**Yoojeong Choi**



**Data Life Cycle on AWS**

*Lab 0. Performance Acceleration for OLTP (Aurora, DynamoDB) with Cache layer*

**Table of Contents**

Lab 설명 3

Lab Architecture 3

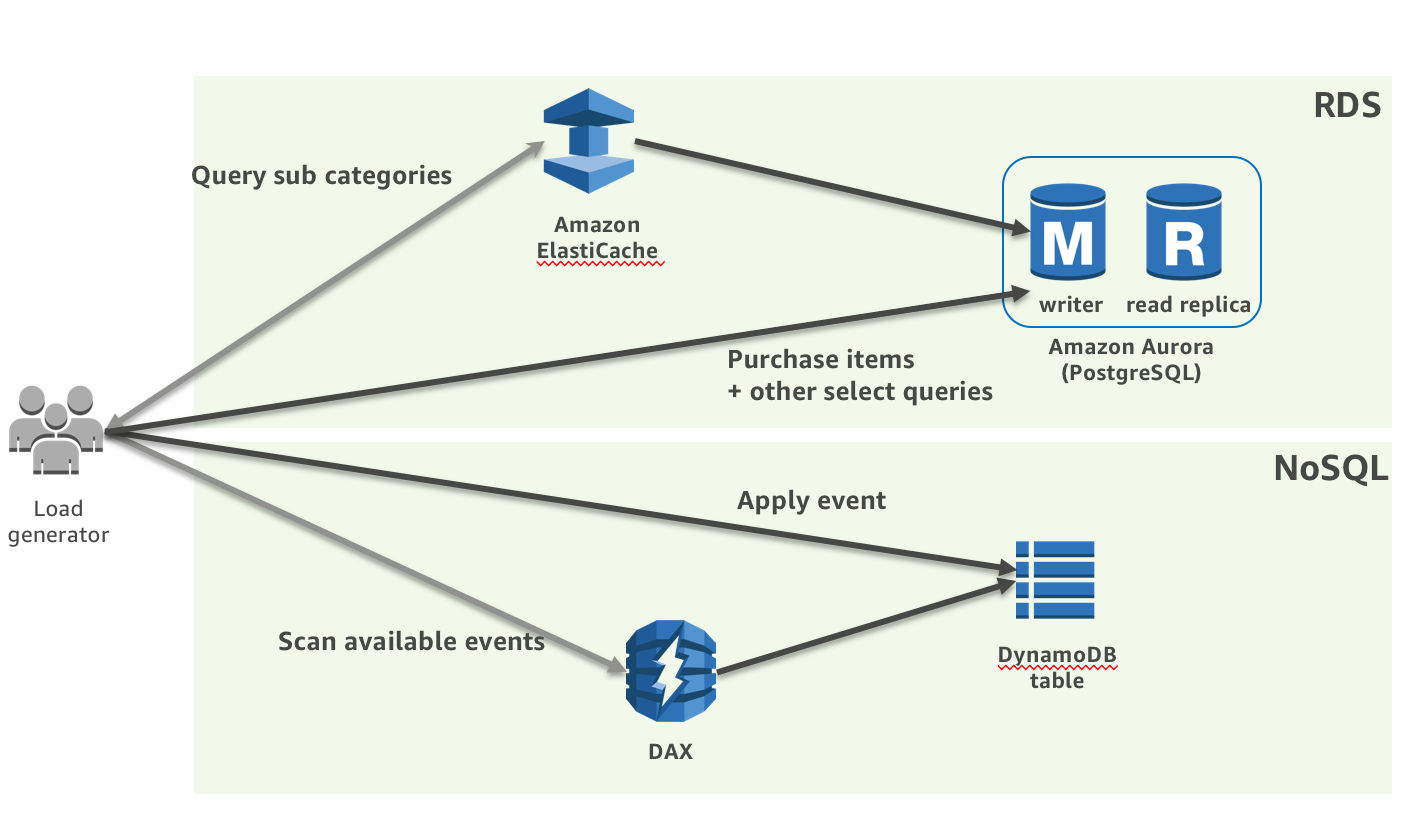
Lab 1-1 Elasticache for RDS 4

Lab 2-2 DAX for Dynamodb 7

# Lab 설명

이번 Lab은 OLTP 데이터(RDB, NoSQL) 사용 시 Read Workload의 성능 개선 실습을 진행합니다. Aurora (RDB) 및 Dynamodb (NoSQL) 데이터를 생성하여 수행 시간을 확인 하고, Cache Layer 추가를 통해 성능 개선을 도모합니다.

