Full Stack Game Demo

Serverless Full Stack with Gomoku

Sep. 2019

Table of Contents

HoL Material by
Version 1.0 Junghun Lee, Sep. 2019
Based on material prepared by Seungmo Koo, Sungsoo Khim, Hyobin An

Hands-on-Lab 실습용 Asset 다운로드
https://leejungh-public.s3.ap-northeast-2.amazonaws.com/hol/GameLiftFlexHol.zip

Source Code
https://github.com/aws-samples/aws-gamelift-sample/tree/FlexMatch

Overview of Game

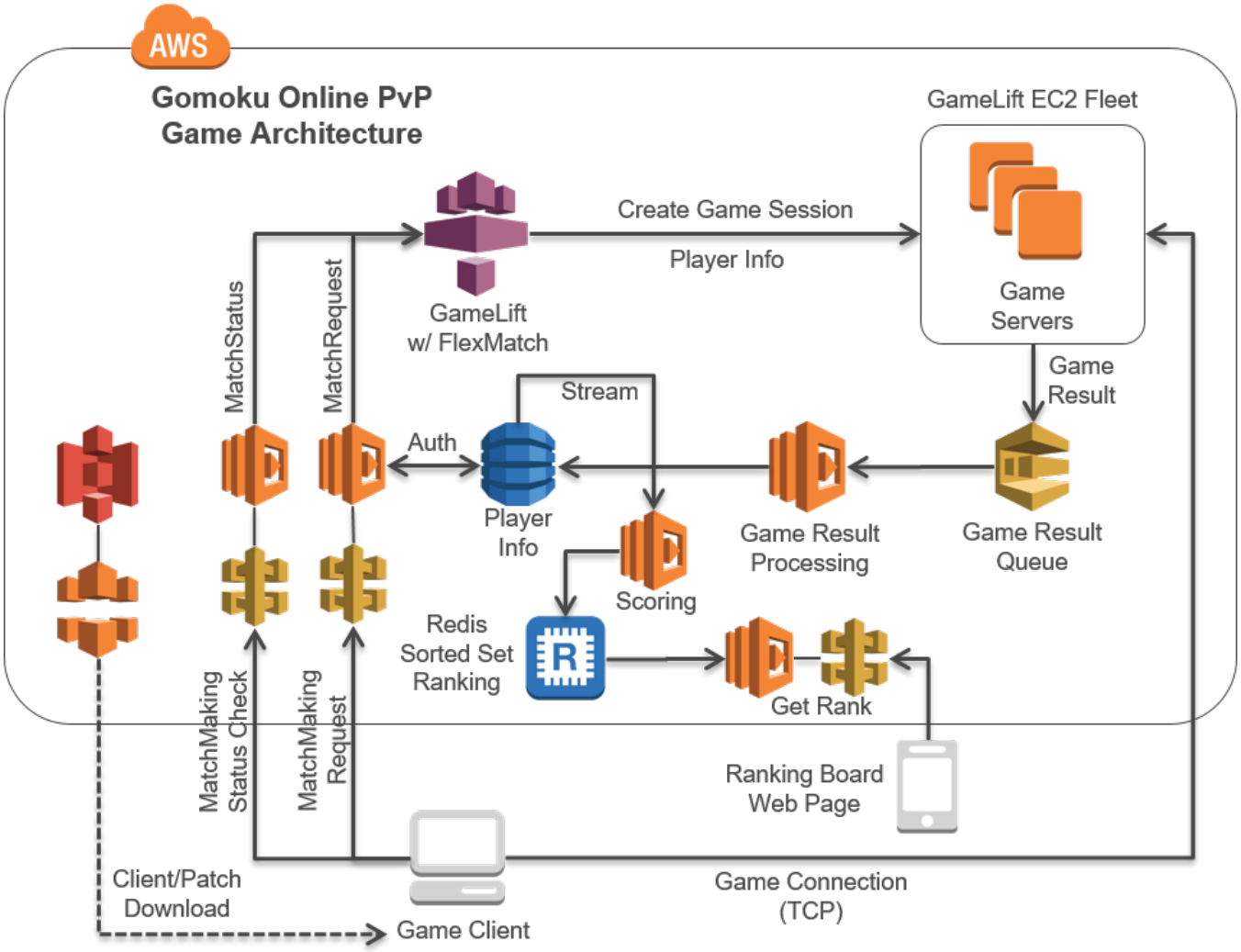
지난 수개월간 만들어오던 여러분의 새로운 게임이 드디어 완성을 눈 앞에 두고 있습니다. 가로 세로 19 칸으로 그어진 선 위에서 우주만물의 음과 양을 상징하는 검은 돌과 흰 돌을 사용하는 게임입니다. 두 명의 플레이어는 그들의 검은 돌과 흰 돌을 각각 격자에 놓게 되고, 처음으로 5 개의 돌을 직선(혹은 대각선)으로 만든 플레이어가 승리하게 됩니다. 여러분은 이 게임의 이름을 오목이라고 부르기로 결정했습니다. 굉장히 익숙하지요? 하지만 그렇지 않습니다(...).

하지만 이 게임을 플레이하는 도중 사소한 오류를 발견합니다. 이 게임을 하려면 두 명의 플레이어가 있어야 하는데, 동일한 사람들과 계속 플레이를 하다 보니 게임이 재미 없다는 것입니다. 지금은 2010 년대이지 1970 년대가 아닙니다. 그래서 여러분은 몇 가지 AWS 의 서비스를 이용하여 "인터넷"을 통한 플레이를 제공하기로 결정했습니다. 힘들 것 같나요? 하지만 게임은 이미 준비되어 있습니다.

자, 그러면 한가지만 명심하면 됩니다. 여러분에게 큰 운영(Ops) 조직이 없기 때문에 Serverless 및 관리형 서비스를 활용하여 최대한 효율적이고 최적화된 방식을 활용해서 만들어야 합니다.
즉, API Gateway, Lambda, ElastiCache(Redis), S3, DynamoDB, SQS, 그리고 GameLift Service 를 사용한 Full Stack 을 만드는겁니다!

Architecture Diagram

아래의 아키텍처는 우리가 이번 Lab 을 통해 만드는 것입니다. 본격적인 개발에 앞서 이렇게 아이디어를 아키텍처로 그리는 것이 좋습니다.



아키텍처에서 볼 수 있듯이 어느 정도의 AWS 지식만 있다면 충분히 할 수 있습니다.

Let's start to stack the full stack!

본격적인 시작에 앞서 게임 클라이언트와 서버 바이너리를 모두 다운로드 받았는지 확인하세요. 소스코드를 다운받아 직접 컴파일 할 수도 있지만, 이번 랩에서 우리는 서버와 클라이언트 모두 이미 컴파일 된 바이너리를 사용하려고 합니다.

이번 랩에서 사용될 파일의 다운로드 URL 은 따로 제공됩니다. (클라이언트 바이너리, 서버 바이너리, Lambda 코드 모두 포함되어 있습니다)
또한 랩 시작 전에 어떤 VPC 를 사용할지 결정하세요. 대부분의 서비스는 VPC 와 관계없지만 몇 가지 서비스는 VPC 에 종속성을 갖고 있습니다. 또한 우리는 보안이 뛰어난 디자인을 가져가고 싶습니다. 이번 랩에서는 default VPC 를 사용할 것이지만, 원한다면 직접 설계한 VPC 를 사용하셔도 괜찮습니다. (이번 랩은 여러분이 이미 VPC 와 보안 그룹 등을 다룰 줄 안다고 가정하고 있습니다)
그러면, 가장 중요한 Database 에서부터 시작하겠습니다.

Section 1: DynamoDB and ElastiCache – Redis and SQS. With a dash of IAM roles.

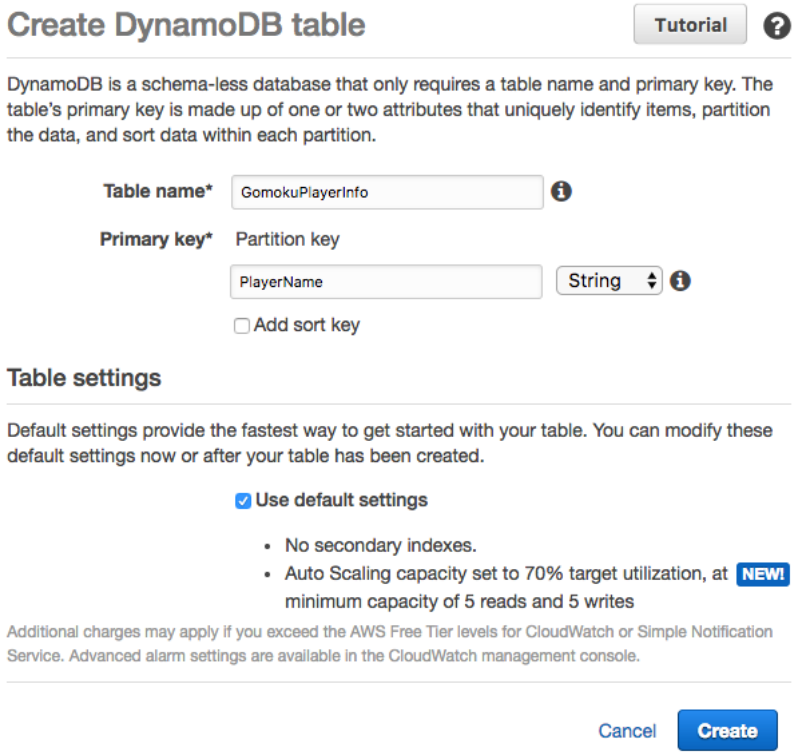
이번 랩은 DynamoDB를 사용해 사용자 정보와 대전 결과를 저장할 것입니다. 우리는 간단한 정보를 저장하고 또한 이 정보는 Web을 통해 접근할 수 있는 리더 보드에 사용될 것입니다. 추가로, DynamoDB 앞단에는 ElastiCache를 배치하여 리더 보드데이터를 캐싱하여 퍼블릭에 제공합니다. 왜냐하면 여러분의 App은 엄청난 인기를 끌게 될 것이고, 때문에 우리는 여러분의 DB가 동일한 요청을 수행하느라 부하가 걸리게 되길 원치 않습니다.

모든 게임 결과는 SQS를 이용해 대기열에 들어가고 Lamdba를 활용하여 DB에 삽입할 것입니다. 그래서 결과를 테이블에 삽입할 때까지 기다리지 않아도 됩니다.이것은 또한 게임 흐름이 다른 구성 요소들이 끝날 때까지 기다리지 않아 플레이어에게 더 좋은 사용자 경험을 줄 수 있습니다. 수 많은 플레이어가 모두 처리해야하는 결과를 보내려고 할 때를 상상해보세요. 이것은 아마 혼란을 발생시키고 좋지 않은 사용자 경험을 주게 될 것입니다.

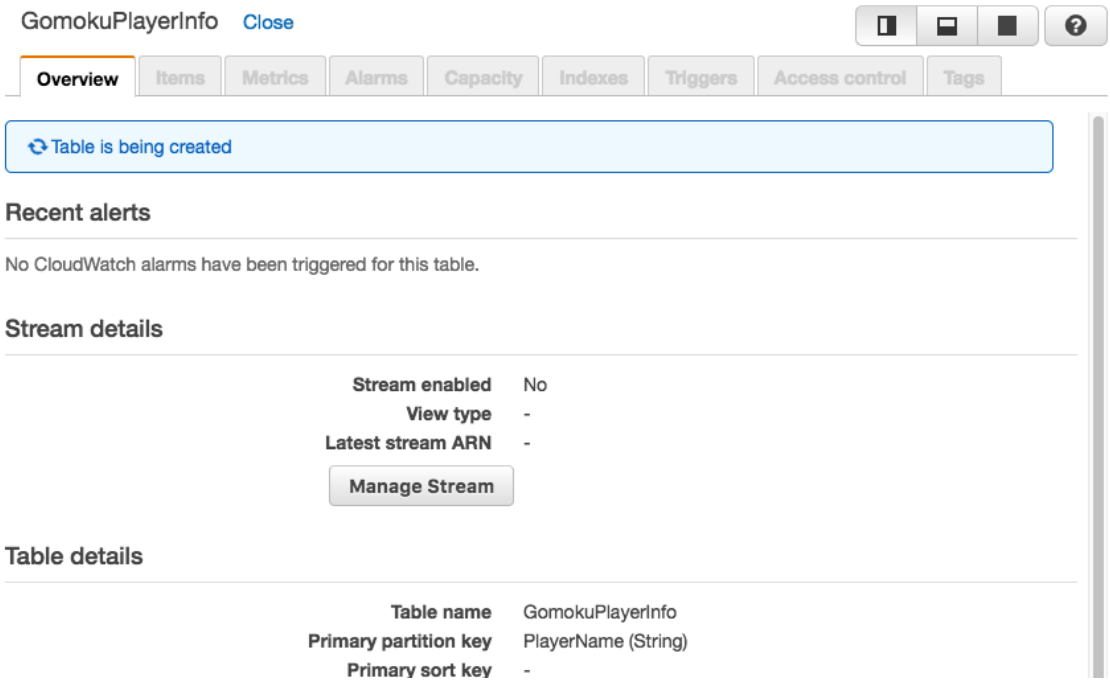
자, 이제 DynamoDB 테이블을 만들어 보겠습니다.

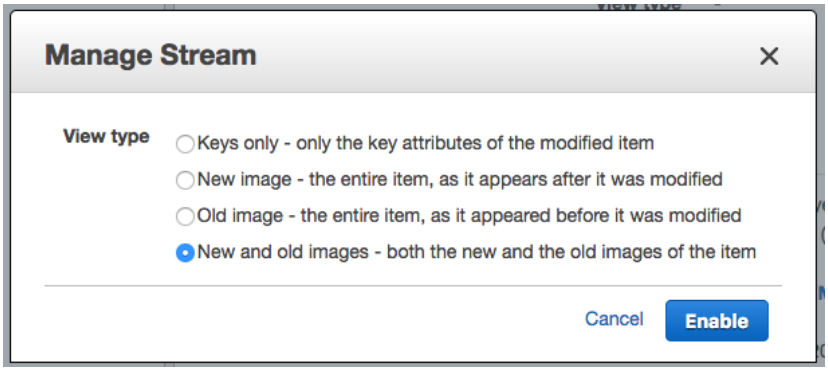
1. AWS 콘솔에 로그인하고 **DynamoDB** 페이지로 이동합니다.
   https://console.aws.amazon.com/dynamodb
2. **리전**을 확인합니다. 랩을 진행하면서 모든 서비스 요소들을 하나의 리전에서 생성하셔야 합니다.
3. 콘솔에서 **Create Table** 을 클릭하여 DynamoDB 테이블 생성을 시작합니다.
4. Table name은 "**GomokuPlayerInfo**" 로 설정하고, Primary Key는 "**PlayerName**" 으로

하고 데이터 타입은 **String**을 선택합니다. 그리고 나서 **Create** 버튼을 클릭합니다.

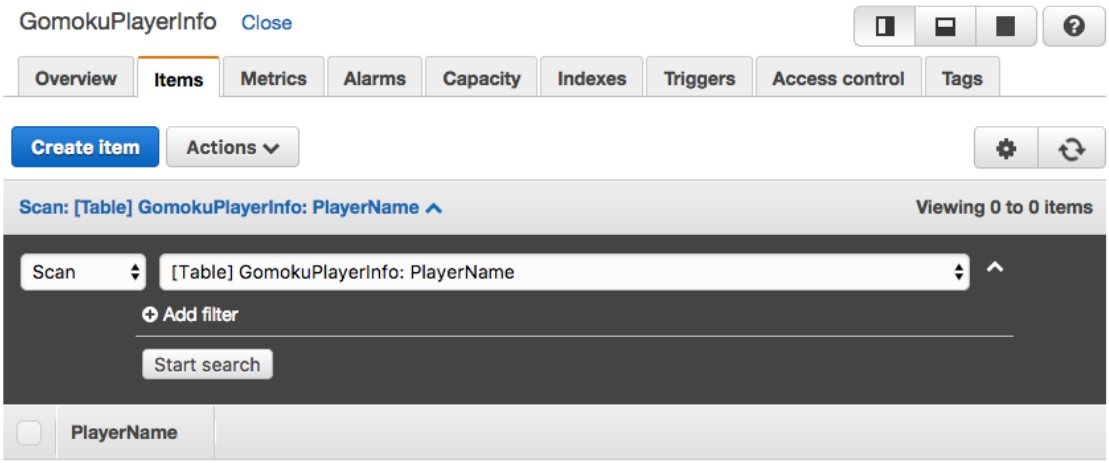


5. 테이블이 생성된 뒤에는 Stream을 활성화 합니다. 기본적으로 stream은 비활성화 되어 있습니다. Manage Stream 버튼을 클릭합니다.

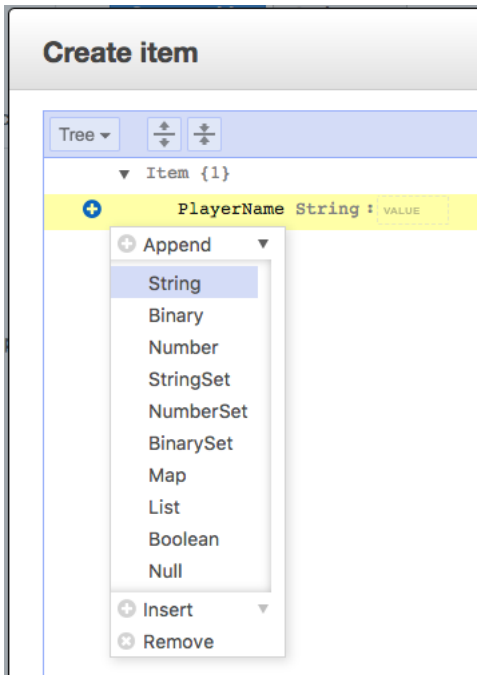




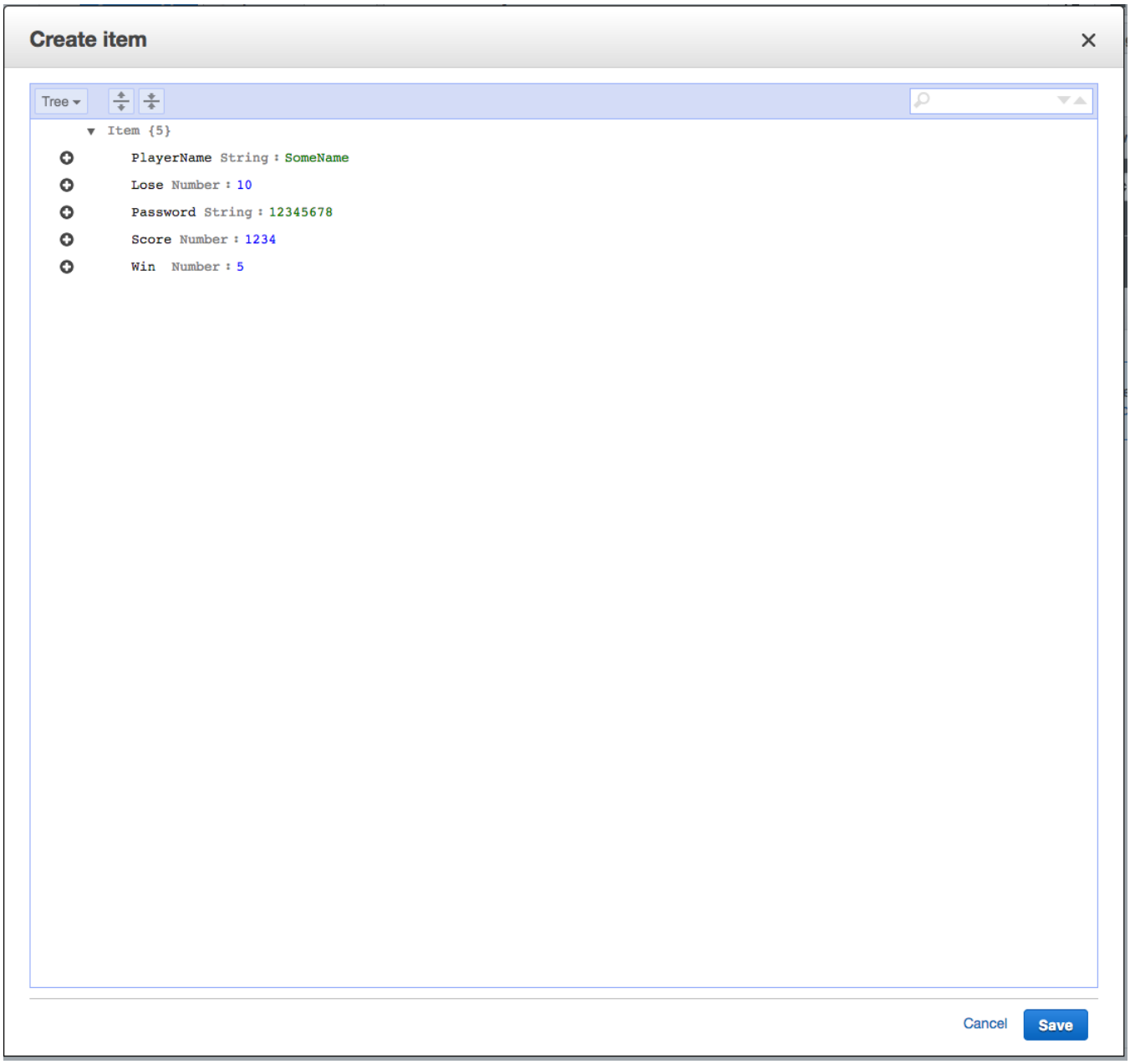
6. **New and old images** 를 선택하고 **Enable**버튼을 클릭합니다.
7. 기본적인 작업은 끝났습니다. 이제 테스트 데이터 샘플을 만들어 줍니다. 테이블에서 **Items** 탭을 선택합니다.



8. **Create Item** 을 클릭하여 새로운 항목을 만들어줍니다.
9. 편집기에서 + 버튼을 클릭하고, **Append**를 선택하여 추가합니다.



10. 다음 스크린 캡처와 같아질 때까지 데이터를 계속 추가해줍니다. 그리고 **Save** 버튼을 클릭하여 저장합니다. (데이터 타입에 주의해주세요)

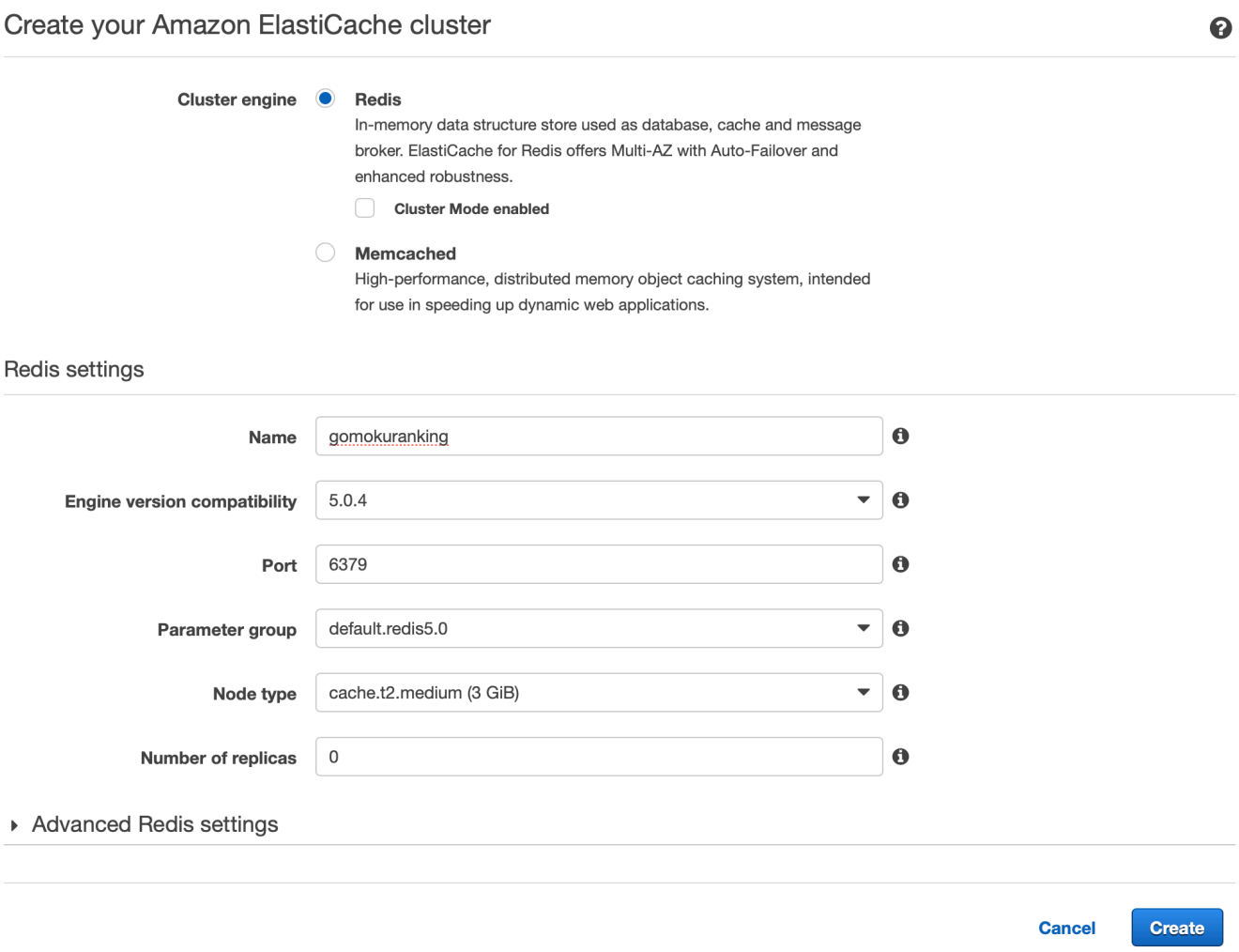


(*) Items tab에서 Scan, [테이블 이름] 을 선택하고 Start search버튼을 누르면 위에서 추가한 항목이 화면이 나타나는 것을 확인 할 수 있습니다.

