered directly by an HTTPS request
- Execute the backend logic behind the APIs
- Lambda manages the compute fleet that offers a balance of memory, CPU, network, and other resources.

AWS Lambda - Scalability

How does Lambda scale?

- Launch functions in a container (execution environment)
- The container will be maintained for some time and reused on another function invocation
- Concurrent executions: each published event is a unit of work
- Execution environment can be configured by environment variables

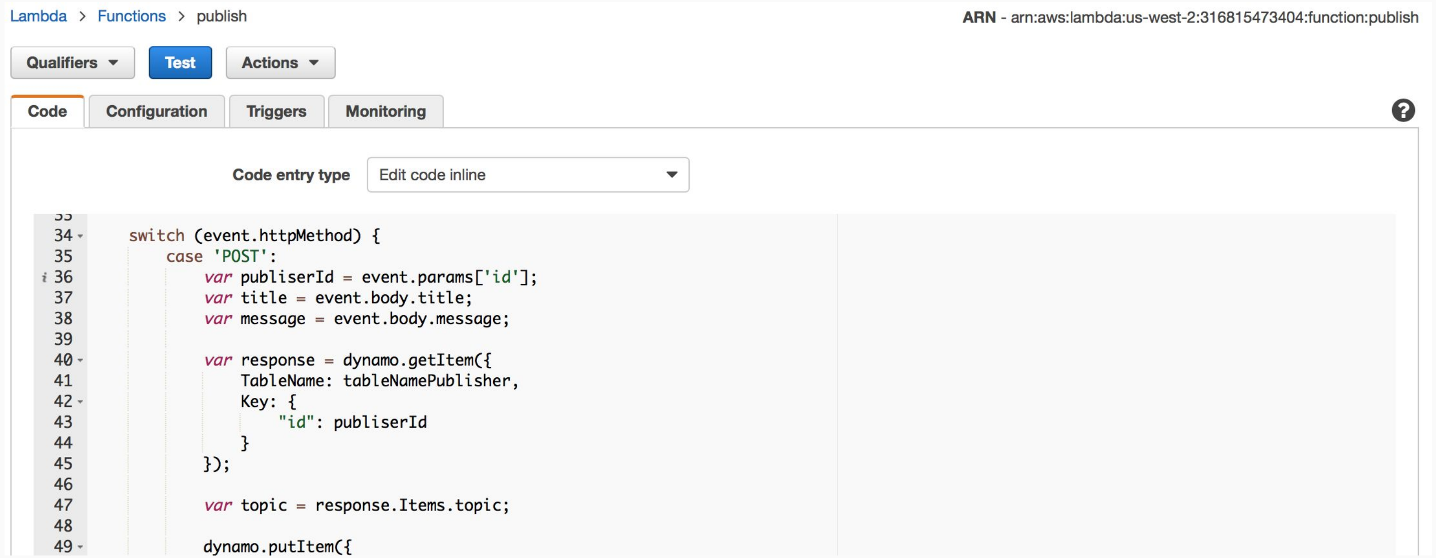
AWS Lambda - Functions



	Function name	Description	Runtime	Code size	Last Modified
○	registerPublisher		Node.js 4.3	867 bytes	3 hours ago
○	getPopularTerms		Node.js 4.3	622 bytes	yesterday
○	subscribe		Node.js 4.3	913 bytes	yesterday
○	publish		Node.js 4.3	888 bytes	yesterday
○	registerSubscriber		Node.js 4.3	863 bytes	yesterday