# Lab Architecture



단계별 상세 Lab은 다음과 같습니다.

Lab 1-1. Elasticache for RDS

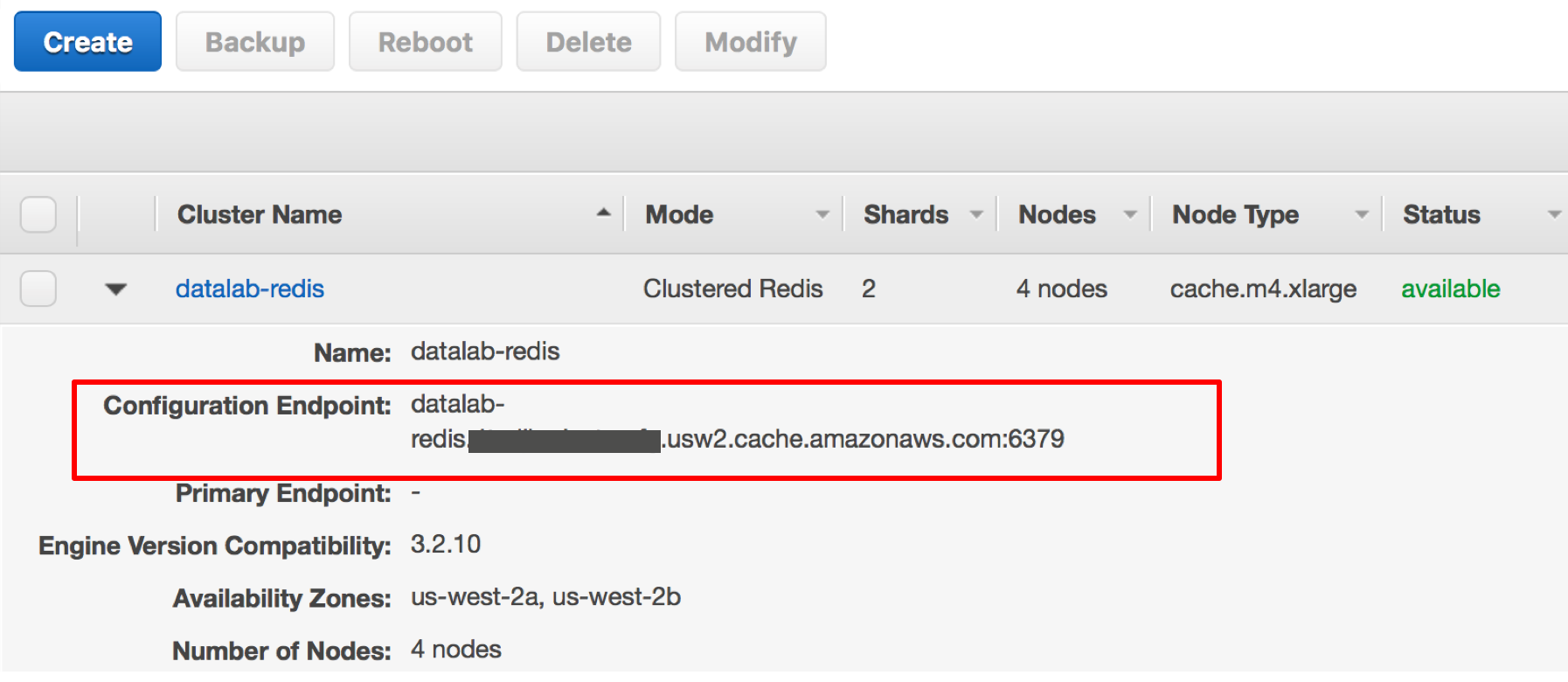
Lab 1-2. DAX for Dynamodb

# Lab 1-1 Elasticache for RDS

1. Aurora database를 조회하는 query\_sub\_categories(sub category 리스트 조회) 워크로드의 수행시간을 Log 에서 확인합니다.

|  |
| --- |
| [ec2-user@ip-10-150-1-248 ~]$ cd /fs1/Log/querySubCategories  [ec2-user@ip-10-150-1-248 querySubCategories]$ ls -lt|head -n 3  total 58812  -rw-rw-r-- 1 ec2-user ec2-user 1297 Jul 18 01:45 201807180145.log  -rw-rw-r-- 1 ec2-user ec2-user 1524 Jul 18 01:44 201807180144.log  [ec2-user@ip-10-150-1-248 querySubCategories]$ tail -f 201807180145.log  **Aurora** Execution time: 0.00296401977539 seconds  **Aurora** Execution time: 0.00289797782898 seconds  **Aurora** Execution time: 0.00031590461731 seconds  **Aurora** Execution time: 0.00149202346802 seconds |