DynamoDB 설정은 끝났습니다.

이제 ElastiCache를 설정합니다. 이것은 순위 정보를 저장할 것입니다.

1. AWS 콘솔에서 **ElastiCache**로 이동합니다. https://console.aws.amazon.com/elasticache
2. ElastiCache cluster를 생성합니다. 우리는 **Redis** 엔진을 사용할 것입니다.
3. 다음 스크립 캡처와 같이 필요한 정보를 입력해줍니다.



Name: gomokuranking

Engine: 5.0.x (As of 2019 June, 5.0.4)

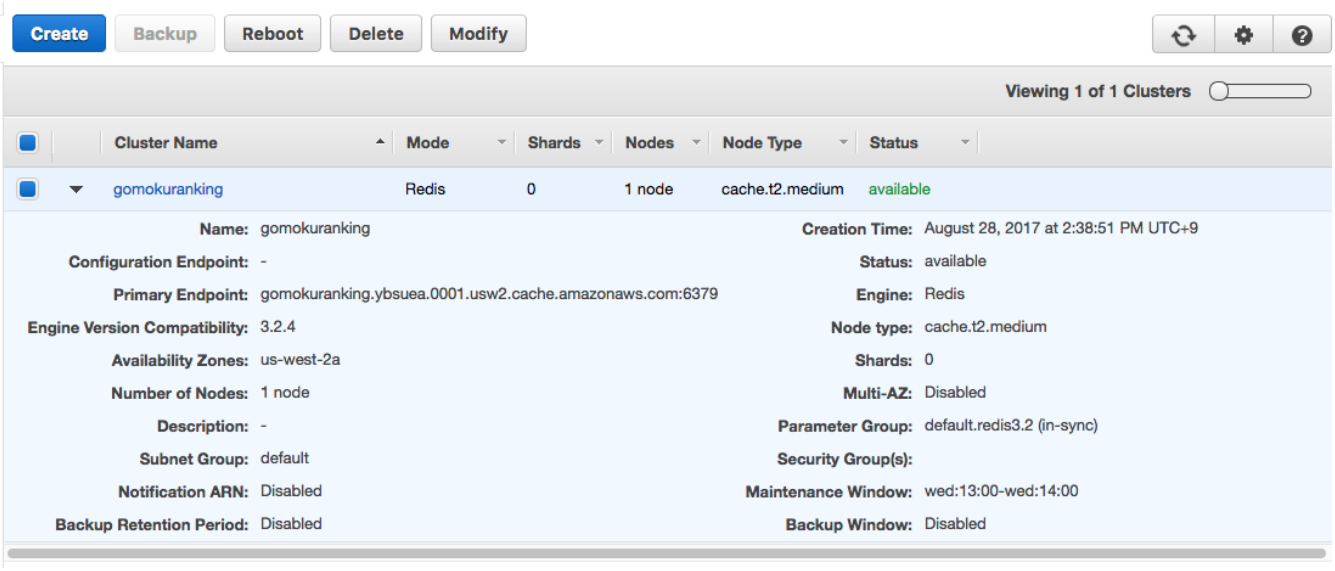
Port: 6379 (default)

Parameter group: default

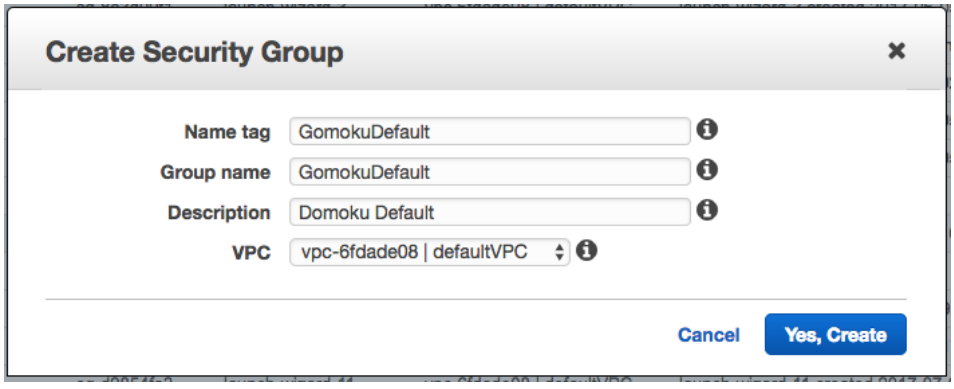
Node type: t2.medium

Number of replicas: 0

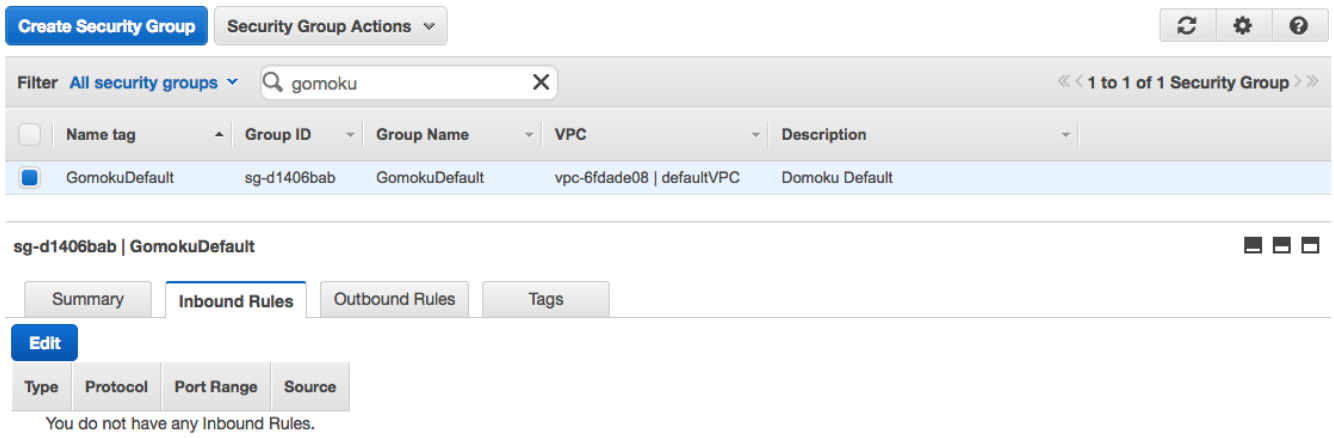
4. 모든 설정이 완료되면, **Create** 버튼을 클릭하여 Redis 클러스터를 생성합니다. (시간이 조금 걸리기 때문에 다음 단계인 SQS 생성을 먼저 진행할 수도 있습니다.)
5. 만약 특정한 VPC내에서 생성하고 싶다면 Advanced Redis setting 페이지로 이동하여 여러분의 VPC 정보를 입력합니다.
6. 생성한 Redis 클러스터의 상태가 available이 되면 Primary Endpoint를 따로 기록해둡니다. 추 후 Lambda 생성 시에 해당 Endpoint가 사용됩니다.



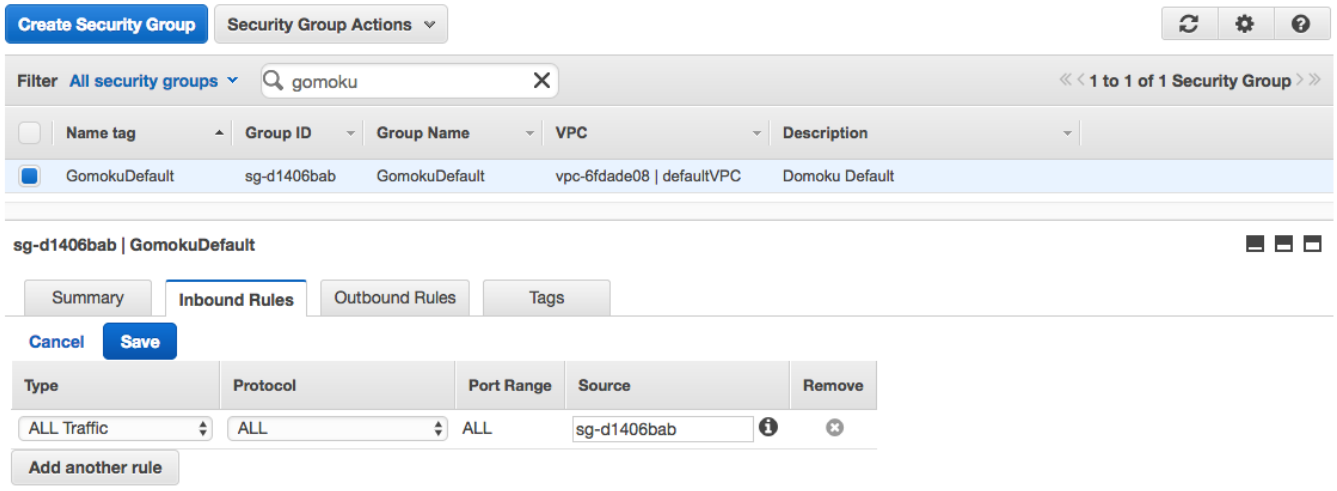
7. 생성한 ElastiCache의 보안을 강화하기 위하여 gomokuranking 클러스터에 안전한 보안 그룹을 생성하여 할당합니다. 이번 실습에서는 아주 간단한 보안그룹 정책을 생성하여 할당하겠습니다.
8. VPC 콘솔에서 https://console.aws.amazon.com/vpc 좌측의 **Security Group** 메뉴를 선택합니다..
9. **Create Security Group** 버튼을 클릭합니다.



10. Name tag, Group name 등에 적절한 정보를 입력해주고, VPC는 실습을 진행 중인 **default VPC**를 선택합니다. (리전에 VPC가 하나라면 default가 따로 표시되지는 않습니다)
11. 보안 그룹 내의 통신을 위하여 inbound 정책을 수정해야 합니다. 생성한 보안 그룹을 선택하고 **Inbound Rules**탭을 클릭합니다.



12. **Edit** 버튼을 클릭하고 다음 스크린 캡처와 같이 정책을 생성합니다. 여기서 **Source**에는 보안 정책 **자신의 Group ID**를 입력합니다. 이렇게 함으로써 이 보안 그룹을 할당한 호스트와 서비스들끼리 통신을 할 수 있습니다. 생성한 보안 그룹을 기억해 둡니다.



Type: All traffic

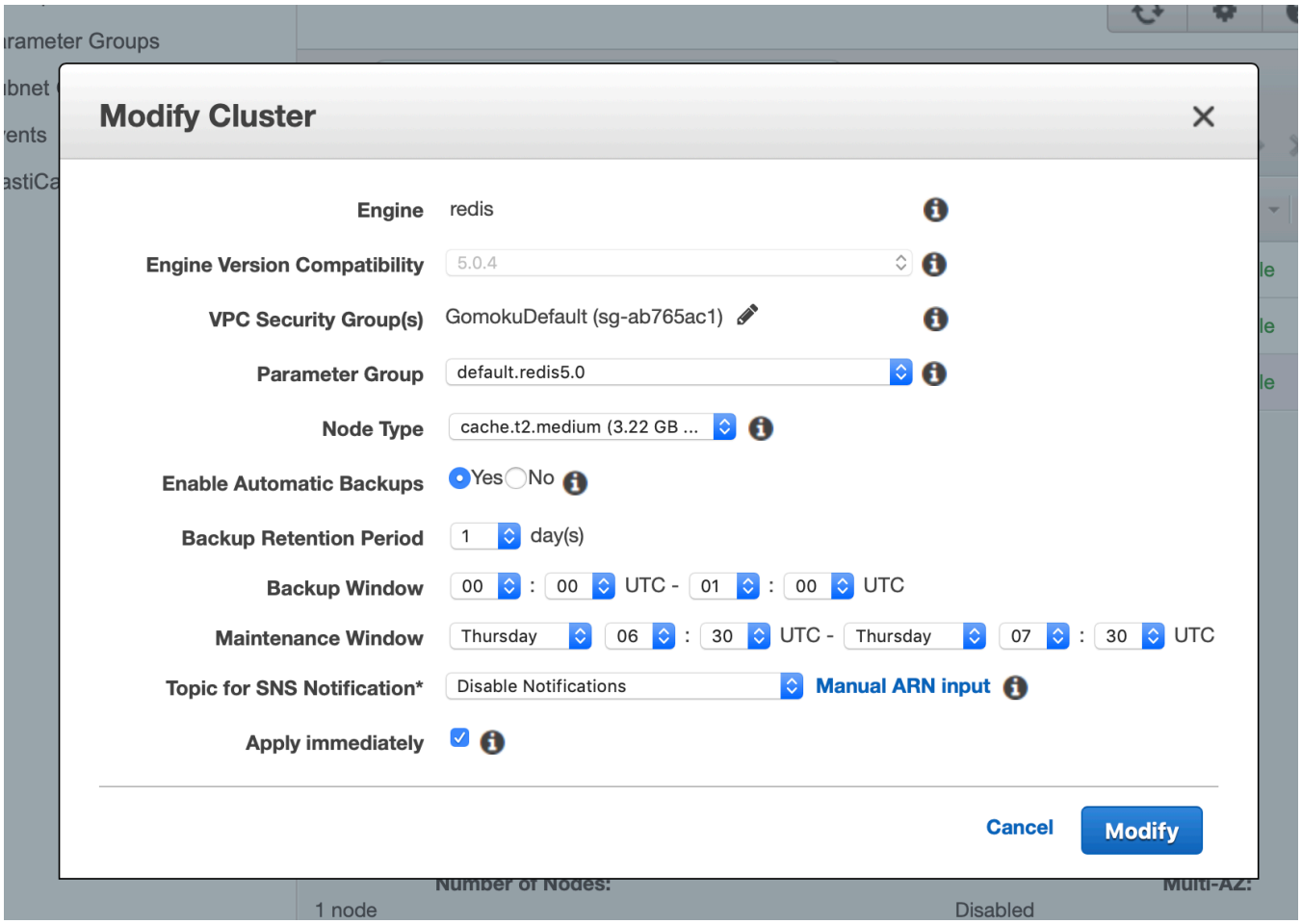
Protocole: All

Source: Security Group itself

13. 보안 그룹 생성을 완료하였다면 다시 ElastiCache 페이지로 돌아와 생성한 Redis

클러스터를 선택합니다.

14. 클러스터를 선택하고 상단의 **Modify** 버튼을 클릭합니다.

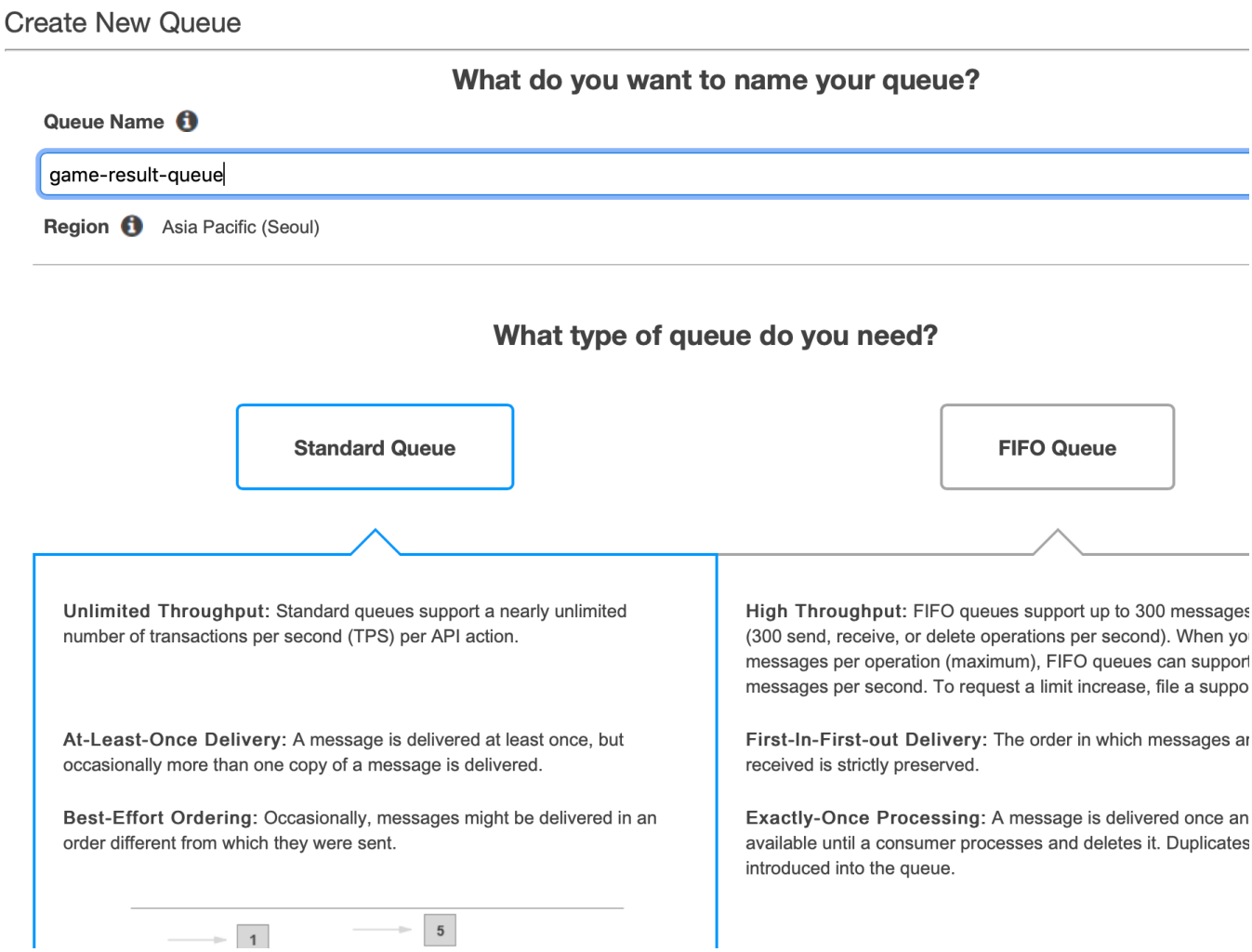


15. 팝업 메뉴에서 VPC Security Group에 방금 생성한 보안그룹을 선택한 뒤 Modify 버튼을 클릭하여 완료합니다.

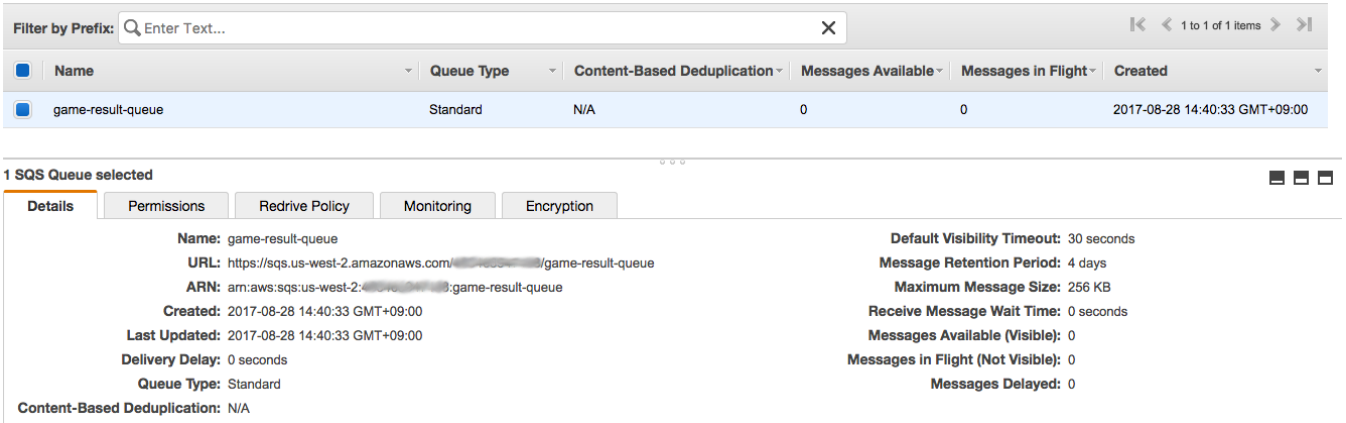
ElastiCache 설정을 완료하였습니다. 이제 SQS 설정을 시작합니다. SQS를 이용하여 게임 결과 처리를 위한 대기열을 만들 것입니다.

1. 콘솔에서 SQS 메뉴로 들어갑니다. https://console.aws.amazon.com/sqs
2. **Create New Queue**를 클릭하여 생성을 시작합니다. Queue 이름은 **game-result-queue**로 입력하고, **Standard Queue**를 선택합니다. **Quick-Create Queue** 버튼을

클릭하여 Queue 생성을 완료합니다.



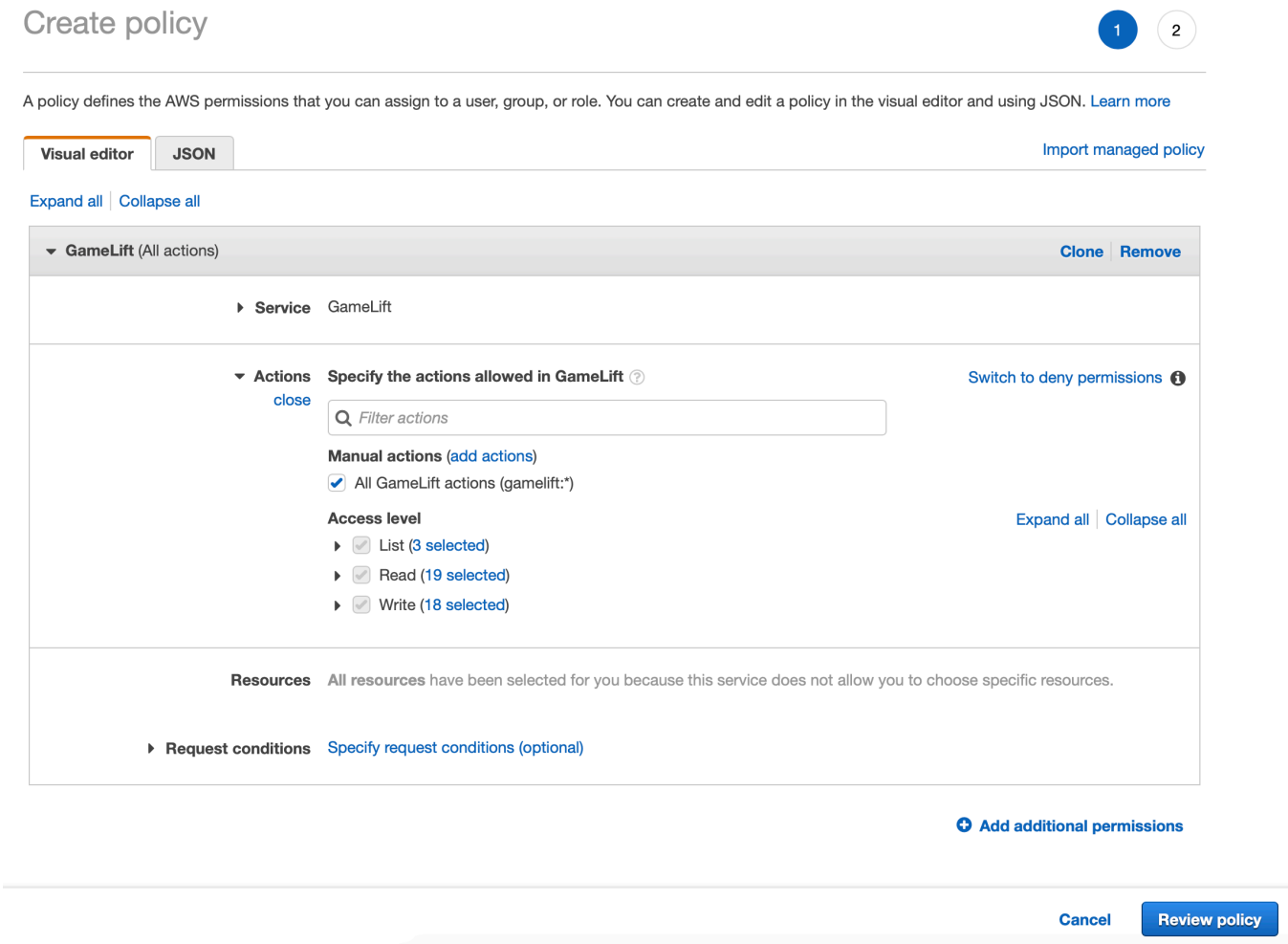
3. Queue 생성이 완료되면 Details에 보이는 endpoint URL을 기록해둡니다. 마찬가지로 뒤의 Lambda 소스 코드에 사용할 예정입니다.