One-to-one mapping between API Gateway APIs and Lambda functions

AWS Lambda - Functions



ARN - arn:aws:lambda:us-west-2:316815473404:function:publish

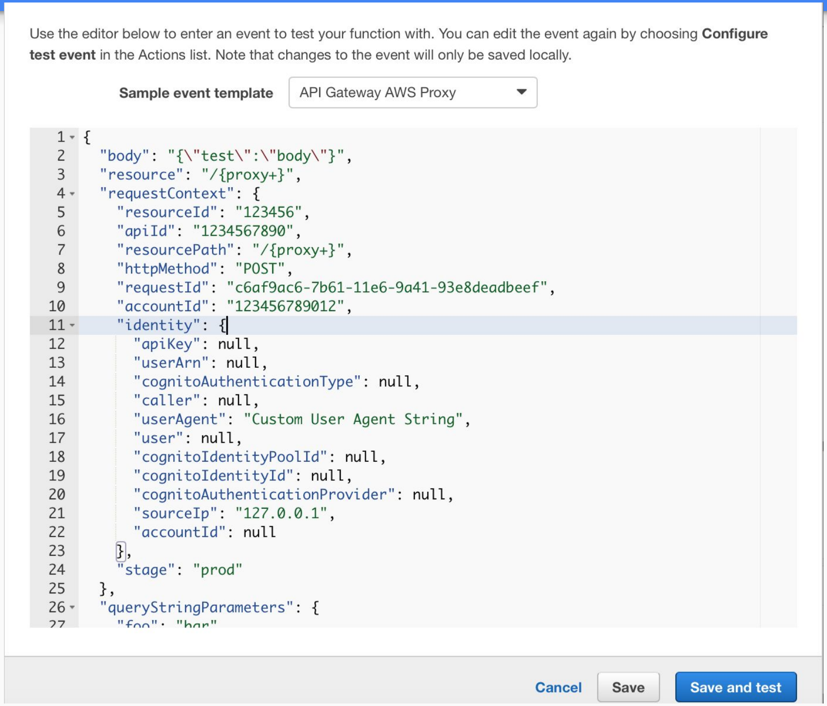
Code entry type Edit code inline

```
    switch (event.httpMethod) {
        case 'POST':
            var publiserId = event.params['id'];
            var title = event.body.title;
            var message = event.body.message;

            var response = dynamo.getItem({
                TableName: tableNamePublisher,
                Key: {
                    "id": publiserId
                }
            });

            var topic = response.Items.topic;

            dynamo.putItem({
```

Create and modify event handlers with the built-in editor.

AWS Lambda - Functions

Test event handlers with the built-in test tool.