1. Elasticache 서비스 화면으로 이동하여 기 생성된 Redis의 Configuration endpoint정보를 확인합니다.



1. **vi /fs1/scripts/python/query\_sub\_categories\_cache.py** 를 수행 후, Redis endpoint를 configuration endpoint로 갱신합니다.

|  |
| --- |
| # Query from Cache  startup\_nodes = [{"host": "**redis endpoint**", "port": "6379"}]  cache\_con = StrictRedisCluster(startup\_nodes=startup\_nodes, decode\_responses=True, skip\_full\_coverage\_check=True) |

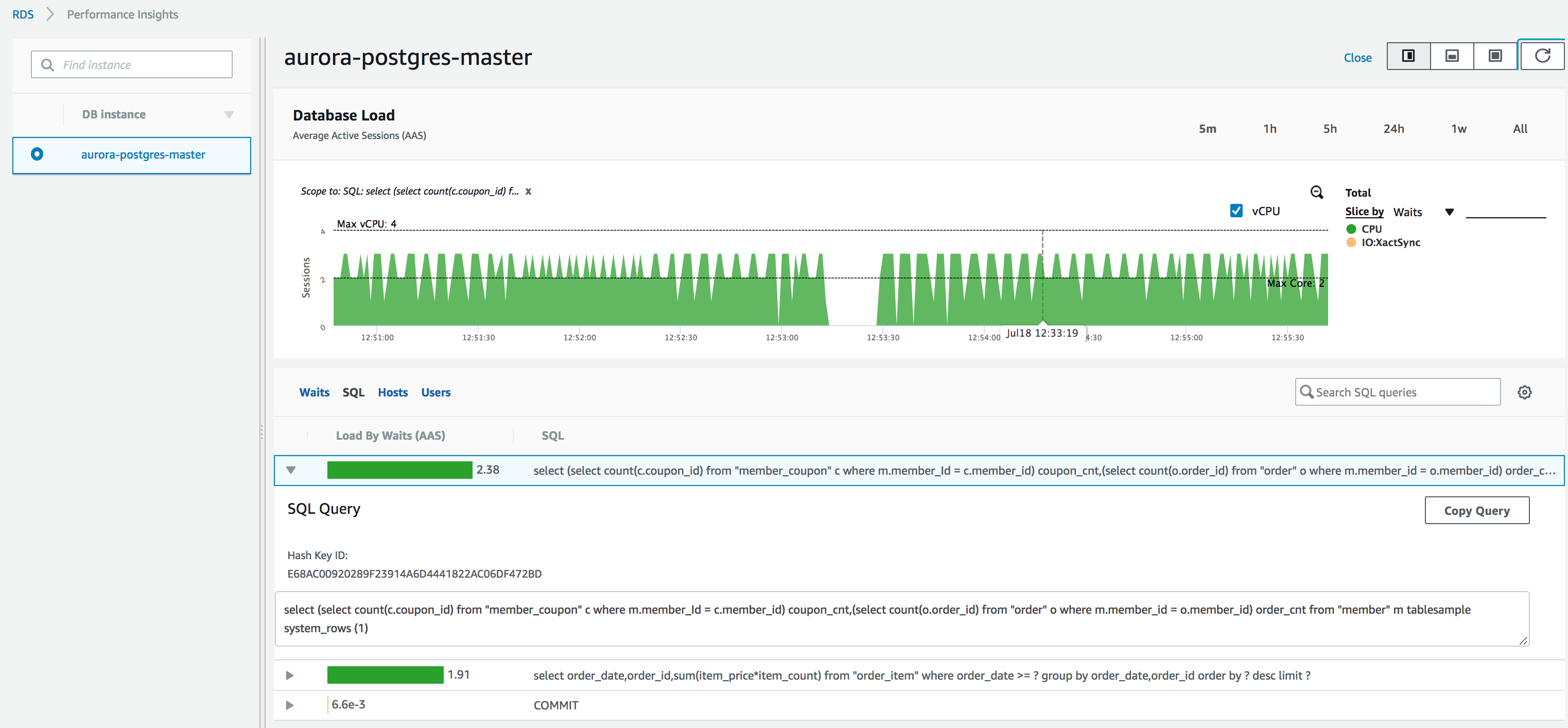
1. **sh /fs1/bin/run\_generator.sh query\_sub\_categories\_cache 1 1** 을 수행하여, Redis에서 조회하는 워크로드를 생성합니다.
2. Elasticache Redis를 조회하는 query\_sub\_categories\_cache(sub category 리스트 조회) 워크로드의 수행시간을 Log 에서 확인합니다.