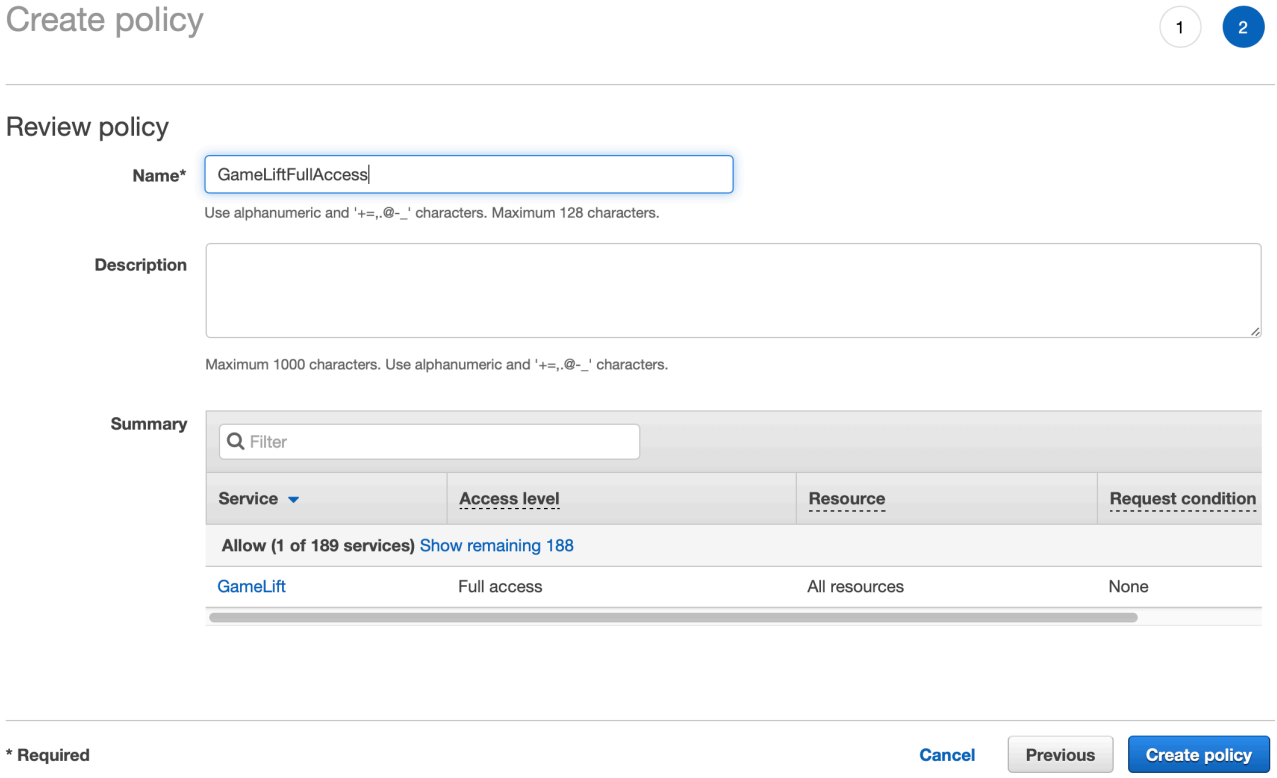
생성된 Queue와 함께 아까 생성된 Dynamo DB, ElastiCache의 동작은 뒤의 Lambda를 통하여 확인하도록 하겠습니다.

마지막으로, 이번 실습에서 만드는 full stack application에 사용할 IAM 정책(policy)와 역할(role)을 생성합니다.

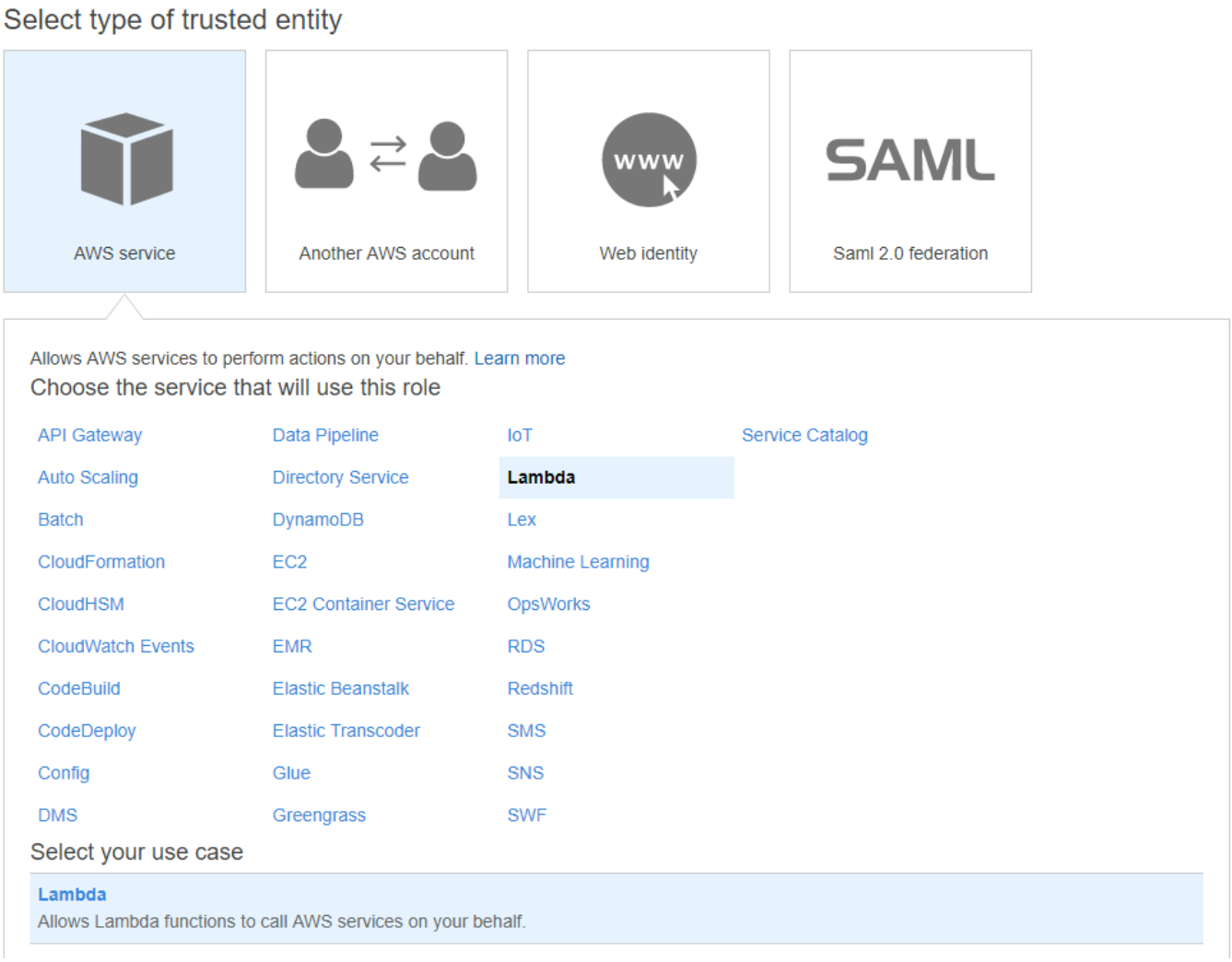
1. 우선, 역할에 사용될 정책을 만듭니다. 이 랩에서는 편의를 위해서 AWS Managed policy를 주로 사용합니다만, GameLift 서비스는 현재 managed policy를 제공하지 않으므로 명시적으로 정책을 생성해야 합니다. 콘솔에서 **IAM** 메뉴로 이동합니다. https://console.aws.amazon.com/iam
2. 메뉴에서 **Policies**를 선택하고 **Create policy** 버튼을 클릭합니다.
3. Visual editor에서 **Service**는 GameLift를 선택하고 **Actions**에서는 랩의 편의상 All GameLift actions를 선택합니다. **Review Policy**를 클릭하여 다음 단계로 진행합니다.



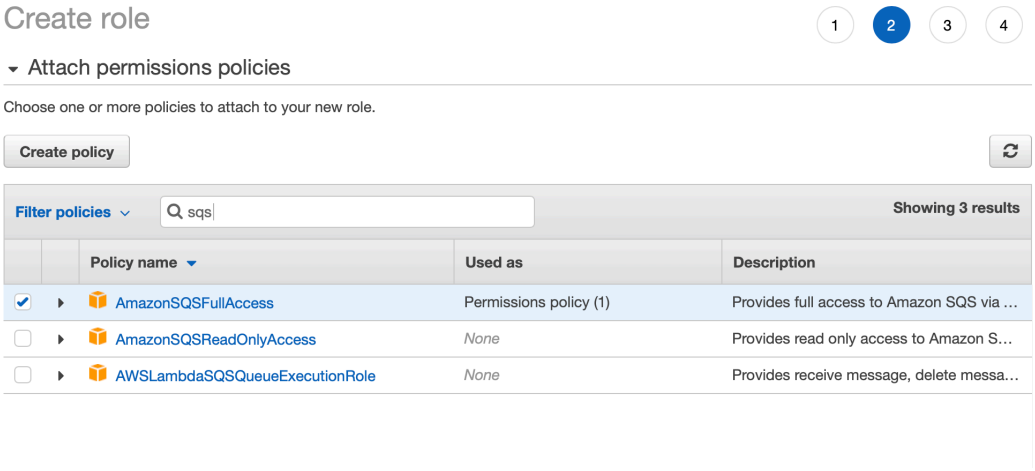
4. 정책의 이름을 입력하고 **Create policy** 버튼을 클릭합니다.



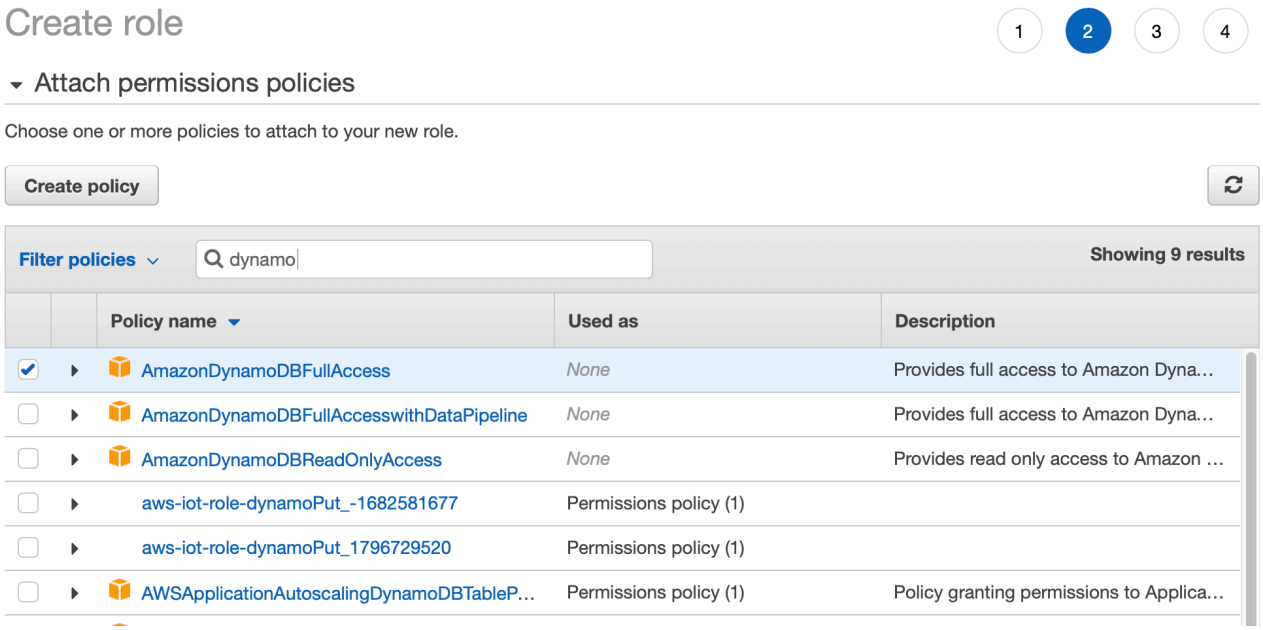
5. 다음은 후반부 Lamdba 함수에 사용할 5 가지 역할(Role)을 생성할 것입니다.
6. IAM 콘솔에서 **Role** 메뉴로 이동한 뒤 **Create role** 버튼을 클릭합니다.
7. Role type에서 AWS Service Role의 **AWS Lambda**를 선택합니다.



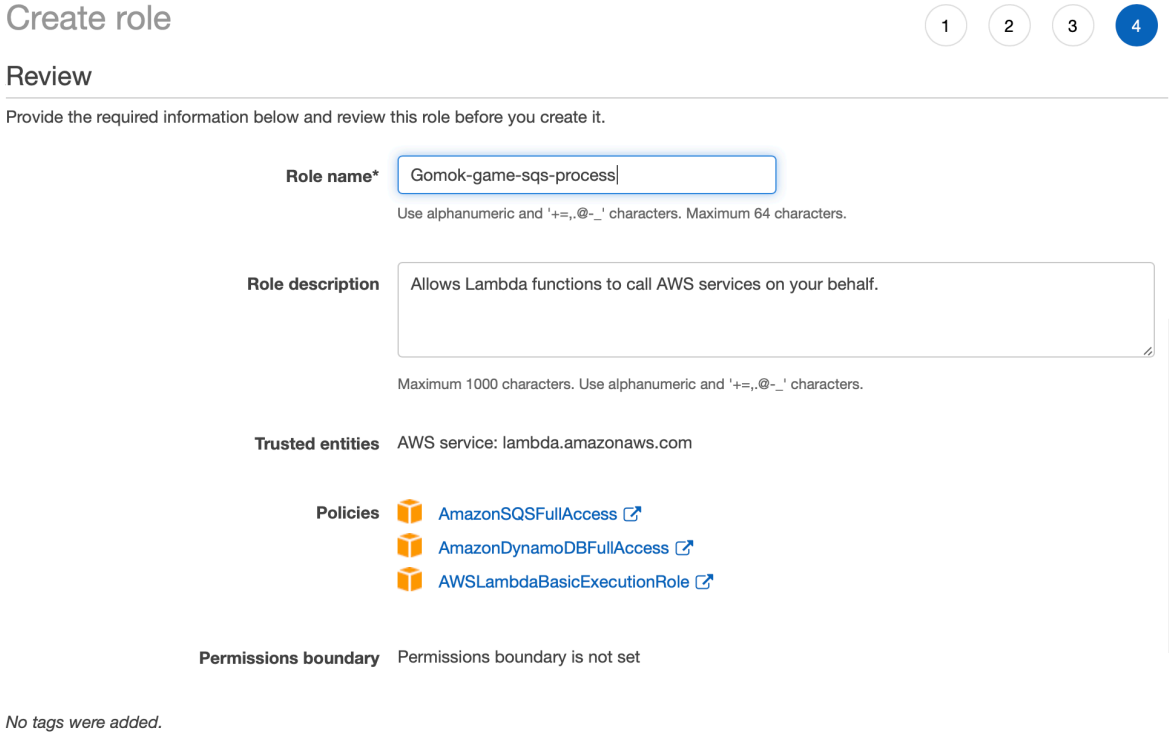
8. 이 Role에는 3개의 정책을 할당할 것입니다. 첫 번째는 **AmazonSQSFullAccess** 정책입니다.



9. 그 다음에는 **AmazonDynamoDBFullAccess** 정책 및 **AWSLambdaBasicExecutionRole** 정책을 선택한 뒤 **Next: Tag** 버튼을 클릭합니다.



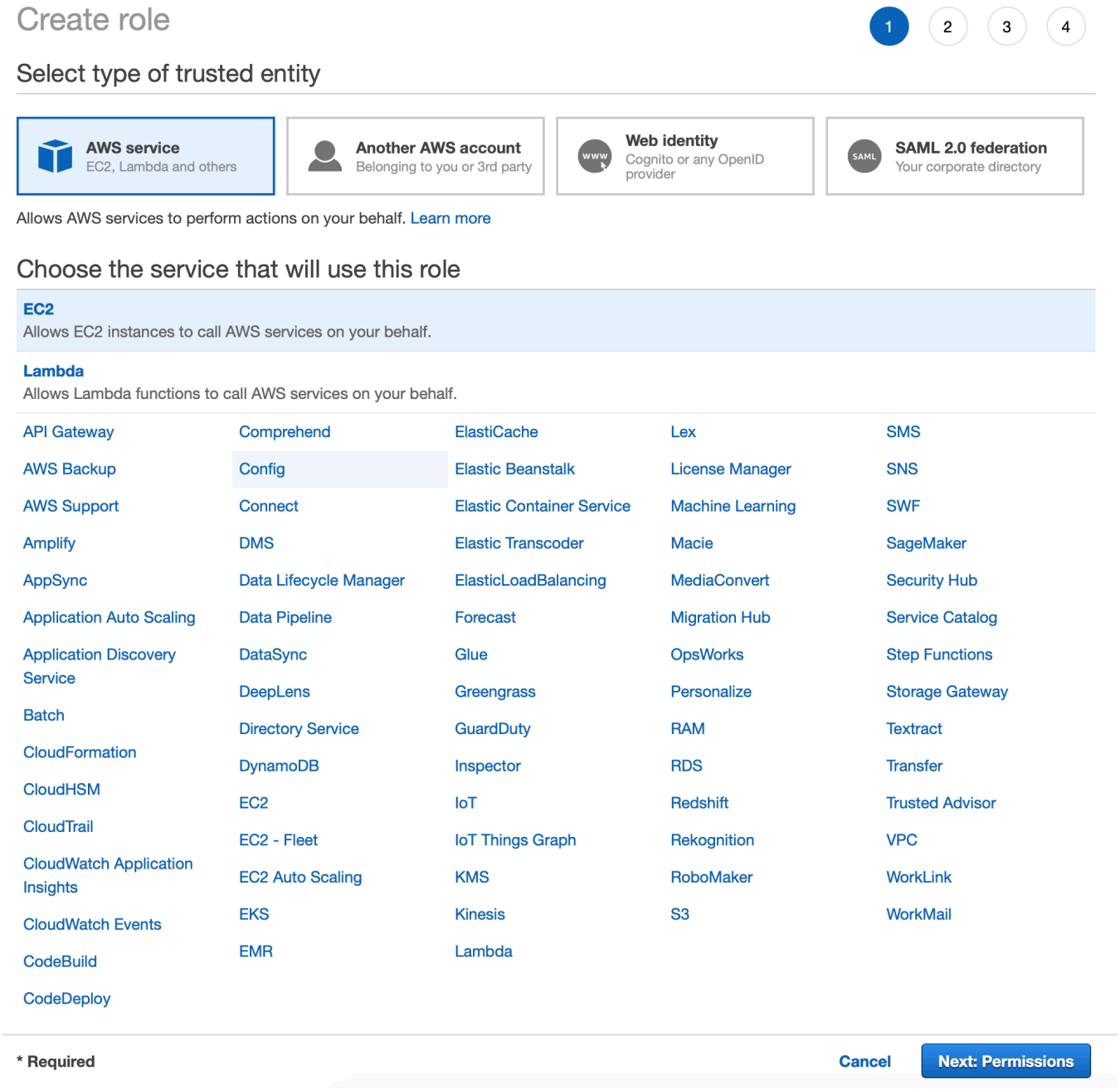
10. Tag는 선택 사항으로 입력합니다. **Next: Review**를 선택합니다. Role name은 **Gomok-game-sqs-process**를 입력합니다. (AWS Console의 최신 UI 업데이트 상황에 따라 일부 웹 인터페이스 변경되었을 수 있습니다.)



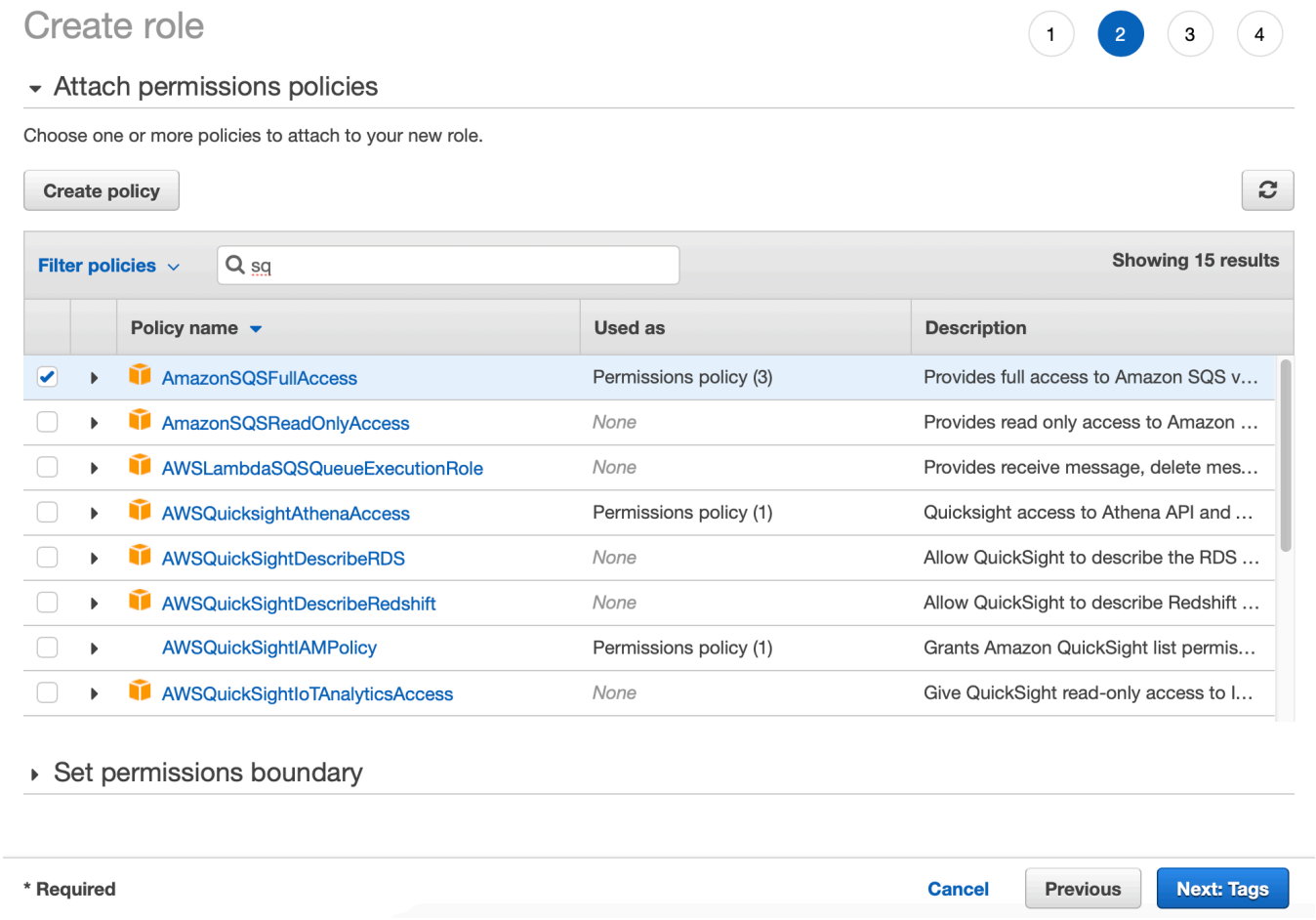
11. **Create role** 버튼을 클릭하여 첫 번째 역할 생성을 완료합니다.
12. 두 번째 역할도 첫 번째와 동일한 방법으로 생성합니다. 하지만 이번에는 **AmazonDynamoDBFullAcess, AmazonVPCFullAccess, AWSLambdaBasicExecutionRole** 정책을 추가합니다.
13. 두 번째 역할의 이름은 **Gomok-game-rank-update**로 지정합니다.
14. 세 번째 Role도 동일한 방법으로 생성합니다. 이번에는 **VPC Full Access, AWSLambdaBasicExecutionRole** 정책을 추가해줍니다.
15. 세 번째 역할의 이름은 **Gomok-game-rank-reader**로 지정합니다.
16. 네 번째 역할도 첫번째와 동일한 방법으로 생성합니다. 이번에는 **AWSLambdaBasicExecutionRole, AmazonDynamoDBFullAccess,** 그리고 앞에서 생성한 **GameLiftFullAccess** 정책을 추가해 줍니다.
17. 네 번째 역할의 이름은 **Gomok-game-match-request**로 지정합니다.
18. 마지막으로 다섯 번째 역할도 동일한 방법으로 생성합니다. 이번에는 **AWSLambdaBasicExecutionRole**과 앞에서 생성한 **GameLiftFullAccess** 정책을 추가해 줍니다.
19. 다섯 번째 역할의 이름은 **Gomok-game-match-status**로 지정합니다.
20. 모두 생성이 완료되었다면 다음 스크린 캡처와 같이 다섯개의 역할을 확인할 수 있습니다.