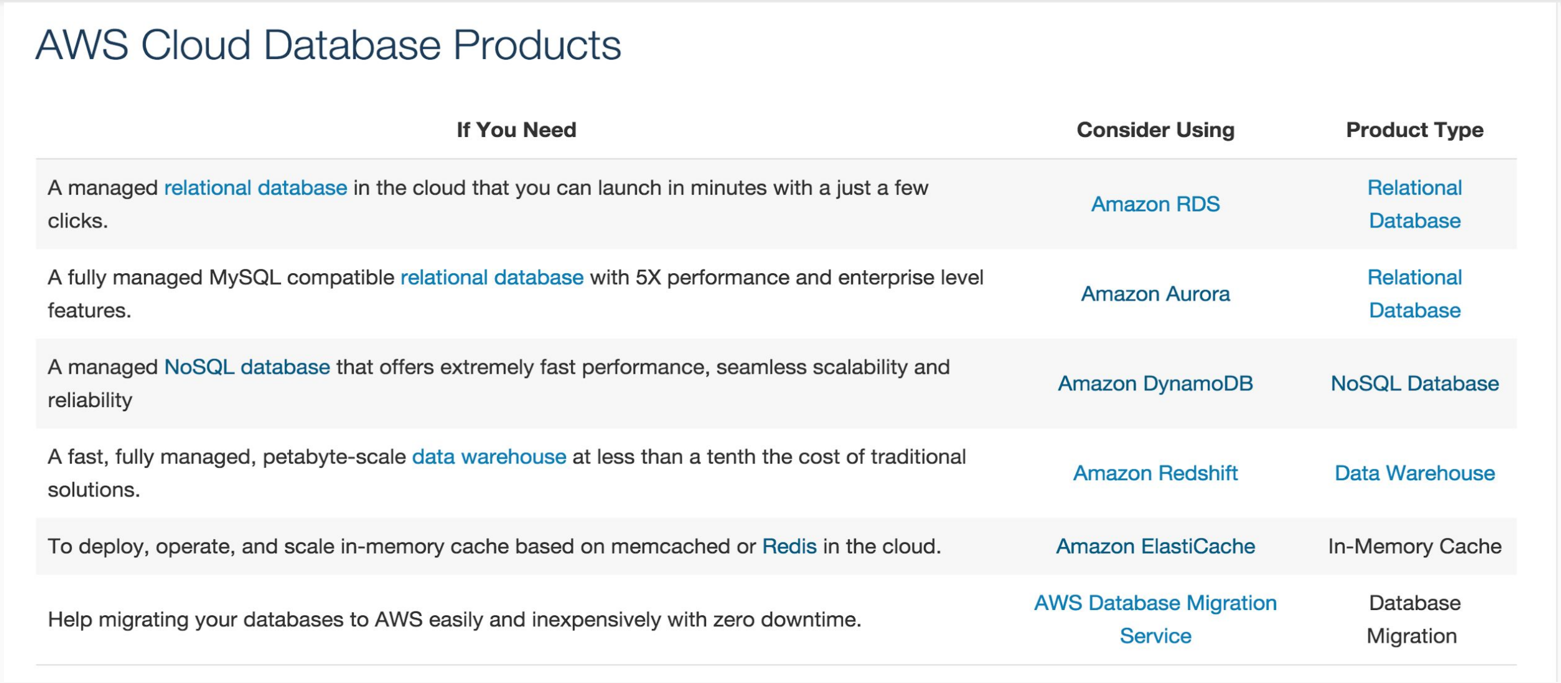


Use the editor below to enter an event to test your function with. You can edit the event again by choosing **Configure test event** in the Actions list. Note that changes to the event will only be saved locally.

Sample event template API Gateway AWS Proxy

```
{
  "body": "{\"test\":\"body\"}",
  "resource": "/{proxy+}",
  "requestContext": {
    "resourceId": "123456",
    "apiId": "1234567890",
    "resourcePath": "/{proxy+}",
    "httpMethod": "POST",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "accountId": "123456789012",
    "identity": {
      "apiKey": null,
      "userArn": null,
      "cognitoAuthenticationType": null,
      "caller": null,
      "userAgent": "Custom User Agent String",
      "user": null,
      "cognitoIdentityPoolId": null,
      "cognitoIdentityId": null,
      "cognitoAuthenticationProvider": null,
      "sourceIp": "127.0.0.1",
      "accountId": null
    },
    "stage": "prod"
  },
  "queryStringParameters": {
    "foo": "bar"
```

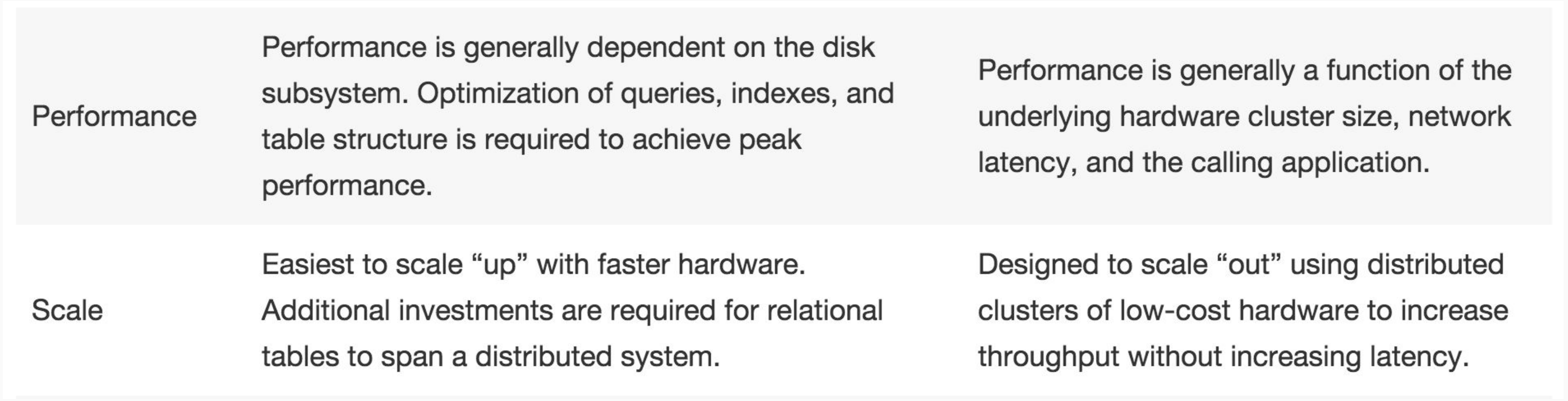
DB options



AWS Cloud Database Products

If You Need	Consider Using	Product Type
A managed relational database in the cloud that you can launch in minutes with a just a few clicks.	Amazon RDS	Relational Database
A fully managed MySQL compatible relational database with 5X performance and enterprise level features.	Amazon Aurora	Relational Database
A managed NoSQL database that offers extremely fast performance, seamless scalability and reliability	Amazon DynamoDB	NoSQL Database
A fast, fully managed, petabyte-scale data warehouse at less than a tenth the cost of traditional solutions.	Amazon Redshift	Data Warehouse
To deploy, operate, and scale in-memory cache based on memcached or Redis in the cloud.	Amazon ElastiCache	In-Memory Cache
Help migrating your databases to AWS easily and inexpensively with zero downtime.	AWS Database Migration Service	Database Migration