|  |
| --- |
| [[ec2-user@ip-10-150-1-248 ~]$ cd /fs1/Log/querySubCategories  [ec2-user@ip-10-150-1-248 querySubCategories]$ ls -lt|head -n 3  total 58864  -rw-rw-r-- 1 ec2-user ec2-user 3588 Jul 18 01:58 201807180158.log  -rw-rw-r-- 1 ec2-user ec2-user 2845 Jul 18 01:57 201807180157.log  [ec2-user@ip-10-150-1-248 querySubCategories]$ tail -f 201807180158.log  **Elasticache Execution time: 0.000730991363525 seconds**  Aurora Execution time: 0.00283193588257 seconds  **Elasticache Execution time: 0.000706911087036 seconds**  Aurora Execution time: 0.00293684005737 seconds |

1. Elasticache Redis 를 적용한 /fs1/scripts/python/query\_sub\_categories\*.py 파이썬 코드를 확인합니다.

|  |
| --- |
| [ec2-user@ip-10-150-1-248 ~]$ cd /fs1/scripts/python/  [ec2-user@ip-10-150-1-248 python]$ ls -l query\_sub\_categories\*.py  -rw-rw-r-- 1 ec2-user ec2-user 2557 Apr 30 07:41 query\_sub\_categories\_cache.py  -rw-r--r-- 1 ec2-user ec2-user 1483 Jul 17 07:10 query\_sub\_categories.py |

1. **RDS>performance insights** 로 이동하여Aurora PostgreSQL내 DB workload 를 확인합니다.