21. 마지막으로 하나의 역할을 더 생성합니다. 앞서는 Lambda가 사용할 역할을 생성하였지만, 이번에는 GameLift에서 생성하는 Fleet에서 사용할 역할을 생성합니다. GameLift의 Fleet에서는 Game의 결과를 SQS에 전송할 때 사용하는 권한이 필요합니다.
22. Create Role을 선택하여 새로운 역할을 생성합니다. 이 역할의 사용은 GameLift

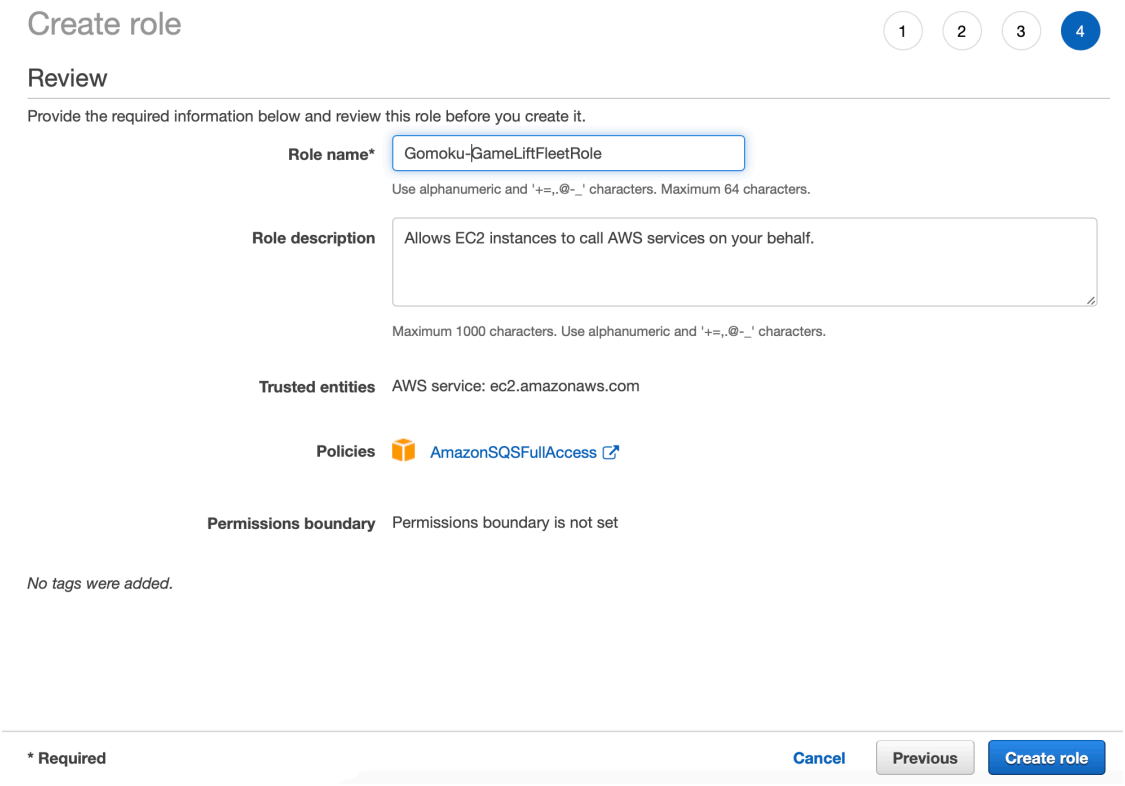
서비스가 수행하므로 GameLift를 서비스를 선택해야 하지만, 콘솔에는 GameLift가 서비스로 표시되지 않습니다. 여기서는 EC2를 선택하고 다음으로 이동합니다.



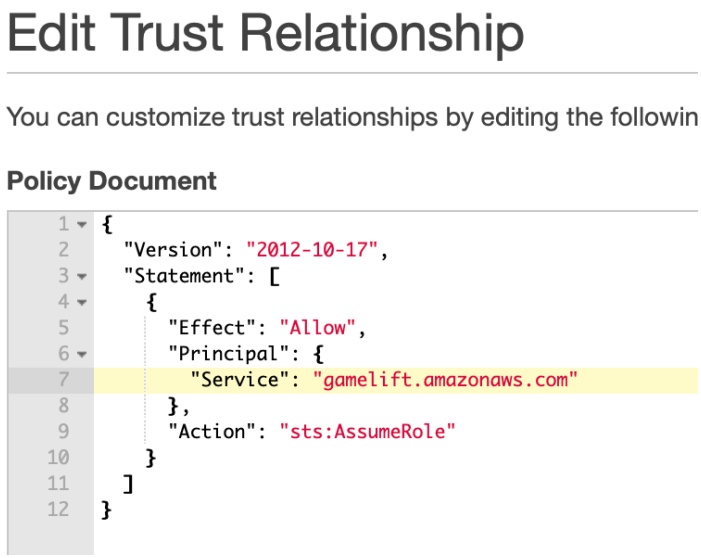
23. 이 역할이 필요한 권한은 AmazonSQSFullAccess를 선택합니다.



24. 역할 이름은 Gomoku-GameLiftFleetRole로 입력합니다.



25. 이 역할을 사용할 수 있는 서비스를 GameLift로 변경해야 합니다. 앞 서 생성한 역할을 선택하여 Trust relationships 탭을 선택하고, Edit trust relationship 버튼을 클릭합니다.
26. 아래와 같이 ec2로 되어 있던 부분을 gamelift로 변경하고 update trust policy 버튼을 클릭하여 저장합니다.

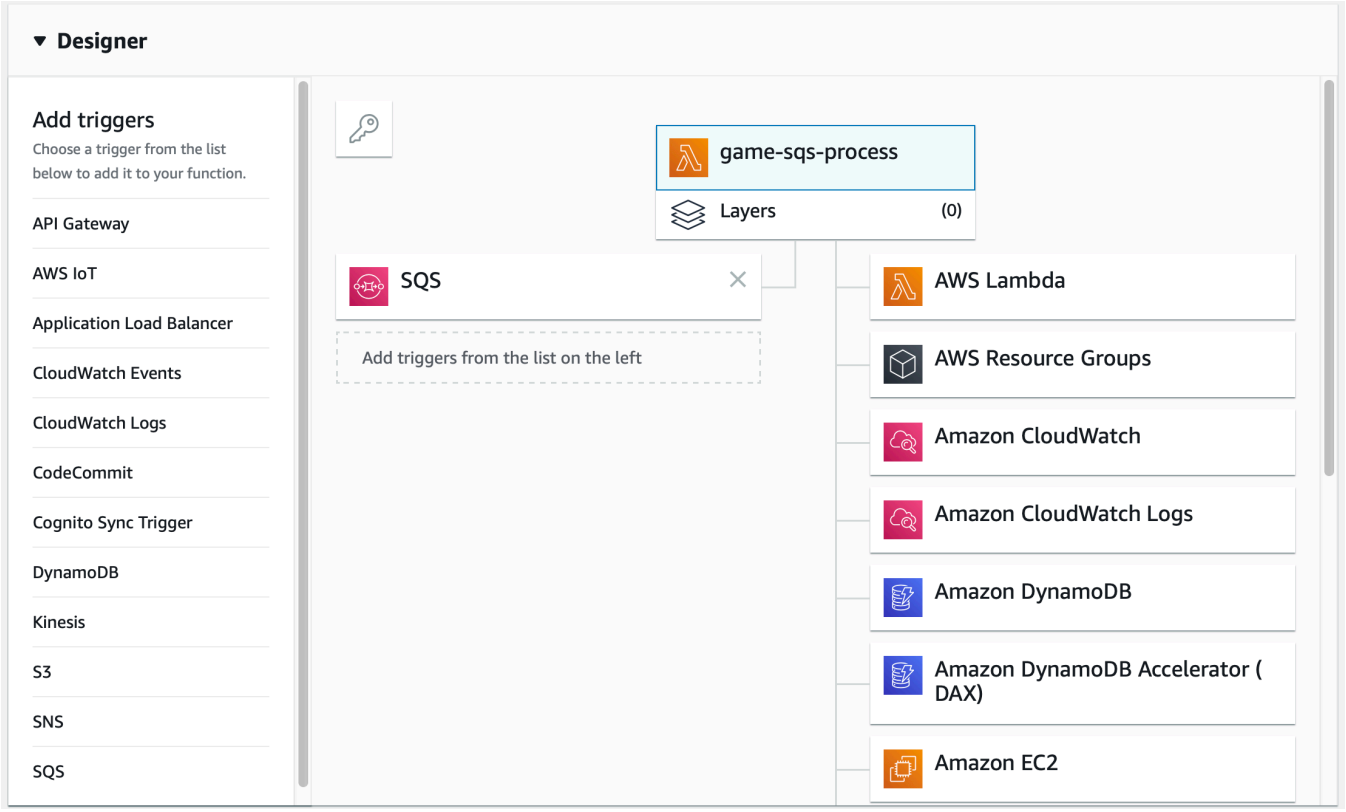


Section 2: Lambda Lambda Lambda

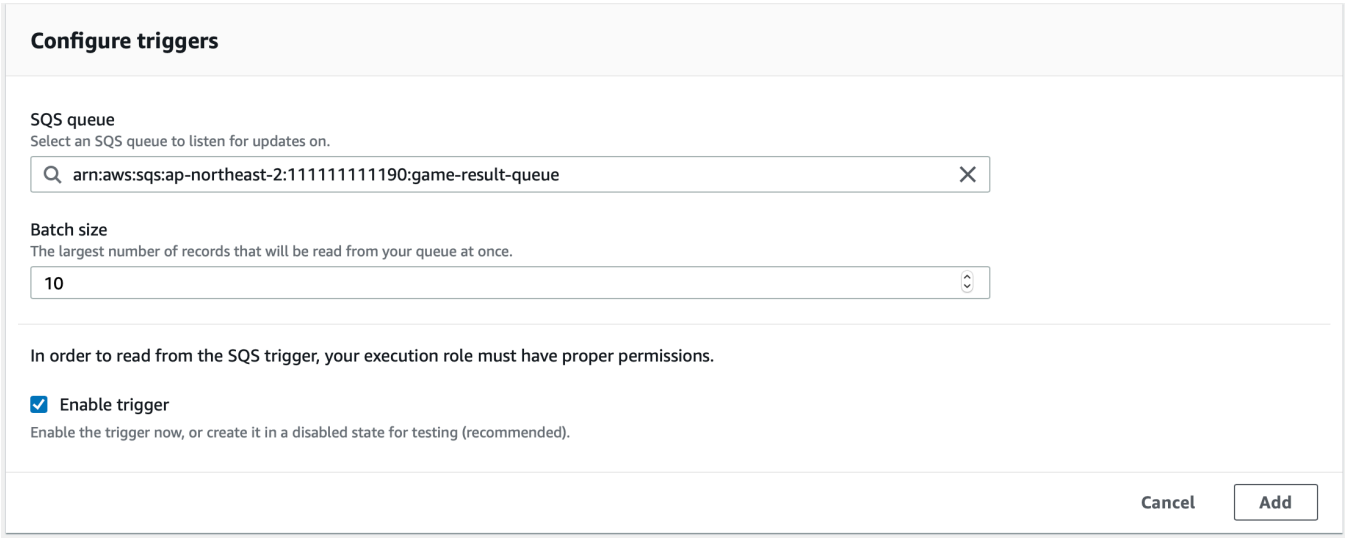
이전 섹션까지 우리는 full game stack의 가장 기초적인 서비스들을 구성했습니다. 지금부터는 사용자가 게임을 즐기고 게임 결과를 처리할 Lambda 를 구성하겠습니다.

이번 랩에서는 총 3개의 Lambda 함수를 생성할 것입니다.

1. 콘솔에서 **Lambda** 메뉴로 이동합니다. https://console.aws.amazon.com/lambda
2. **Create function** 버튼을 클릭하여 첫번째 함수 생성을 시작합니다.
3. **Author from scratch** 메뉴를 선택하여 빈 함수를 우선 생성합니다.
4. **Name** 항목에는 **game-sqs-process**를 입력합니다.
5. **Runtime**은 Python 2.7을 선택합니다.
6. **Permissions** 항목에서 **Role** 은 Use an existing role을 선택하고 기존에 만들어둔 **Gomok-game-sqs-process**를 선택하고 Create function을 실행합니다.
7. 생성이 완료되면 Designer그룹의 Add triggers 하단에 SQS(Simple Queue Service)를 Lambda의 실행 트리거로 선택합니다. 그러면 하단에 Configure triggers 그룹이 생성됩니다.



8. SQS queue는 앞서 생성한 SQS의 arn이 선택되어 있는 것을 확인합니다. Enable trigger가 선택된 것을 확인하고 Add 버튼을 선택합니다.



9. 다시 상단의 Designer 그룹으로 돌아와 아래의 생성할 함수의 정보를 참고하여 Lambda 함수를 작성합니다. (GameLiftFlexHol.zip 파일을 압축해제하면 Lambda 폴더 밑에 있는 GameResultProcessing.py에서 표시된 변수를 적합하게 수정해서 넣어주는 과정입니다)

   Code: GameResultProcessing.py 파일의 내용을 Copy&Paste합니다. 코드 내의 **region_name** 부분은 여러분이 랩을 수행하는 리전으로 되어 있는지 확인합니다. (예: ap-northeast-2)

   Advanced settings: 128MB Memory and **0 min 30** sec **timeout**

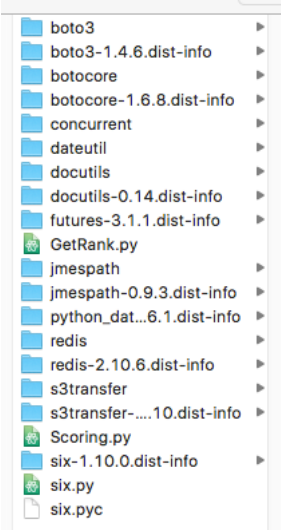
   region_name : 여러분이 사용하시는 리전의 코드입니다.

10. **Save** 버튼을 클릭하여 함수를 생성합니다. 생성한 함수는 SQS 에 기록된 게임 결과 점수를 읽어와 DynamoDB 에 업데이트하는 역할을 수행합니다.

두 번째 Lambda 부터는 앞의 Lambda와는 다르게 Python 배포 패키지를 통째로 업로드 하여 Lambda 함수를 생성할 것입니다. 이전에 생성한 함수는 Python 표준 SDK만을 사용하기 때문에 인라인 편집기를 사용했지만, 다른 Lambda 함수들은 Redis 라이브러리를 참조하기 때문에 배포 패키지를 업로드 하여 함수를 생성할 것입니다. Lambda 함수를 Python 배포 패키지를 업로드하여 생성해볼 수 있는 아주 좋은 기회입니다.

소스 코드를 작성(수정)한 후 배포 패키지 형태로 묶은 다음(LambdaDeploy.zip 파일) 이를 업로드하여 Lambda 함수를 생성할 것입니다. (만약 배포 패키지를 어떻게 직접 만드는지 알고 싶으시다면 Appendix A를 참조하세요.)

1. GameLiftFlexHol.zip 파일의 Lambda 폴더 하위에 LambdaDeploy 폴더에 GetRank.py와 Scoring.py라는 두 개의 Python 파일이 보일 것입니다.

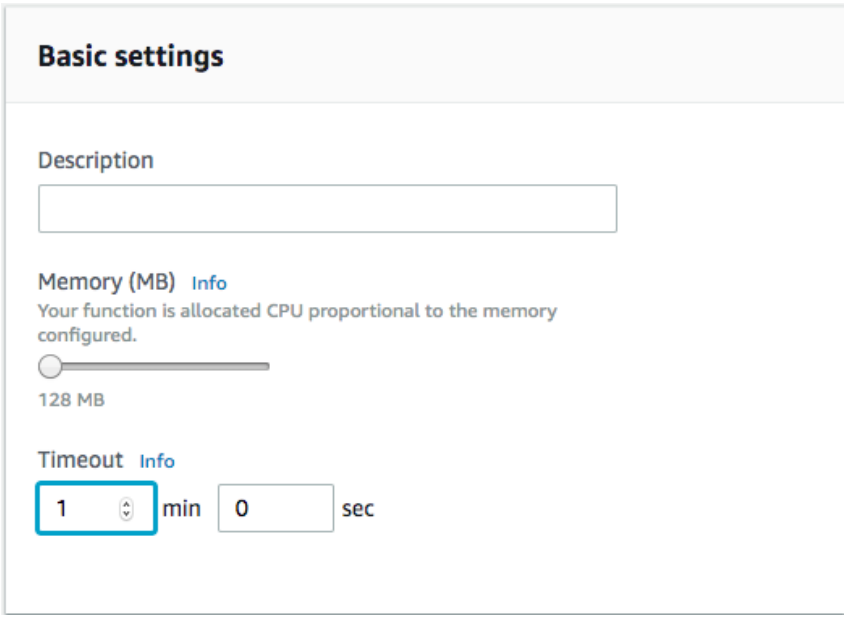


2. 우선 **Scoring.py** 파일을 열고 6번째 줄의 Redis 클러스터 정보를 여러분이 생성한 ElastiCache의 Endpoint로 수정합니다.
3. 두 번째로 **GetRank.py** 파일을 열고 6번째 줄의 Redis 클러스터 정보를 생성한 ElastiCache의 Endpoint로 수정합니다.
4. 두 개의 파일 모두 저장한 뒤, 다시 LambdaDeploy.zip으로 압축해줍니다. (참고: "LambdaDeploy"폴더가 압축파일에 포함되면 안됩니다. 즉, GetRank.py 및 Scoring.py파일은 압축파일 내의 루트경로에 있어야 합니다.)

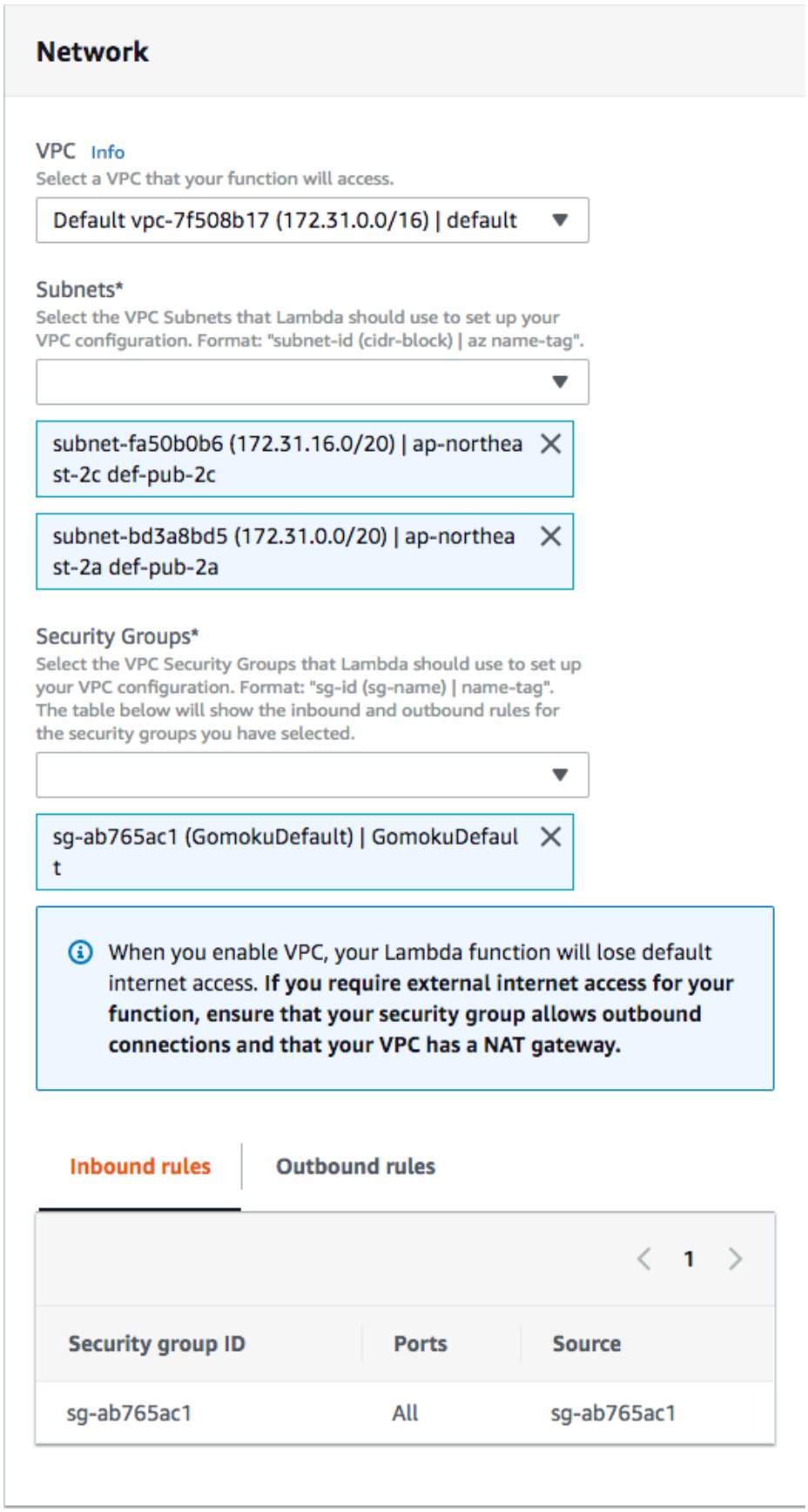
배포 패키지를 완성하였습니다. 이제 이를 이용하여 Lambda 함수를 생성하겠습니다.

1. 앞 서와 동일하게 **Author from scratch** 메뉴를 선택하여 함수 생성을 시작합니다.
2. Name은 **game-rank-update**으로 지정하고, Runtime은 **Python 2.7**로 선택하고, Role은 **Gomok-game-rank-update**을 선택하고 Create function을 누릅니다.
3. Function code그룹에서 Code entry type을 **Upload a .ZIP file**로 선택하고 **LambdaDeploy.zip**을 업로드합니다.