Relational vs NoSQL

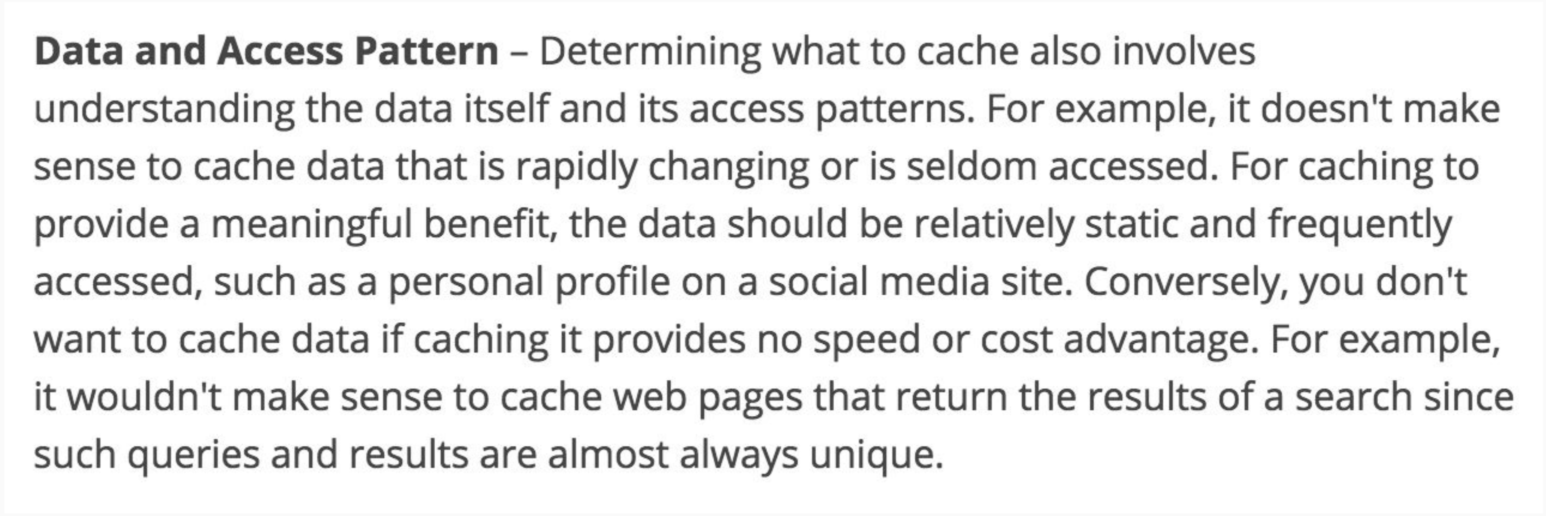


	Relational	NoSQL
Performance	Performance is generally dependent on the disk subsystem. Optimization of queries, indexes, and table structure is required to achieve peak performance.	Performance is generally a function of the underlying hardware cluster size, network latency, and the calling application.
Scale	Easiest to scale “up” with faster hardware. Additional investments are required for relational tables to span a distributed system.	Designed to scale “out” using distributed clusters of low-cost hardware to increase throughput without increasing latency.

Relational vs NoSQL

- We are running extremely simple queries
- We are focusing on scalability which means horizontal scaling vs vertical scaling, SQL is better at vertical, NoSQL is better at horizontal