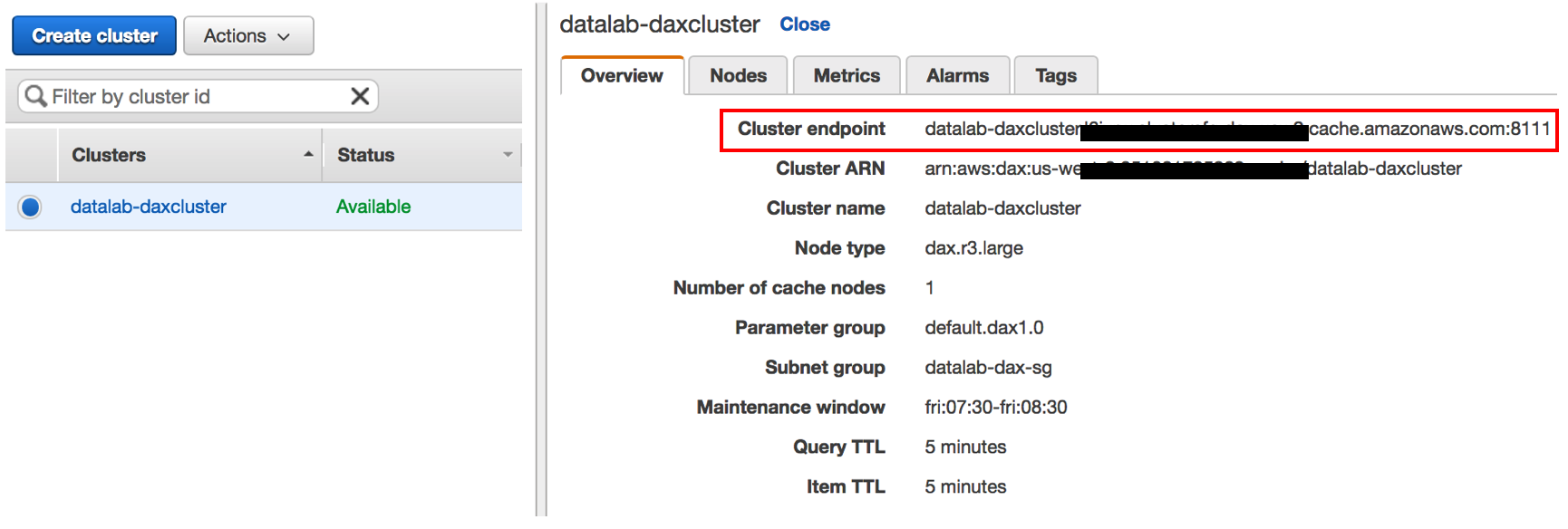


# Lab 2-2 DAX for Dynamodb

1. Dynamodb 를 SCAN하는ddb\_scan\_event (현재 오픈되어있는 event 조회) 워크로드의 수행시간을 Log에서 확인합니다.

|  |
| --- |
| [ec2-user@ip-10-150-1-248 ~]$ cd /fs1/Log/ddb\_scan\_event  [ec2-user@ip-10-150-1-248 ddb\_scan\_event]$ ls -lt|head -n 3  total 1548  -rw-rw-r-- 1 ec2-user ec2-user 4116 Jul 18 02:05 201807180205.log  -rw-rw-r-- 1 ec2-user ec2-user 6171 Jul 18 02:04 201807180204.log  [ec2-user@ip-10-150-1-248 ddb\_scan\_event]$ tail -f 201807180205.log  **DynamoDB** => Available event count:4. # of records scanned: 5. Execution time: 0.00484204292297 seconds  **DynamoDB** => Available event count:4. # of records scanned: 5. Execution time: 0.00485801696777 seconds |

1. Dynamodb콘솔의 **DAX > Clusters** 로 이동하여cloudformation 에서 생성한 DAX 클러스터의 Cluster endpoint 를 확인합니다.



1. **vi /fs1/scripts/python/ddb\_scan\_events\_dax.py**를 수행 후, DAX endpoint를 해당 cluster endpoint로 갱신합니다.

|  |
| --- |
| endpoint = "[**dax cluster endpoint**]"  dax = amazondax.AmazonDaxClient(session, region\_name=region, endpoints=[endpoint])  client = dax |

1. **sh /fs1/bin/run\_generator.sh ddb\_scan\_events\_dax 1 1**을 수행하여, DAX에서 조회하는 워크로드를 생성합니다.
2. DAX를 조회하는 ddb\_scan\_events\_dax (현재 오픈되어 있는 event 조회) 워크로드의 수행시간을 Log 에서 확인합니다.

|  |
| --- |
| [ec2-user@ip-10-150-1-248 Log]$ cd /fs1/Log/ddb\_scan\_event  [ec2-user@ip-10-150-1-248 ddb\_scan\_event]$ ls -lt|head -n 3  total 1628  -rw-rw-r-- 1 ec2-user ec2-user 4178 Jul 18 03:32 201807180332.log  -rw-rw-r-- 1 ec2-user ec2-user 4426 Jul 18 02:14 201807180214.log  [ec2-user@ip-10-150-1-248 ddb\_scan\_event]$ tail -f 201807180332.log  **DynamoDB(DAX)** => Available event count:4. # of records scanned: 5. Execution time: **0.00251913070679 seconds**  DynamoDB => Available event count:4. # of records scanned: 5. Execution time: 0.00598883628845 seconds  **DynamoDB(DAX)** => Available event count:4. # of records scanned: 5. Execution time: **0.00265312194824 seconds**  DynamoDB => Available event count:4. # of records scanned: 5. Execution time: 0.00670504570007 seconds |

SCAN 작업에 대한 DAX성능 향상이 크지 않아 보일 수 있습니다. 참고로 QUERY 또는 GET 등 소량 작업 대해 DAX를 적용하면 성능은 더욱 향상됩니다.

1. DAX를 적용하는 /fs1/scripts/python/ddb\_scan\_events\*.py 파이썬 스크립트 코드를 확인합니다.

|  |
| --- |
| [ec2-user@ip-10-150-1-248 ~]$ cd /fs1/scripts/python/  [ec2-user@ip-10-150-1-248 python]$ ls -l ddb\_scan\_events\*.py  -rw-rw-r-- 1 ec2-user ec2-user 1915 Jul 17 08:49 ddb\_scan\_events\_dax.py  -rw-rw-r-- 1 ec2-user ec2-user 1964 Jul 17 08:19 ddb\_scan\_events.py |