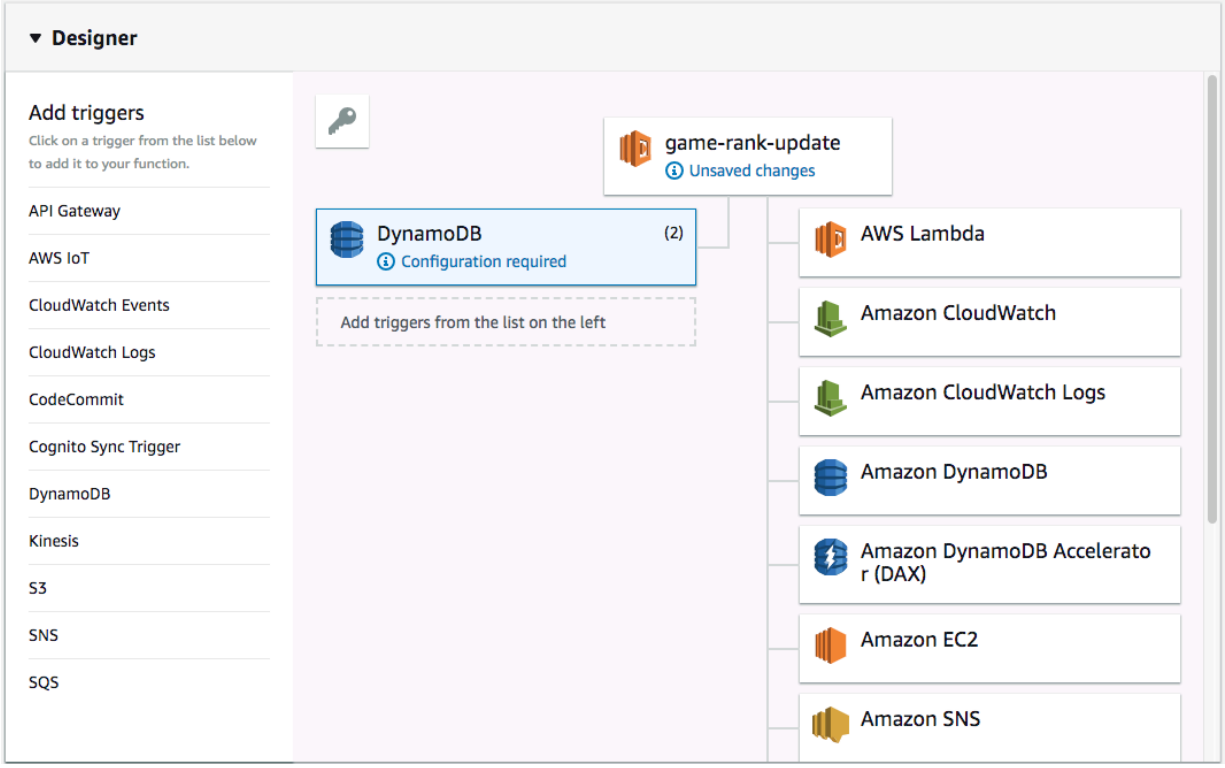
4. Handler 항목에서는 **Scoring.handler** 를 입력합니다.
5. 하단의 Basic settings 그룹에서 Timeout을 1분으로 변경합니다.



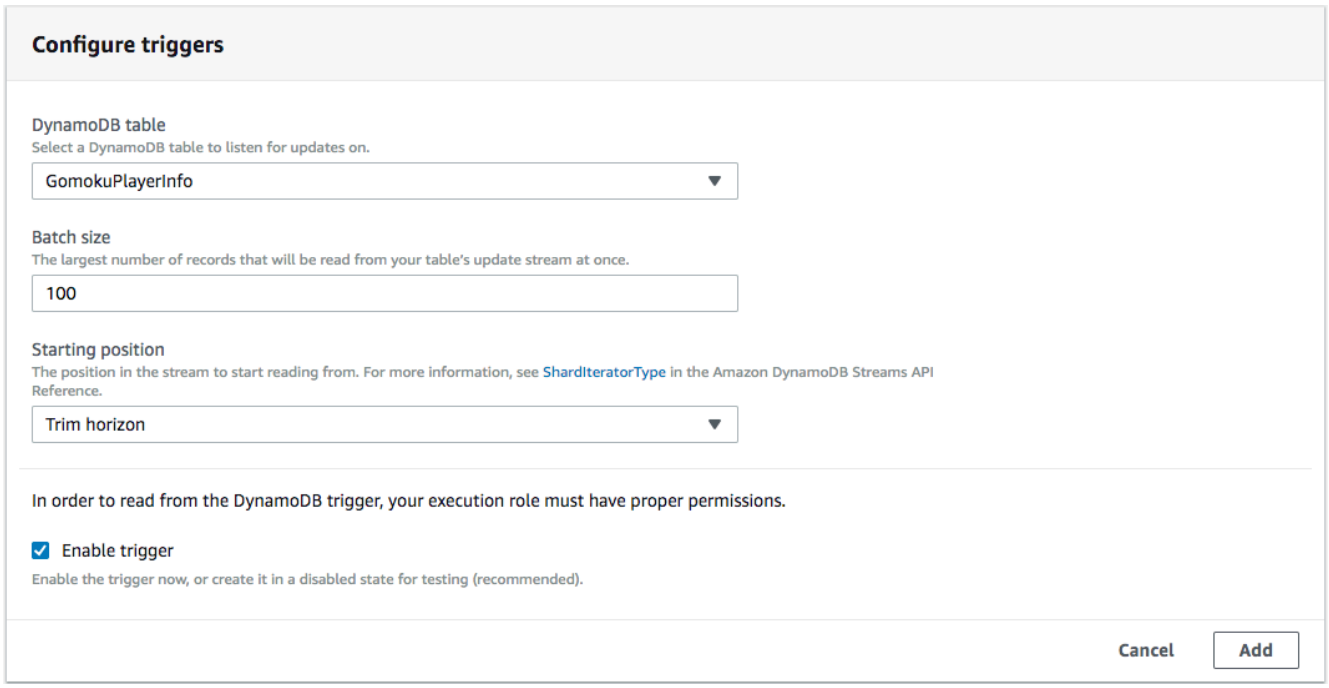
6. Network 항목에서 VPC를 기존에 생성한 ElastiCache가 있는 VPC로 선택합니다. subnet항목은 모두 선택하여 주시고, Security Group은 이전에 생성했던 GomokuDefault로 선택하여 주시기 바랍니다.



7. 상단의 Designer 그룹의 Add triggers에서 **DynamoDB**을 선택합니다.



8. 하단에 Configure triggers그룹에서 **GomokuPlayerInfo** DynamoDB 테이블을 Trigger 테이블로 사용할 것입니다. 다른 부분은 기본값을 유지하고 Starting position은 **Trim horizon**을 선택합니다. **Enable trigger**를 체크한 뒤 **Add** 버튼을 클릭합니다.

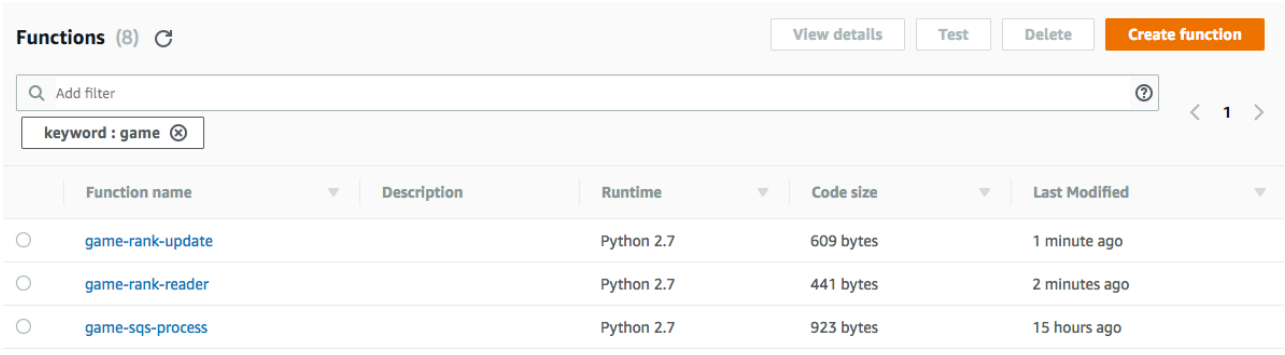


9. 위의 과정이 완료되면 오른쪽 상단의 **Save**를 눌러 함수 생성을 완료합니다. (이 단계에서 함수 Test를 누르게 되면 아직 실패합니다.)

마지막으로 세 번째 Lambda 함수를 생성하겠습니다.

1. 세 번째 함수도 **Author from scratch** 를 선택하여 생성합니다.
2. Name항목에서 **game-rank-reader**를 입력하고, Runtime을 Python 2.7로 설정하고 Role을 이전에 만들어둔 **Gomok-game-rank-reader**를 선택합니다.
3. Create function을 누른 후Code entry type에서 Upload a .ZIP file을 선택하여 LambdaDeploy.zip을 업로드합니다.
4. Handler 부분에서 **GetRank.handler**를 입력합니다.
5. Basic settings 부분에서 Timeout을 1 min으로 설정합니다.
   Network 부분에서 이전 함수 동일하게 VPC 를 기존에 생성한 ElastiCache 가 있는 VPC 로 선택합니다. subnet 항목은 모두 선택하여 주시고, Security Group 은 이전에 생성했던 GomokuDefault 로 선택하여 주시기 바랍니다.
6. 이제 Save 를 눌러 Lambda 함수를 생성합니다.

여기까지 완료했다면 다음 스크린캡처와 같이 3개의 Lambda 함수가 생성된 것을 확인할 수 있습니다.



이제 모든 Lambda 함수는 준비되었습니다. 지금부터는 이 함수를 실행할 방법이 필요합니다. 처음 생성한 두 함수는 SQS와 DynamoDB에 의해 호출될 것입니다. 하지만 마지막 함수는 trigger를 설정하지 않았습니다. 어떻게 해야 해당 함수를 호출할 수 있을까요? 여기에서 API Gateway를 사용할 것입니다. 이 부분은 다음 섹션에서 내용이 이어집니다.

여기까지 준비가 되면 Section 1과 2에서 만들어진 여러 요소들이 어떻게 동작하는지를 테스트해 볼 수 있습니다. 간략하게 설명하면 전체 흐름을 보면 SQS에 데이터가 삽입되면 만들어진 Lambda함수가 실행되면서 해당 SQS데이터를 DynamoDB에 업데이트합니다. Dynamo DB에 입력된 데이터는 DynamoDB Stream의 Trigger를 통하여 다른 Lambda함수가 실행되어 Redis Cache가 업데이트 되는 형태를 가지게 됩니다.

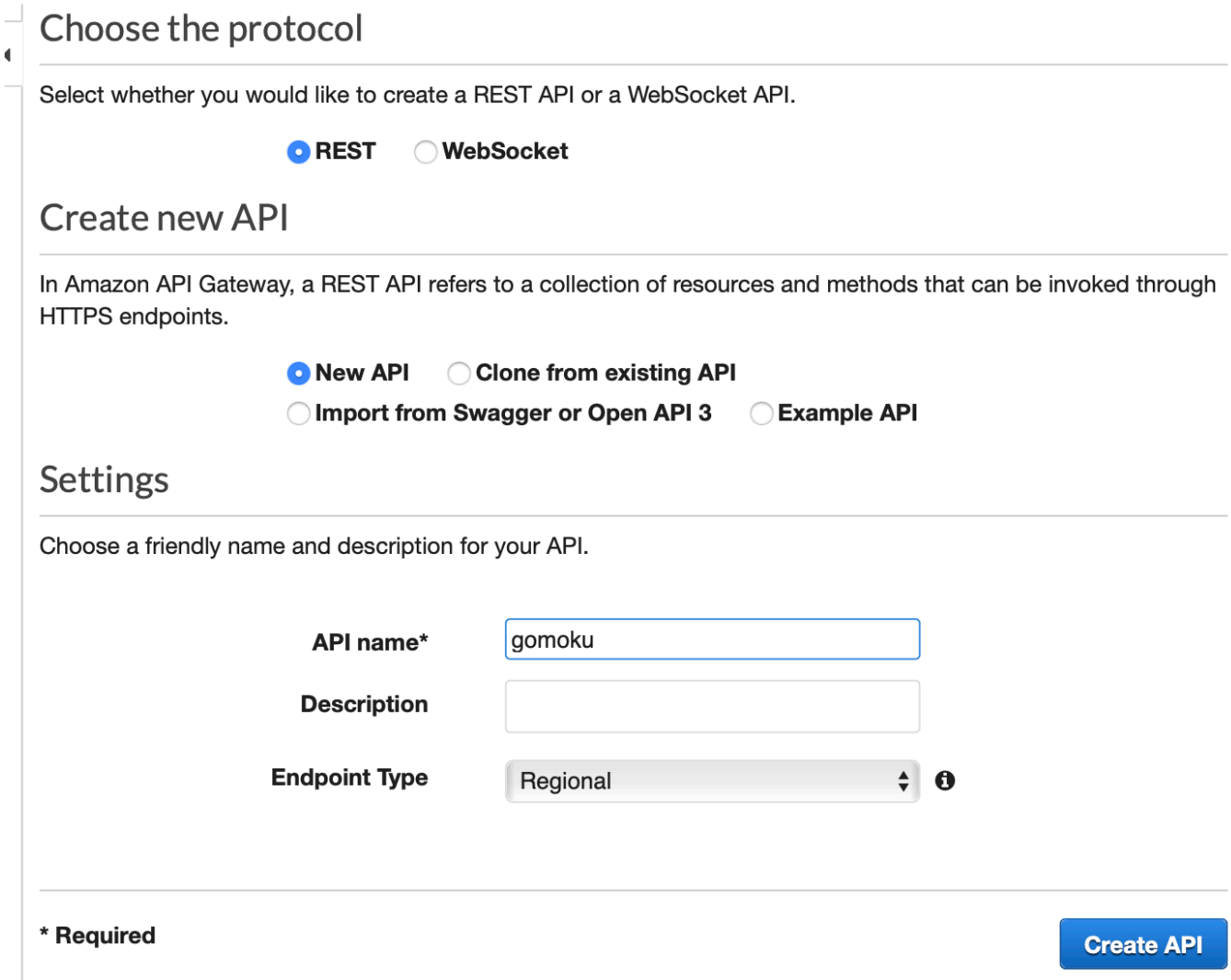
간단하게 Redis Cache의 내용을 확인할수는 없으나 전체 흐름이 정상적으로 동작하는지 여부를 테스트 데이터의 입력과 CloudWatch를 통하여 확인해보도록 하겠습니다.

1. Console을 통하여 SQS를 열어줍니다. https://console.aws.amazon.com/sqs
2. 위에서 생성한 **game-result-queue**를 선택하고 화면 위의 **Queue Actions**버튼을 눌러줍니다.
3. 나오는 메뉴 중에서 **Send Message**를 선택합니다.
4. 나오는 텍스트 상자에 아래의 JSON을 넣어줍니다. (특수 기호 때문에 붙여 넣기보다는 직접 입력을 추천합니다. 대소문자 구분에 주의하십시요. 정상 동작하지 않을 경우, CloudWatch Logs에 남아 있는 lambda 출력을 확인하세요.)
   { "PlayerName" : "SomeName" , "WinDiff" : 1, "LoseDiff" : 0, "ScoreDiff" : 100 }
5. 그리고 **Send Message**버튼을 눌러줍니다.
6. 이제 DynamoDB콘솔로 옮겨가서 https://console.aws.amazon.com/dynamodb **GomokuPlayerInfo** 테이블을 열어줍니다. **Items** 탭을 보면 위에서 넣어준 JSON데이터에 해당하는 항목이 반영되어 있는 것을 확인 할 수 있습니다. (이미 같은 PlayerName에 해당하는 항목이 있다면 Win이 1증가하고 Score가 100증가했을 것입니다.)

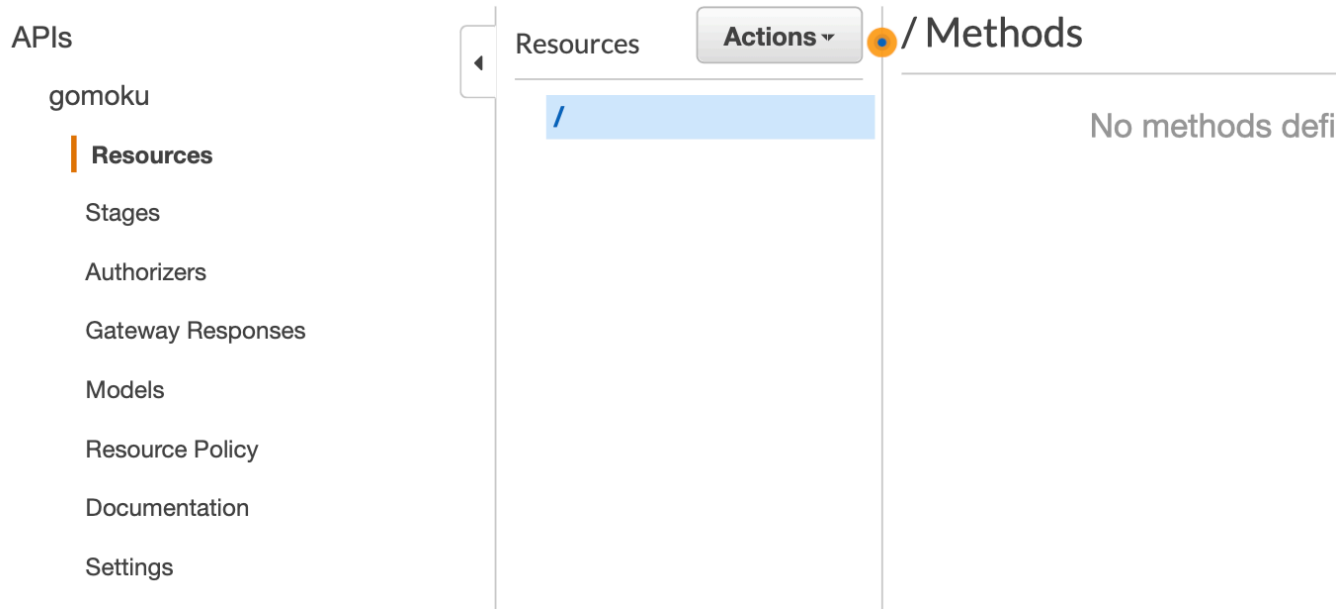
Section 3: Face the users with API Gateway and S3

우리의 Lambda 함수 중 한가지가 API Gateway를 필요로 한다는 것을 알고 있습니다. 이것은 개발 중의 애플리케이션의 Lambda 함수를 실행할 일종의 "Gateway" 역할을 하기 때문에 API Gateway라 이름이 붙여졌습니다.

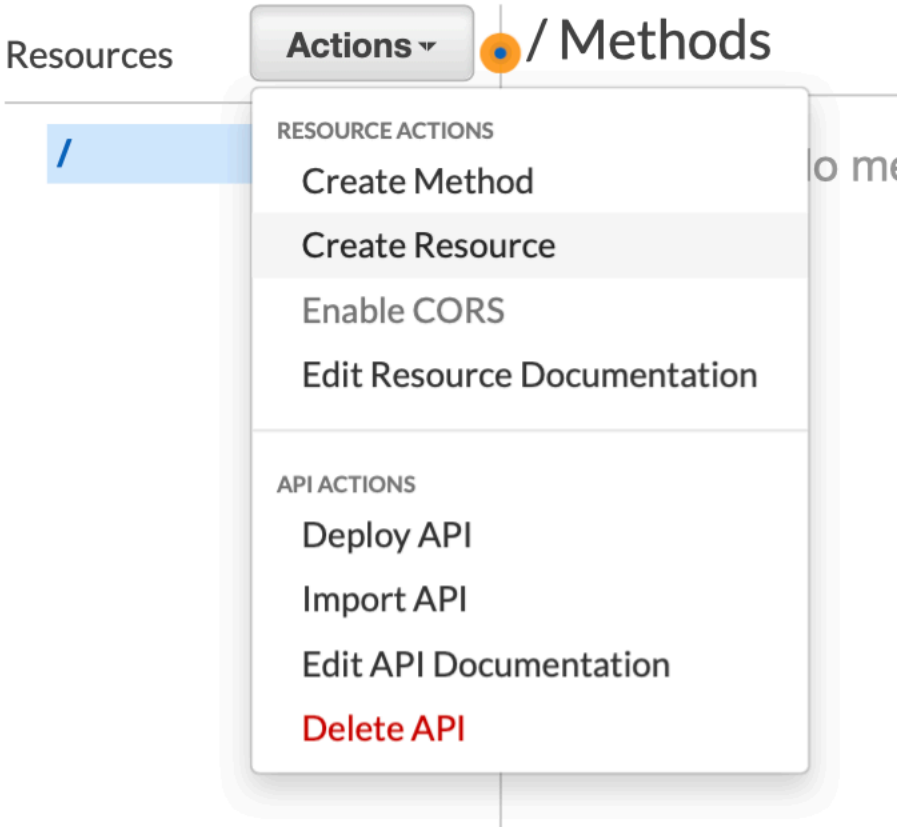
1. 우선 콘솔에서 API Gateway 메뉴로 이동합니다.
   https://console.aws.amazon.com/apigateway
2. **Create API**를 선택하고, API name은 **gomoku**을 입력한 뒤 **Create API** 버튼을 클릭합니다.



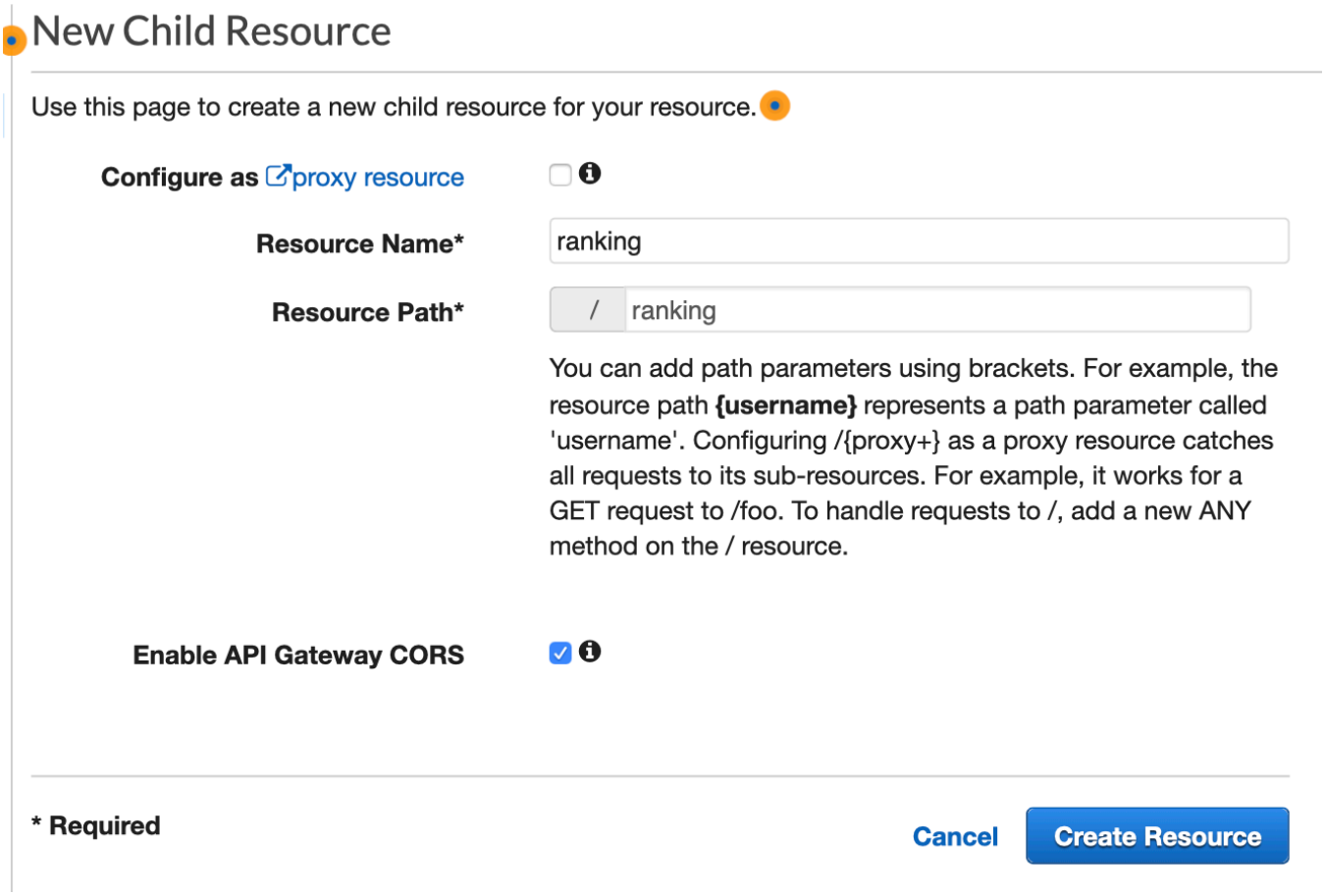
3. API가 생성된 뒤에는 다음과 같은 빈 화면이 보일 것입니다.