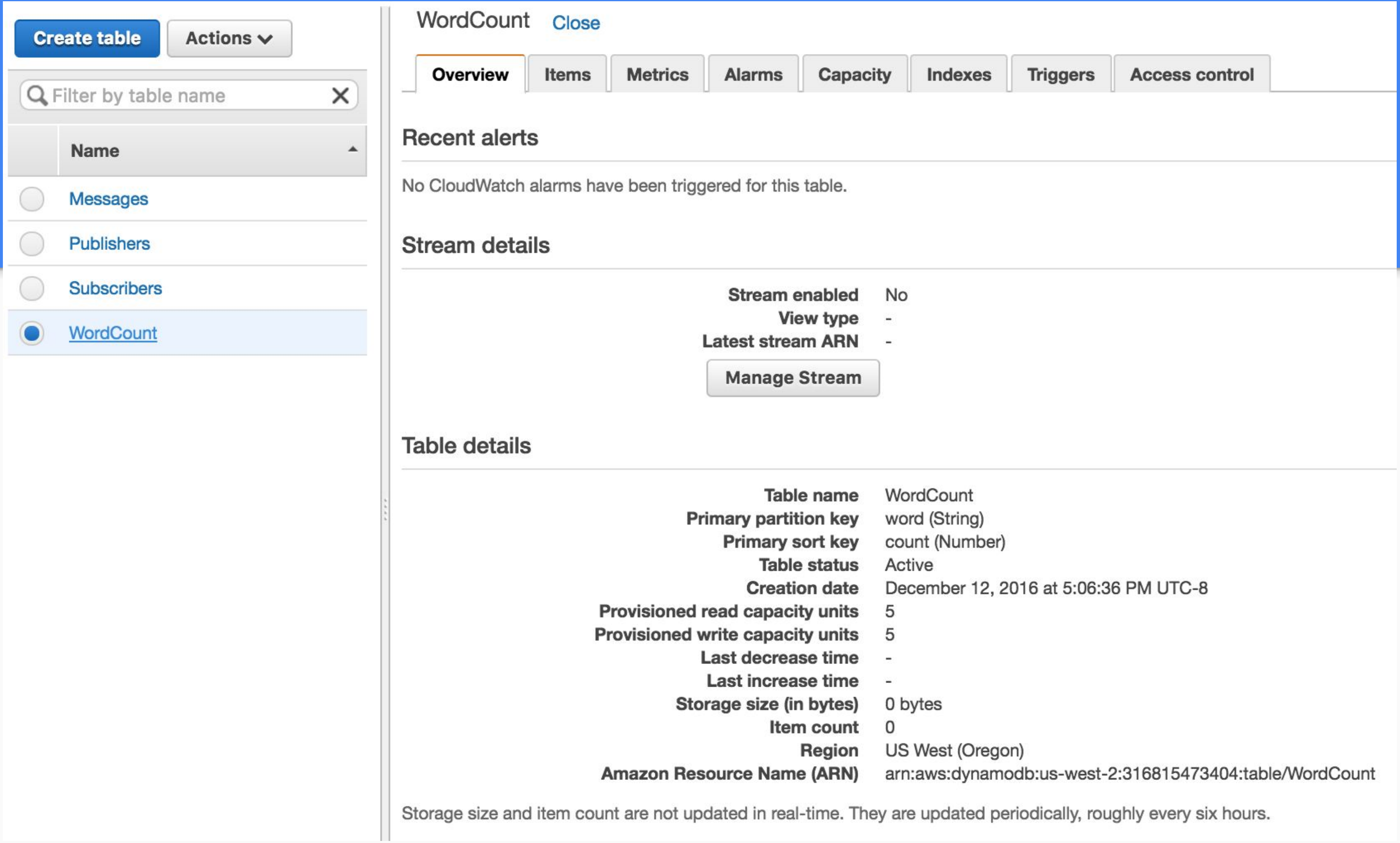
AWS Elasticache



Data and Access Pattern – Determining what to cache also involves understanding the data itself and its access patterns. For example, it doesn't make sense to cache data that is rapidly changing or is seldom accessed. For caching to provide a meaningful benefit, the data should be relatively static and frequently accessed, such as a personal profile on a social media site. Conversely, you don't want to cache data if caching it provides no speed or cost advantage. For example, it wouldn't make sense to cache web pages that return the results of a search since such queries and results are almost always unique.

Amazon DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models.



- Messages
- Publishers
- Subscribers
- ● WordCount

WordCount Close

Recent alerts

No CloudWatch alarms have been triggered for this table.

Stream details

Stream enabled	No
View type	-
Latest stream ARN	-

Table details

Table name	WordCount
Primary partition key	word (String)
Primary sort key	count (Number)
Table status	Active
Creation date	December 12, 2016 at 5:06:36 PM UTC-8
Provisioned read capacity units	5
Provisioned write capacity units	5
Last decrease time	-
Last increase time	-
Storage size (in bytes)	0 bytes
Item count	0
Region	US West (Oregon)
Amazon Resource Name (ARN)	arn:aws:dynamodb:us-west-2:316815473404:table/WordCount

Storage size and item count are not updated in real-time. They are updated periodically, roughly every six hours.

Individual Tables

- Potential savings on Access locks when using a table for a topic
- We thought about using individual tables, but the overhead to create the tables was too great.

Final point

AWS Lambda
Dynamo DB

Reference:

AWS Serverless Multi-tier Architectures:
https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf

We Made the Whole Company "Serverless":
https://blog.cloudsploit.com/we-made-the-whole-company-serverless-5a91c27cd8c4#.j03atfxlq

AWS Documentation:
https://aws.amazon.com/documentation/

Thanks!