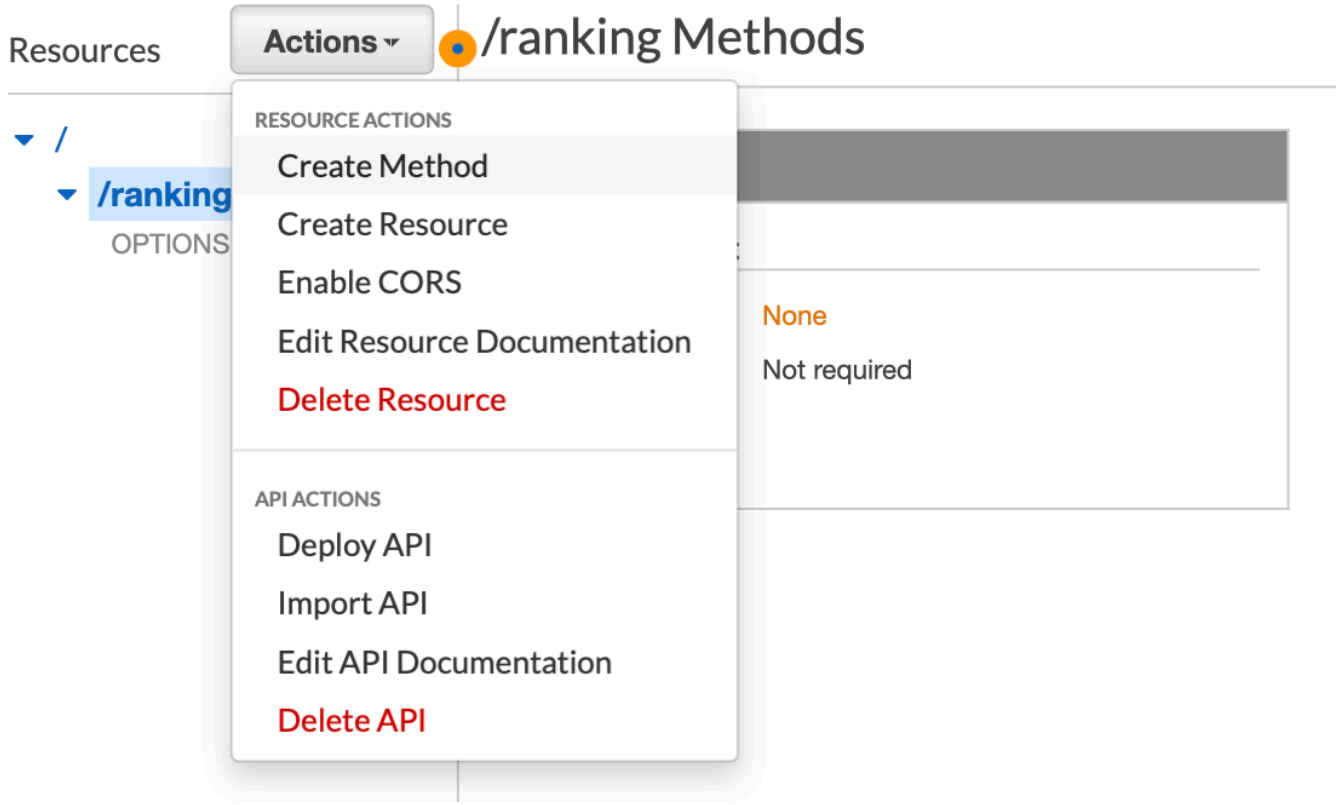
4. 우선 Resource를 만들고 그 Resource에 Method를 생성합니다. **Actions** 버튼을 클릭하고 resource를 만듭니다.



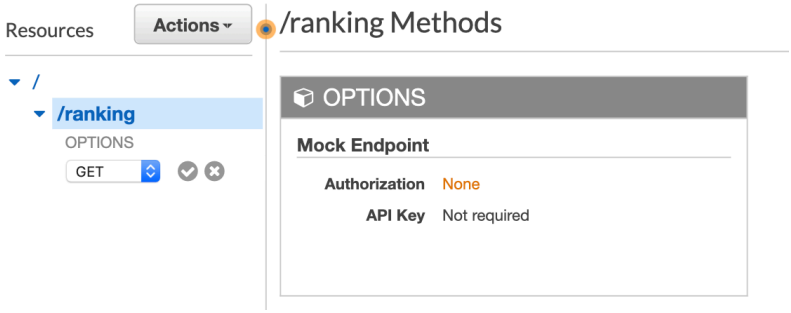
5. 아래와 같이 Resource Name에 ranking을 입력하고 Enable API Gateway CORS를 선택합니다. Create Resource를 클릭합니다.



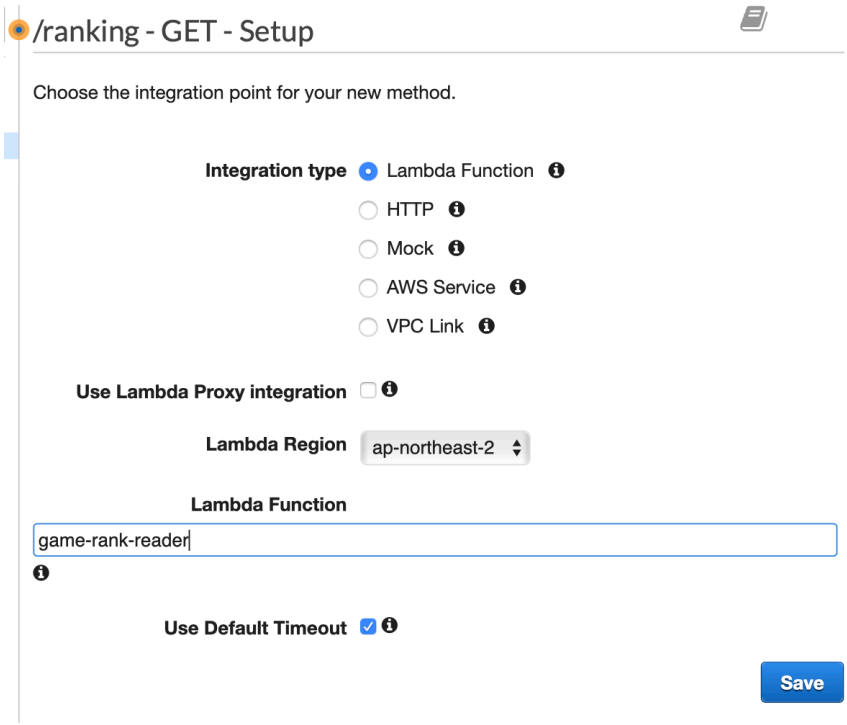
6. 다음은 새로운 Method를 생성하는 것입니다. **Actions** 버튼을 클릭한 뒤 **Create Method** 메뉴를 선택합니다.



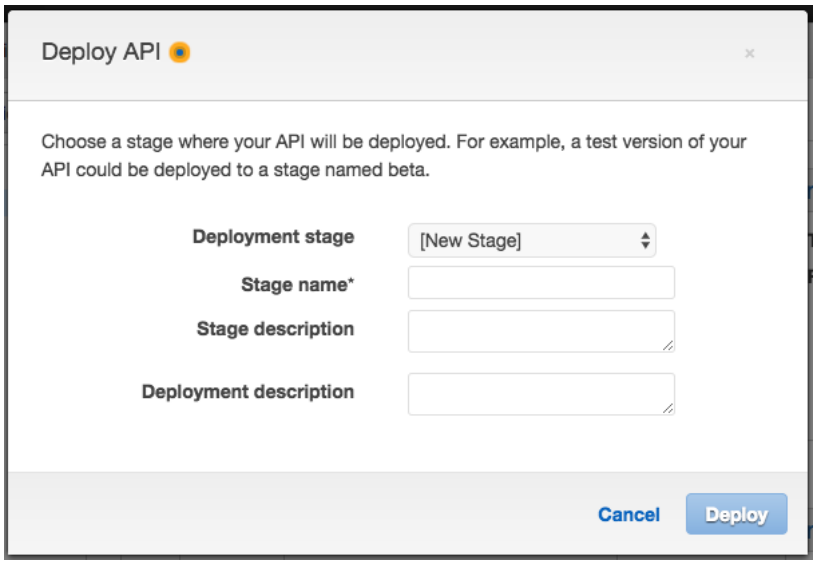
7. 아래 작은 리스트박스가 보일 것입니다. **GET**을 선택한 뒤 옆의 체크 버튼을 클릭합니다.



8. GET 메소드의 상세 설정에서 Integration type은 **Lambda Function**을 선택하고 Lambda Region에 실습을 진행 중인 Region을 선택합니다. Lambda Function에는 앞서 생성한 **game-rank-reader**를 선택한 뒤 **Save** 버튼을 클릭합니다.

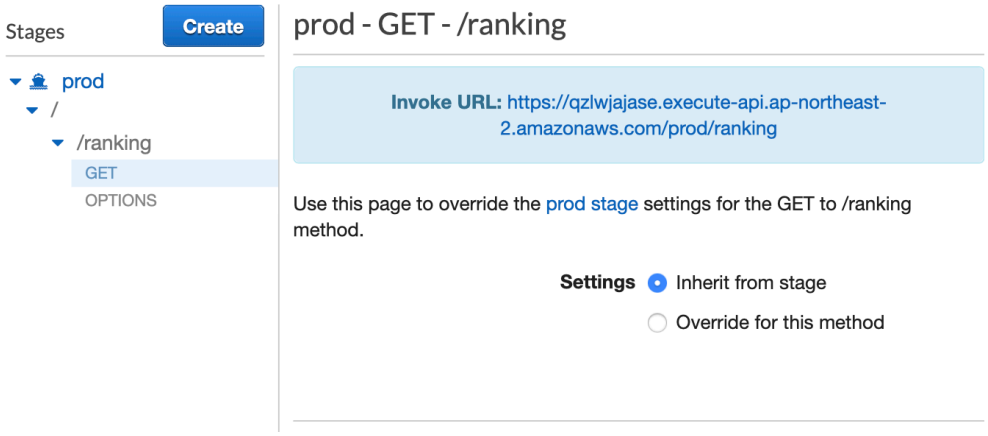


9. API 구성이 되었습니다. 이제 prod 단계에 배포를 해보겠습니다. **Actions** 버튼을 클릭하고 **Deploy API** 메뉴를 클릭합니다.



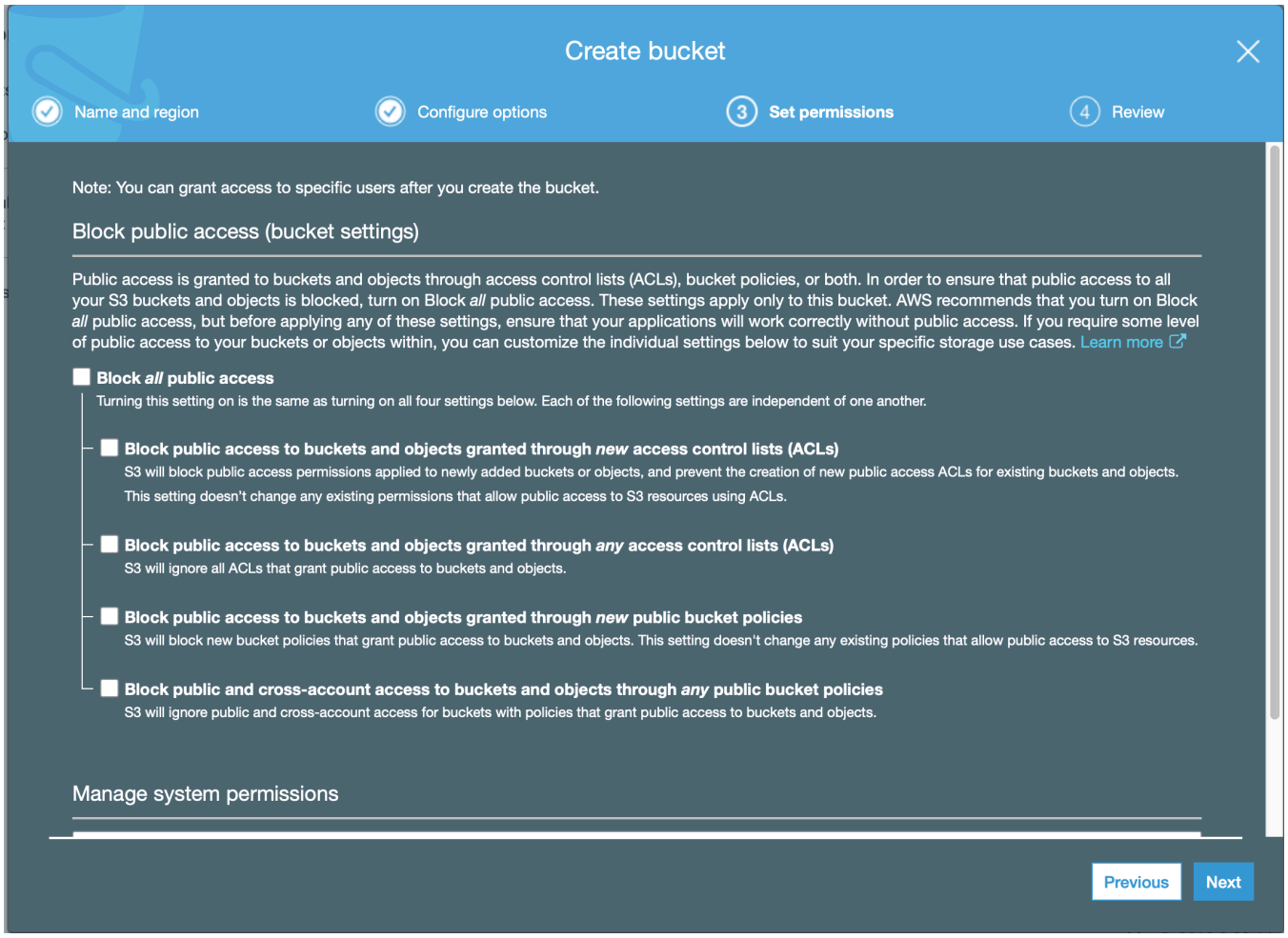
10. **[New Stage]** 를 선택하고 Stage name에는 **prod**를 입력합니다. **Deploy** 버튼을 클릭하여 진행합니다.
11. 완료되면 다음 스크린 캡처와 같이 Stage 구성이 된 것을 확인할 수 있습니다. Prod 배포의 /ranking 메뉴 하단의 GET을 선택하면 나오는 Invoke URL을 기록해둡니다. 이

후 S3를 이용한 정적 웹페이지 구성에 사용됩니다.

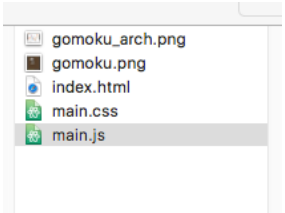


이제 S3가 호스팅하는 웹 사이트를 생성할 것입니다. 버킷에 Ranking board html파일과 Javascript 파일을 업로드하는 것만으로 쉽게 웹 사이트를 호스팅할 수 있습니다.

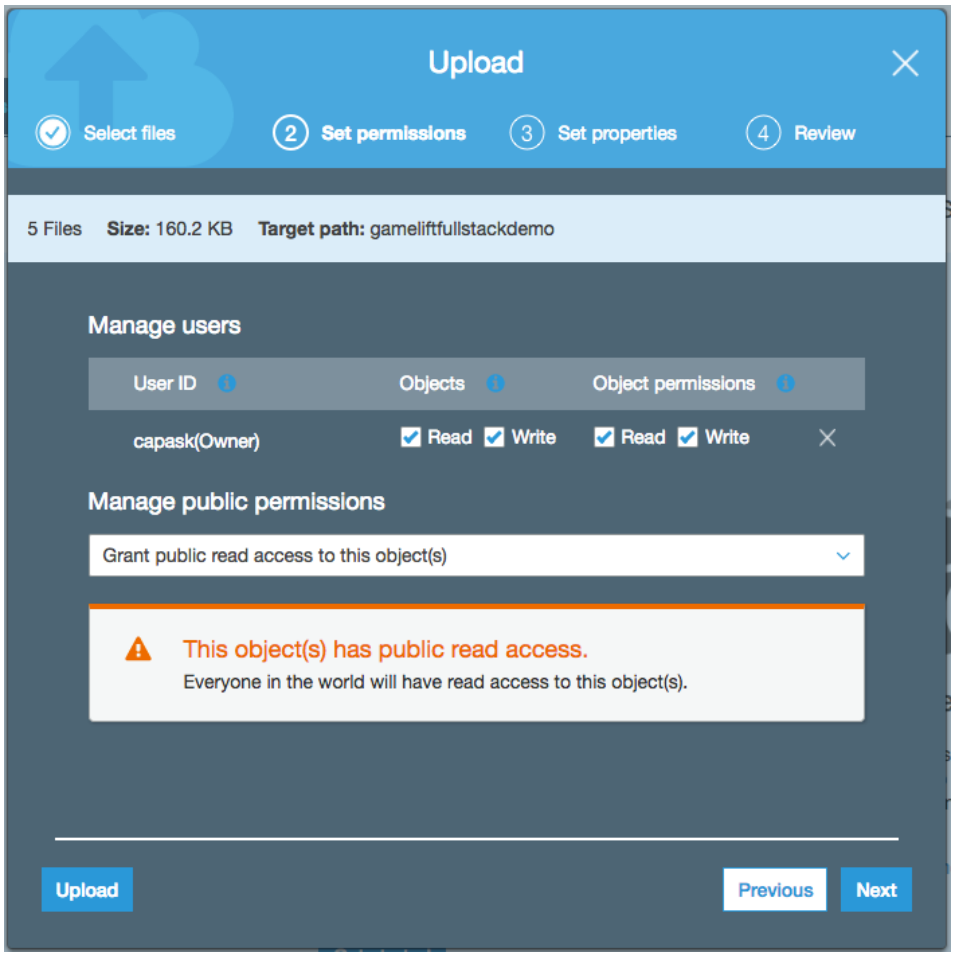
1. S3 서비스로 이동하여 웹 사이트에 사용할 S3 버킷을 생성합니다. 버킷 생성 시, public object를 허용하기 위해 Block all public access를 uncheck합니다.



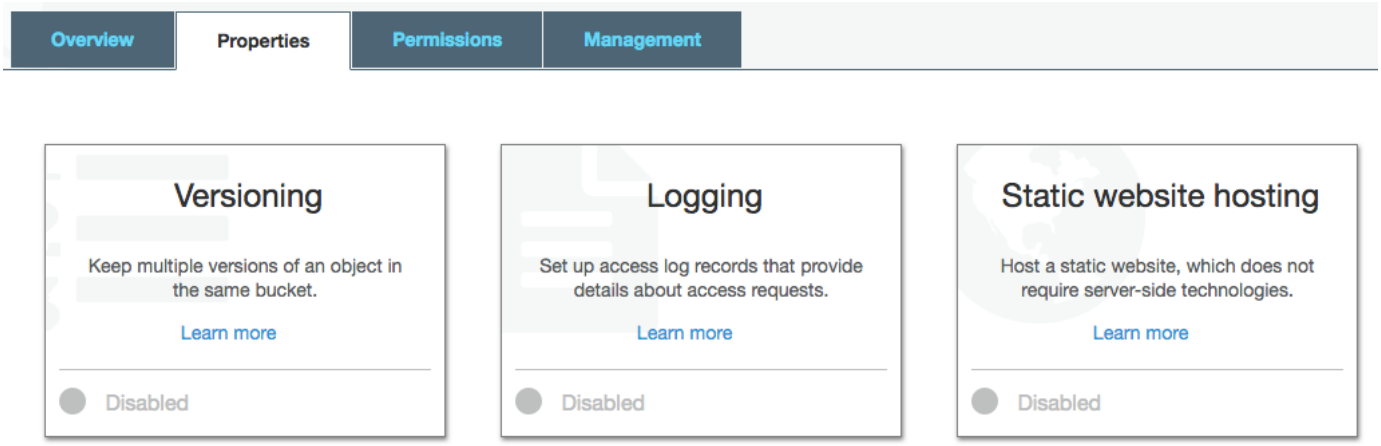
2. 실습에 사용되는 파일 중 web 디렉토리가 있을 것입니다. 디렉토리 내의 **main.js** 파일을 텍스트 편집기로 엽니다.
3. 48번째 줄의 API Endpoint에 위에서 생성한 API Gateway의 Invoke URL로 수정합니다.
4. 수정한 main.js를 저장한 뒤 web 디렉토리의 파일을 전부 앞서 생성한 버킷에 업로드해줍니다.



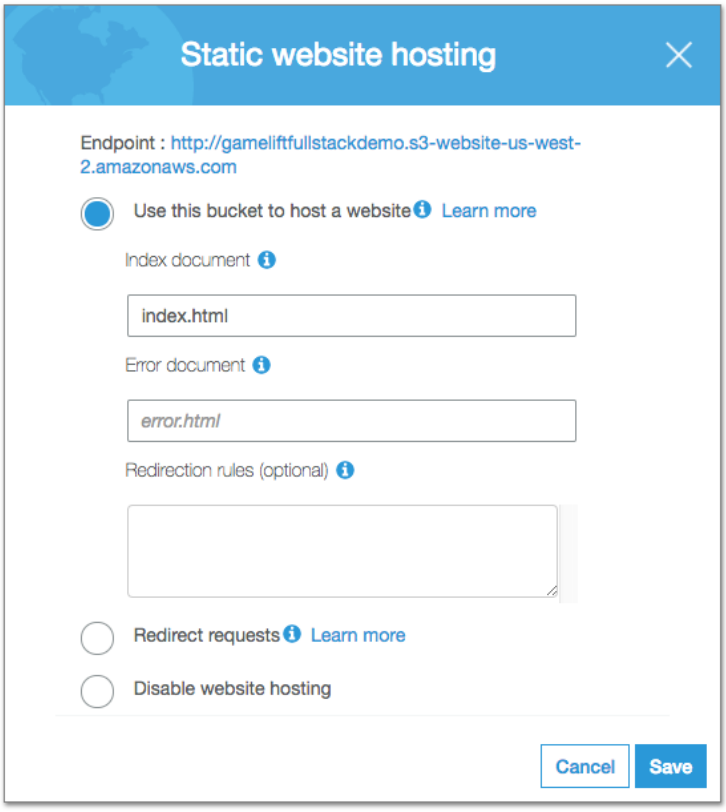
5. 업로드할 때 Public read access 권한을 부여합니다.



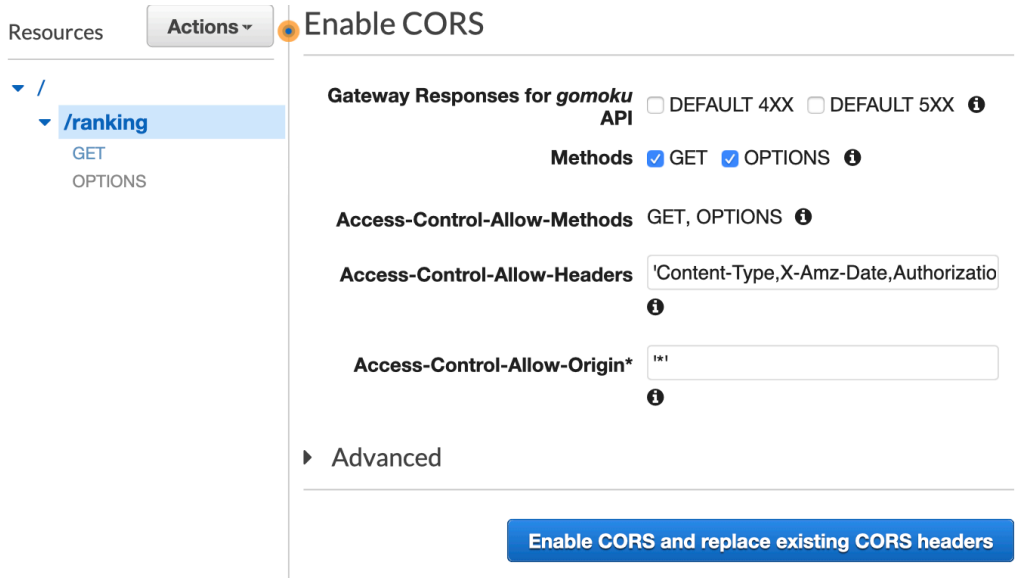
6. 버킷에서 Static website hosting을 활성화해주어야 합니다. 버킷의 **Properties** 탭으로 이동합니다.



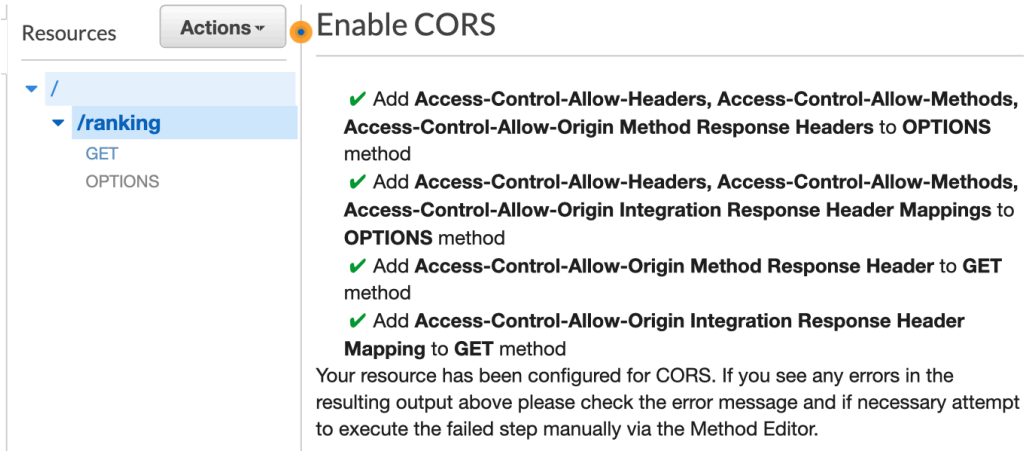
7. **Static website hosting**을 활성화합니다. Index document에는 **index.html**을 입력한 뒤 **Save** 버튼을 클릭합니다.



8. 파일 업로드를 완료하면 Static website hosting의 Endpoint로 접속하여 단순한 웹 페이지를 확인할 수 있습니다.
9. API Gateway로 돌아와서 CORS 설정을 해주어야 합니다. 이를 통해 Ranking board 정보를 웹 페이지에서 읽어 올 수 있게 됩니다.
10. API Gateway로 돌아와서 **Actions**버튼을 클릭하고 **Enable CORS** 옵션을 선택합니다.
11. CORS 페이지에서 Access-Control-Allow-Origin 에 static website URL로 수정해줍니다. (URL 뒤의 / 가 영향을 미칠 수 있기 때문에 확실하지 않으면 기본 값인 * 로 진행합니다.)



12. **Enable CORS** 버튼을 클릭하여 진행합니다.
13. 정상적으로 완료되었다면 왼쪽 탭에 OPTIONS 가 추가된 것을 확인할 수 있습니다.



14. 마지막으로 **Actions** 메뉴의 **Deploy API** 버튼을 클릭하여 prod 단계에 배포합니다.

배포한 Static Page를 웹브라우저에서 열어주거나 10번에서 기록한 API 의 Invoke URL을 브라우저에서 열어주시면 Section 2에서 테스트로 입력했던 데이터가 표시되거나 JSON(직접 Invoke URL을 열었을 경우) 으로 표기 되는 것을 확인 할 수 있습니다.

Section 4: Putting all together in GameLift

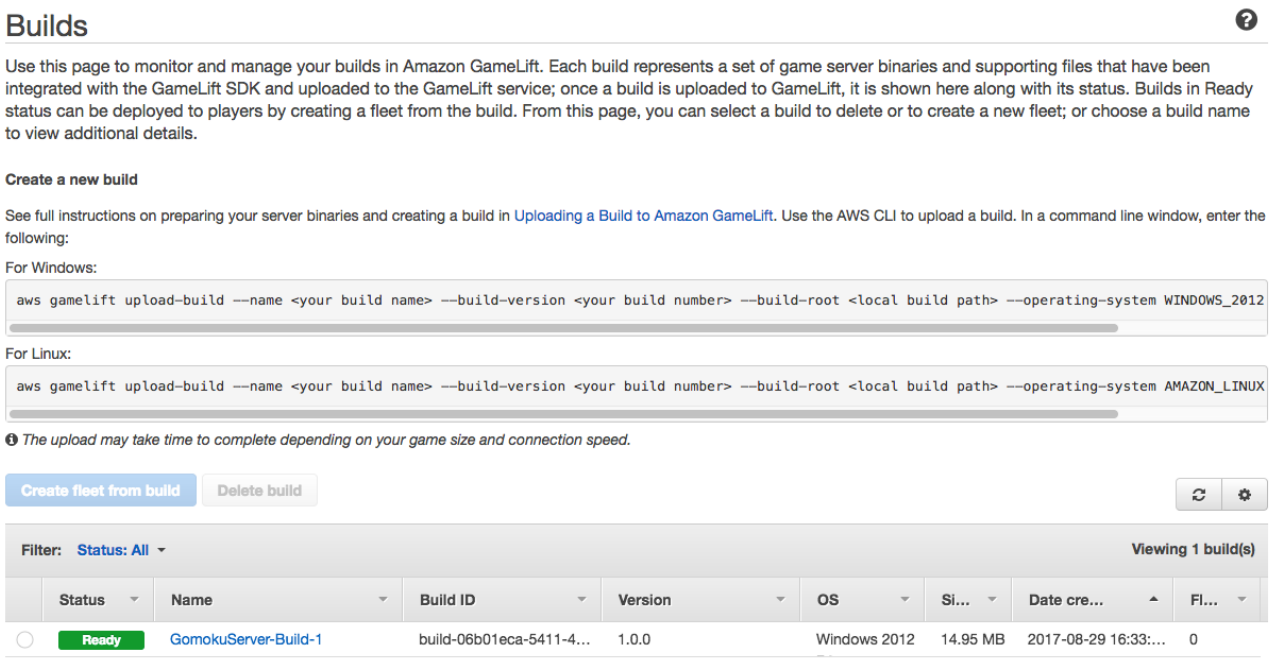
이제 서버 바이너리를 Gamelift 서비스와 함께 동작하도록 구성하는 작업을 시작하겠습니다. 서버 바이너리는 컴파일 된 형태로 제공되었기 때문에 추가 작업이 필요하지는 않습니다. 하지만 직접 컴파일하여 생성한 바이너리를 소스 코드로 사용하실 수도 있습니다.

1. 이번 실습에는 이미 컴파일된 파일이 준비되어 있지만, 직접 컴파일 하고 싶으실 경우 Appendix B를 참조할 수 있습니다.
2. GomokuServer.exe, aws-cpp-sdk-*.dll, config.ini, install.bat, aws-cpp-sdk-gamelift-server.dll 파일들이 Binaries/GomokServer 폴더에서 확인할 수 있습니다.
3. 텍스트 편집기를 통하여 config.ini 파일을 수정합니다. 파일에는 총 3군데 구성 요소가 있습니다. SQS_REGION는 SQS를 생성한 Region입니다. (예: ap-northeast-2). SQS_ENDPOINT 에는 SQS의 Endpoint를 입력합니다.
4. ROLE_ARN은 섹션1에서 마지막으로 만들었던 GameLift Fleet을 위한 ROLE_ARN을 입력합니다. (아래의 스크릿샷 처럼 따옴표 없이 입력해주세요. Role Arn은 Fleet 내부의 인스턴스에서 얻어 올 수도 있으나 현재 데모용 Gomoku 게임서버에서 구현되어 있지 않으므로 명시적으로 구성해 줍니다.)

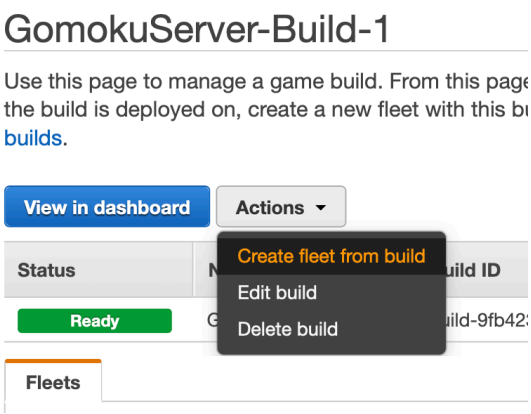
```
[config]

# GameResult SQS
SQS_REGION = ap-northeast-2
SQS_ENDPOINT = https://sqs.ap-northeast-2.amazonaws.com/592446325190/game-result-queue
ROLE_ARN = arn:aws:iam::592446325190:role/Gomoku-GameLiftFleetRole
```

5. Gamelift는업로드의 복잡성 때문에 현재 CLI를 통한 업로드 만을 지원합니다. AWS CLI 환경이 구성되어 있지 않다면 Appendix C를 참고하여 구성합니다.
6. GomokuServer 폴더에서 다음의 GameLift 업로드 명령어를 통해 빌드를 업로드 합니다.

   aws gamelift upload-build --name "GomokuServer-Build-1" --build-version "1.0.0" --build-root . --region ap-northeast-2
7. 업로드할 때 실습 Region을 올바르게 설정하였는지 확인해야 합니다.
8. 진행하는 중 콘솔의 GameLift 서비스로 가면 빌드가 업로드 되는 것을 확인할 수 있습니다. https://console.aws.amazon.com/gamelift



9. 콘솔의 빌드 페이지에서 방금 업로드한 빌드를 선택합니다. **Create fleet from build** 버튼을 클릭합니다. 이를 통해 게임 서버의 fleet을 생성하게 됩니다.



10. 다음의 정보를 입력합니다. 언급이 없는 부분은 기본값으로 진행합니다.

Name: GomokuGameServerFleet-1

Instance Type: C4.large

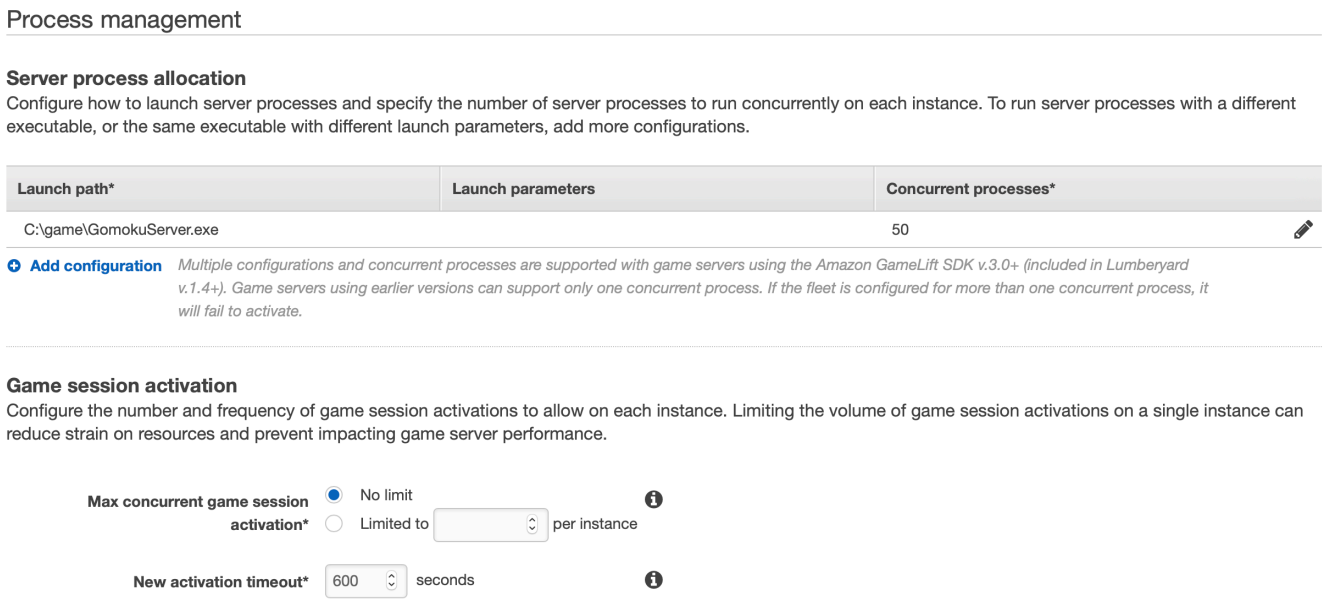
Fleet type: Spot

Instance Role ARN: 섹션1의 마지막에서 GameLift Fleet을 위해 만들었던 Role의 ARN

Launch path: GomokuServer.exe (직접 컴파일 했다면 컴파일한 바이너리명)

Concurrent processes: 50

우측의 녹색 체크 버튼을 클릭하여 확인합니다.



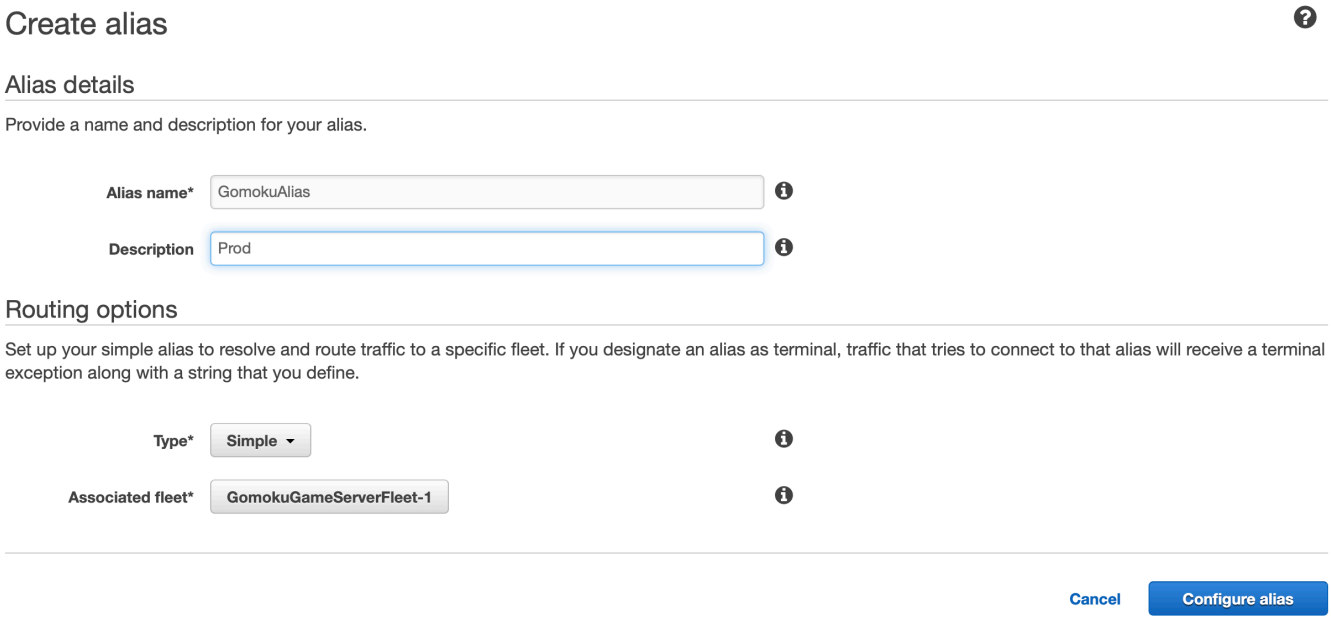
EC2 Port Settings: Port range; 49152-60000

Protocol; TCP, IP address range; 0.0.0.0/0 입력후 녹색 체크 버튼

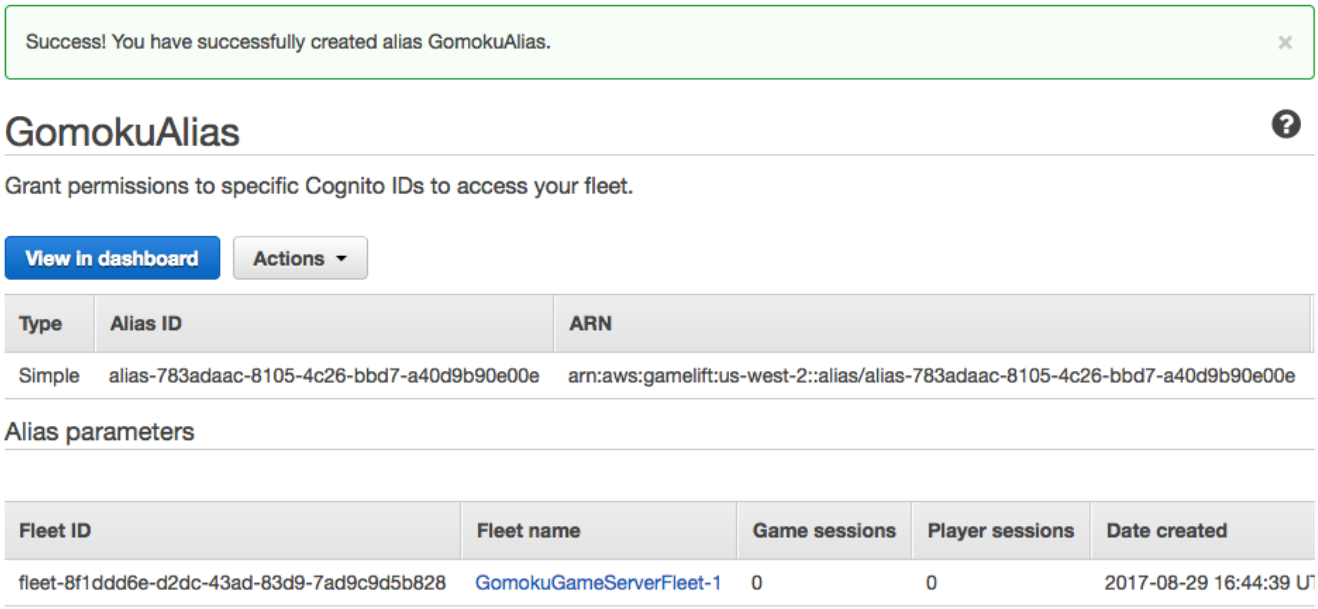
Initialize fleet를 클릭하고 잠시 기다립니다.

11. 생성이 시작되면 Fleet에서 다음과 같은 화면을 확인할 수 있습니다.

12. 생성이 완료되었다면 왼쪽의 파란 박스가 Active 상태의 녹색으로 변할 것입니다. 시간이 약 20분 정도 소요됩니다.
13. Fleet을 생성하는 동안에 Alias를 생성하겠습니다.
14. 메뉴에서 **Create alias** 옵션을 선택합니다. 그리고 Alias name과 Description을 입력해줍니다.
15. Routing options의 Type은 **Simple**을 선택하고, Associated fleet에서 **[Select fleet]**을 선택한 뒤 생성한 fleet을 선택합니다.



16. Fleet 상태는 진행 상태에 따라 Downloading/ Validating/ Activating 가 있습니다.



17. 실제로 Gamelift fleet을 이용하기 위하여 Alias ID를 사용하게 됩니다. 해당 Alias ID를 기록해둡니다.

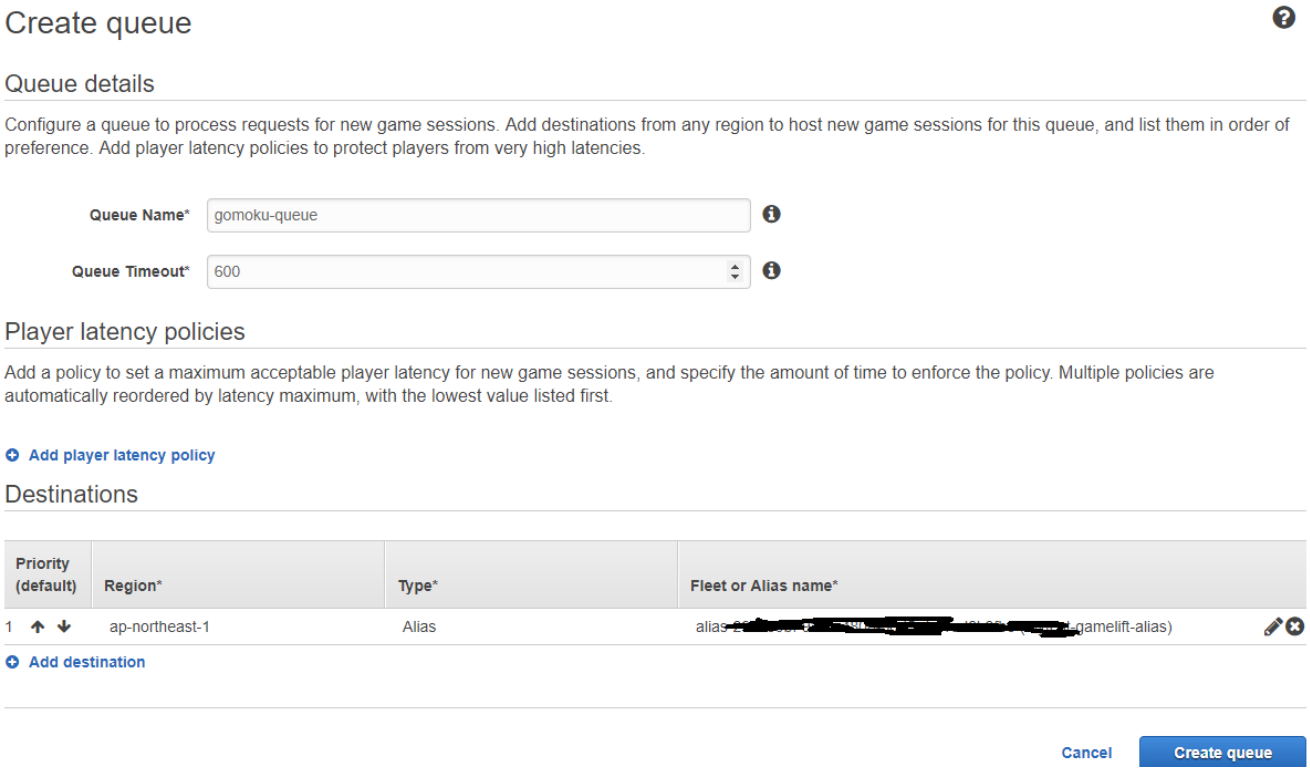
기본적으로 240분 동안 아무 활동이 없다면 Fleet은 인스턴스 0개로 스케일-인 합니다. 이번 실습을 진행하는 동안에는 문제가 없지만 조금 더 오래 실행하고 싶다면 auto scale parameter를 최소한 1로 변경해야 합니다. 아니면 직접 1개의 인스턴스를 실행하도록 override해주어야 합니다.

Section 5: Serverless FlexMatch 구성하기

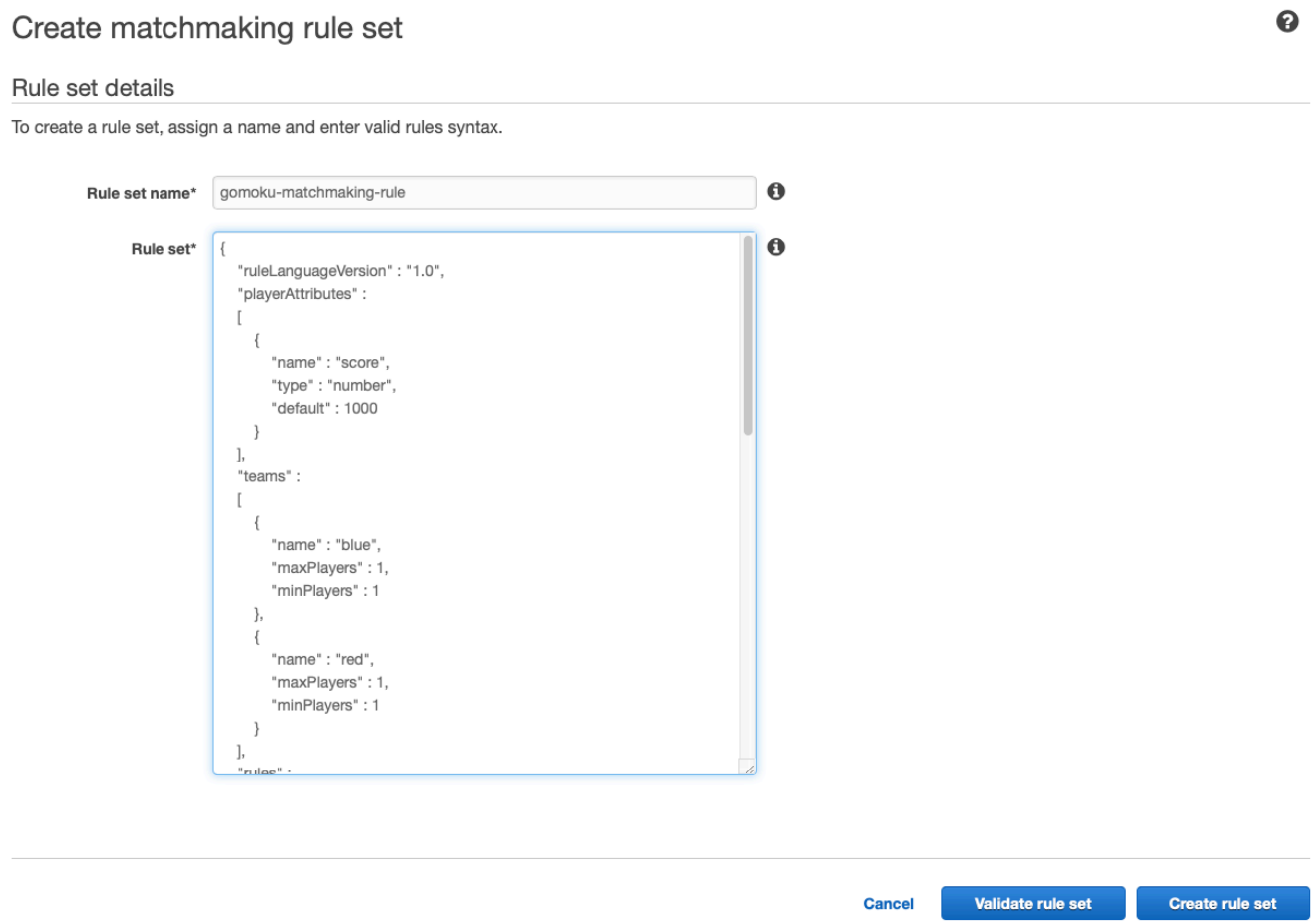
이제 FlexMatch 를 구현하기 위한 GameLift, Lambda, API Gateway 를 구성합니다.

FlexMatch 를 사용하기 위해서는 GameLift 의 Queue 서비스를 사용해야 합니다.

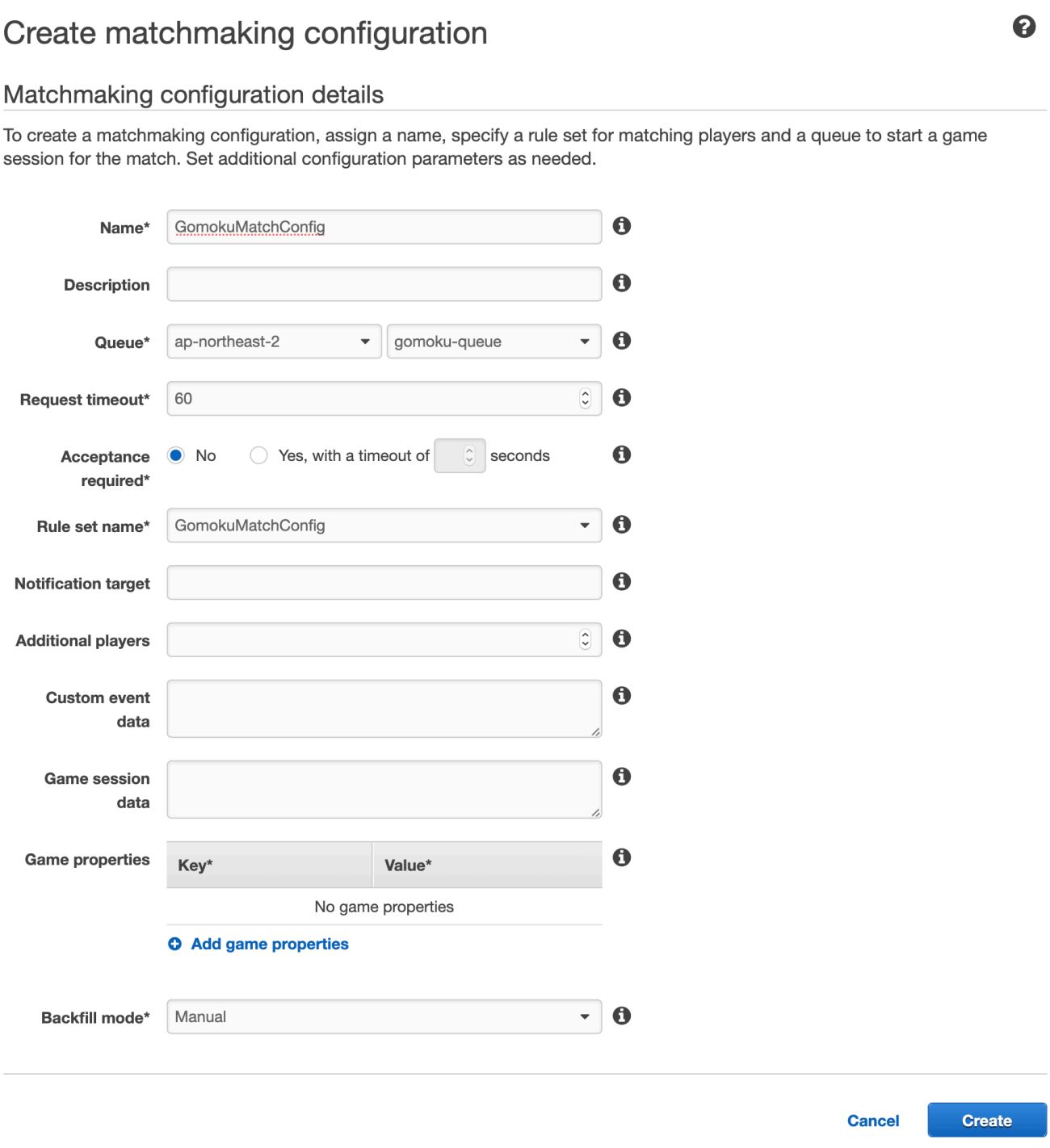
1. 메뉴에서 **Create a queue** 를 선택합니다.
2. 아래와 같이 Queue 의 이름을 입력하고, Add destination 에서 앞서 생성한 Alias 를 선택합니다. **Create queue** 를 선택하여 queue 를 생성합니다.



3. 다음 단계를 FlexMatch 를 위한 rule 을 만듭니다. **Create matchmaking rule set** 을 선택합니다.
4. 아래와 같이 **Rule set** 의 이름을 입력하고 Rule set 을 넣어 줍니다. Rule set 은 앞서 받은 파일에서 **GomokuRuleSet.json** 의 파일 내용을 복제하여 넣어 줍니다. 여기서의 rule 은 간단하게 score 점수가 300 점 이내의 사용자간에 Match 를 시켜주며 해당 사용자 match 가 시간내에 안 될 경우, rule 을 완화하여 match 시켜 줍니다. Validate rule set 을 선택하여 rule set 이 정상인지 확인 한 후에 create rule set 으로 rule set 을 생성합니다.



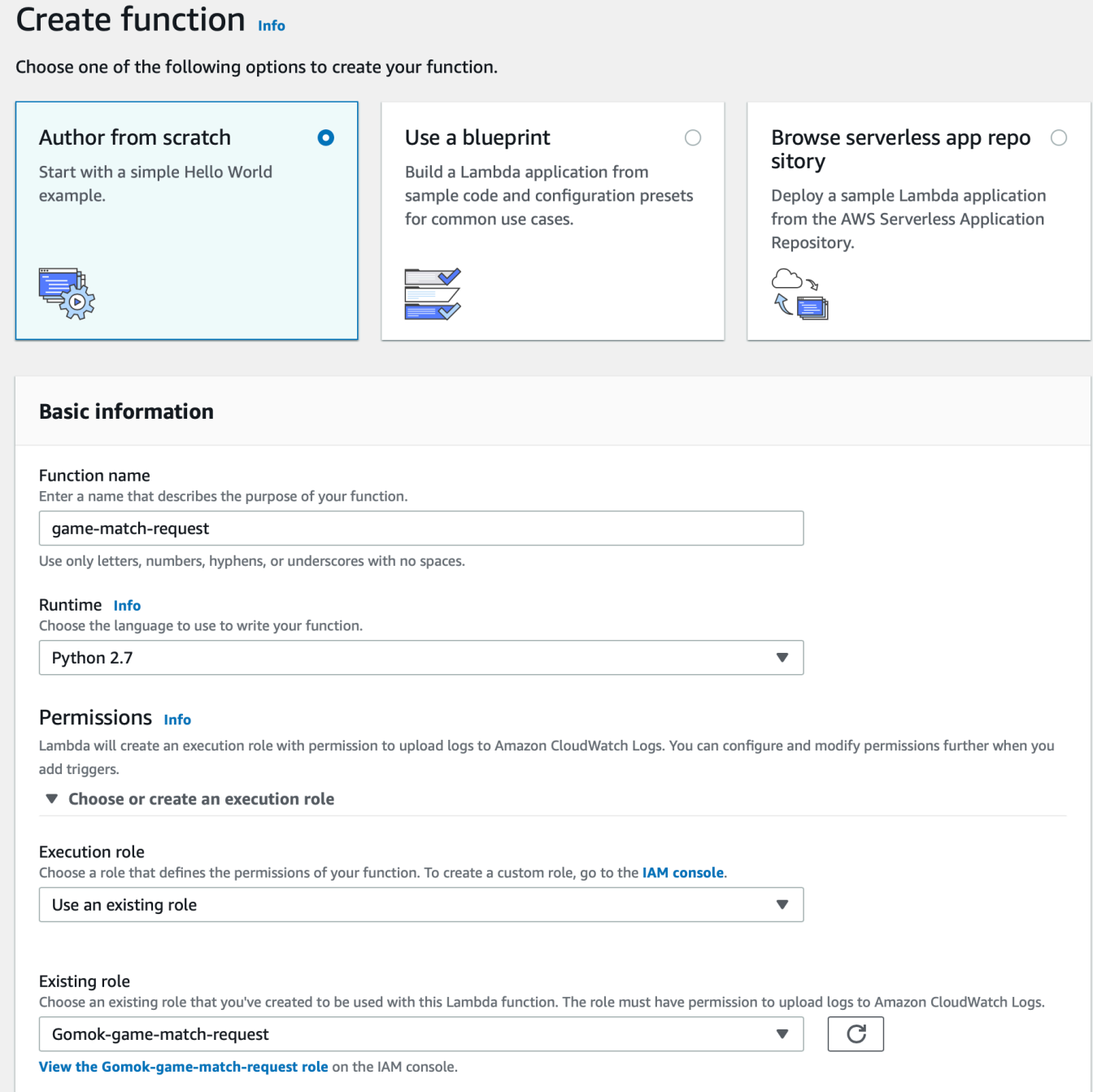
5. 다음은 앞서 만든 Queue 와 Matchmaking rule set 을 연결 시켜 주는 작업을 수행합니다. **Create matchmaking configuration** 을 메뉴에서 선택합니다. 아래와 같이 앞서 만든 queue 와 rule set 을 선택하여 configuration 을 생성합니다.



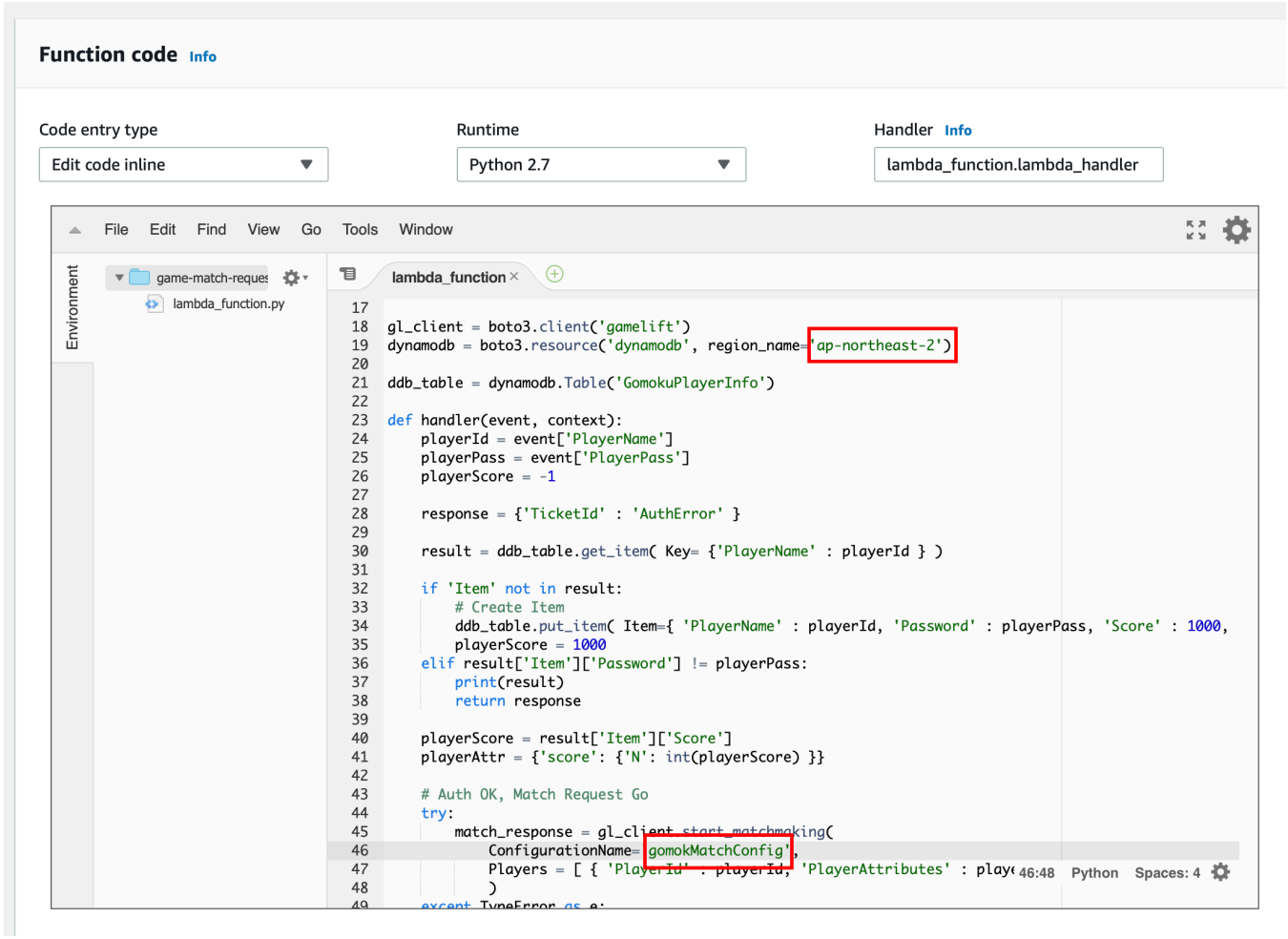
이제 GameLift 의 FlexMatch 설정은 완료하였습니다. 다음은 game client 가 FlexMatch 요청을 하는 Lambda 와 API Gateway 를 구성합니다.

이번 랩에서는 총 2개의 Lambda 함수를 생성할 것입니다. 하나는 client로부터 MatchMaking 요청을 처리하는 람다함수이고, 다른 하나는 MatchMaking 결과를 확인하기 위한 요청을 처리하는 람다입니다. 첫번째로 Matchmaking 요청을 처리하는 람다를 생성합니다.

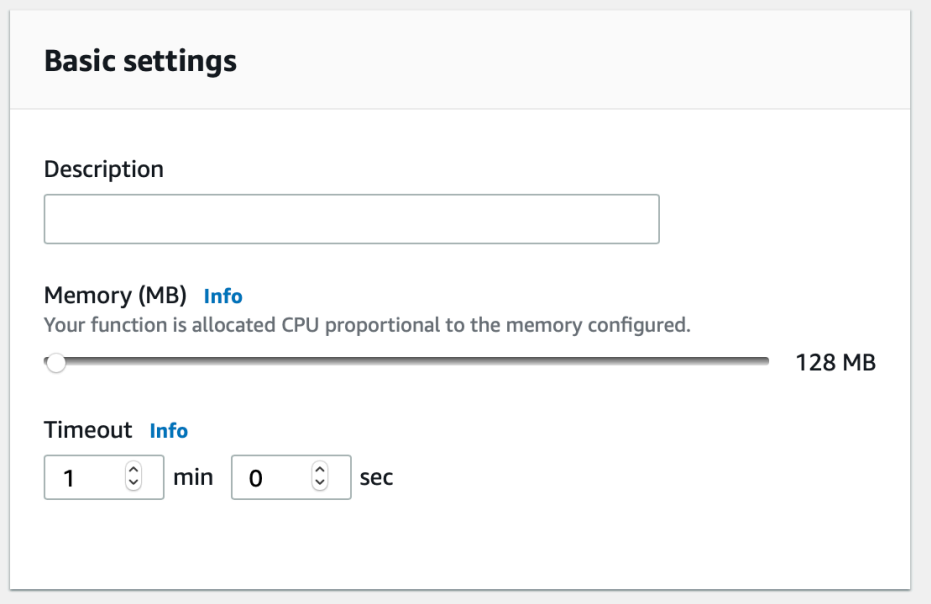
1. 콘솔에서 **Lambda** 메뉴로 이동합니다. https://console.aws.amazon.com/lambda
2. **Create function** 버튼을 클릭하여 첫번째 함수 생성을 시작합니다.
3. **Author from scratch** 메뉴를 선택하여 빈 함수를 우선 생성합니다.
4. **Name** 항목에는 **game-match-request**를 입력합니다.
5. **Runtime**은 Python 2.7을 선택합니다**.**
6. **Permissions** 항목에서 **Role** 은 Use an existing role을 선택하고 기존에 만들어둔 **Gomok-game-match-request**를 선택하고 Create function을 실행합니다.



7. 제공된 소스 파일 중 Lambda 폴더 밑에 있는 MatchRequest.py의 내용을 Lambda 코드창에 복제해 넣습니다. 코드 상의 Region과 Match Config의 이름이 앞서 생성한 이름과 동일한지 확인합니다. (대소문자 일치하도록 확인)



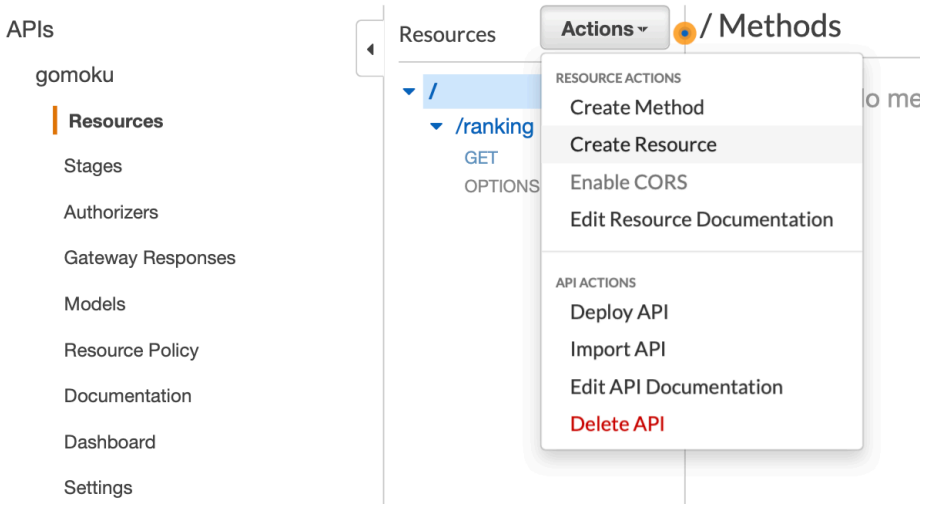
8. Lambda 함수의 실행 시간 및 메모리 설정을 아래와 같이 설정합니다.



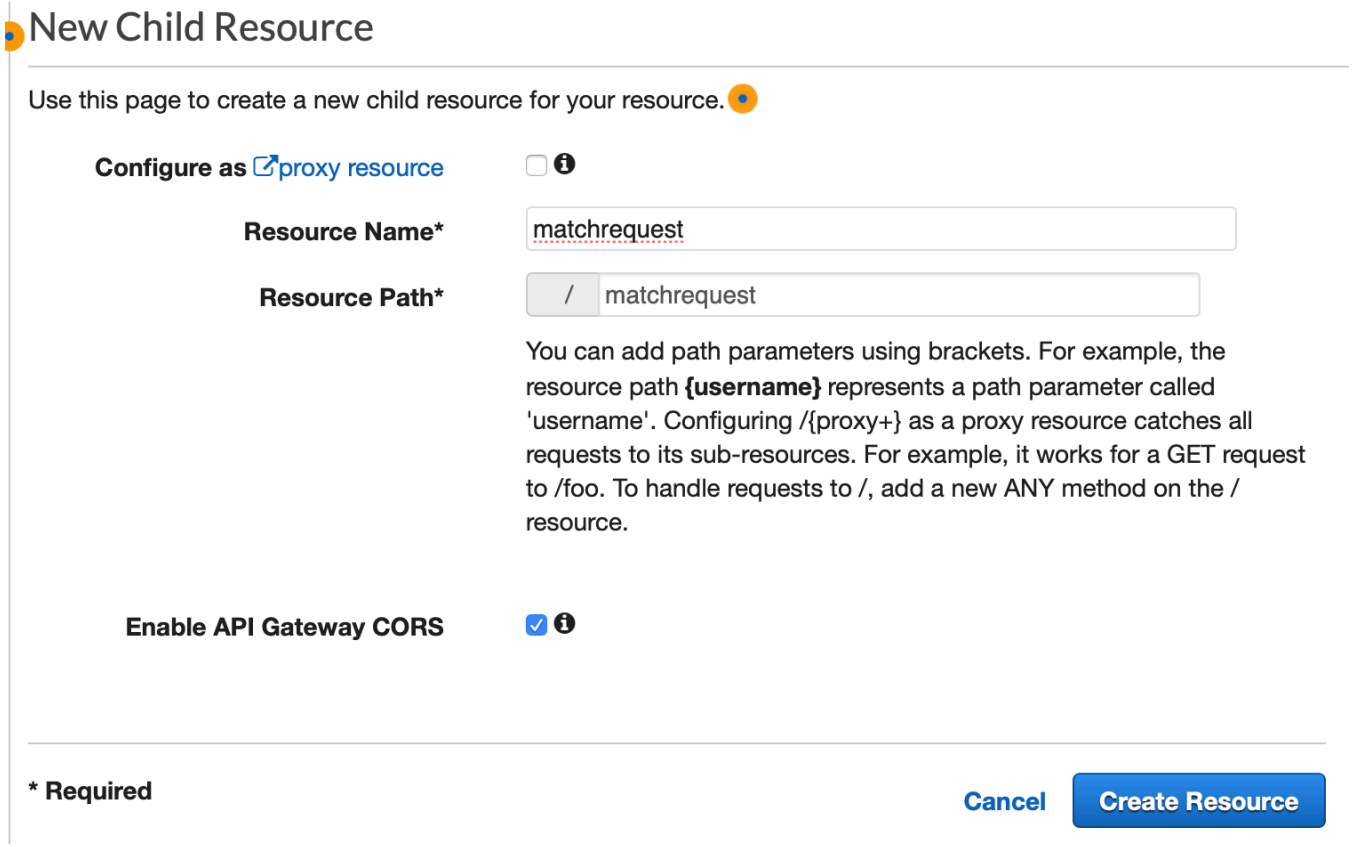
9. **Save** 버튼을 클릭하여 함수를 생성합니다. 생성한 함수는 game client 의 Match making 요청을 받아 DynamoDB 에서 사용자의 정보를 읽고 이에 기반하여 GameLift 에 match making 을 요청합니다.
10. **Create function** 버튼을 클릭하여 두번째 함수 생성을 시작합니다.
11. **Author from scratch** 메뉴를 선택하여 빈 함수를 우선 생성합니다.
12. **Name** 항목에는 **game-match-status**를 입력합니다.
13. **Runtime**은 Python 2.7을 선택합니다.
14. **Permissions** 항목에서 **Role** 은 Use an existing role을 선택하고 기존에 만들어둔 **Gomok-game-match-status**를 선택하고 Create function을 실행합니다.
15. 제공된 소스 파일 중 Lambda 폴더 밑에 있는 MatchStatus.py 의 내용을 Lambda 코드창에 복제해 넣습니다.
16. 앞서와 같이 Lambda 함수 실행시간을 1 분으로 조정합니다.
17. **Save** 버튼을 클릭하여 함수를 생성합니다. 생성한 함수는 game client 가 TicketId 로 자신이 요청한 match making 요청이 완료되었는지를 확인하고 요청이 완료되었다면 client 가 접근할 게임서버의 IP 주소와 Port 를 알려 줍니다.

다음은 앞서 생성한 Lambda 함수를 game client 가 호출할 수 있도록 API Gateway 를 설정합니다.

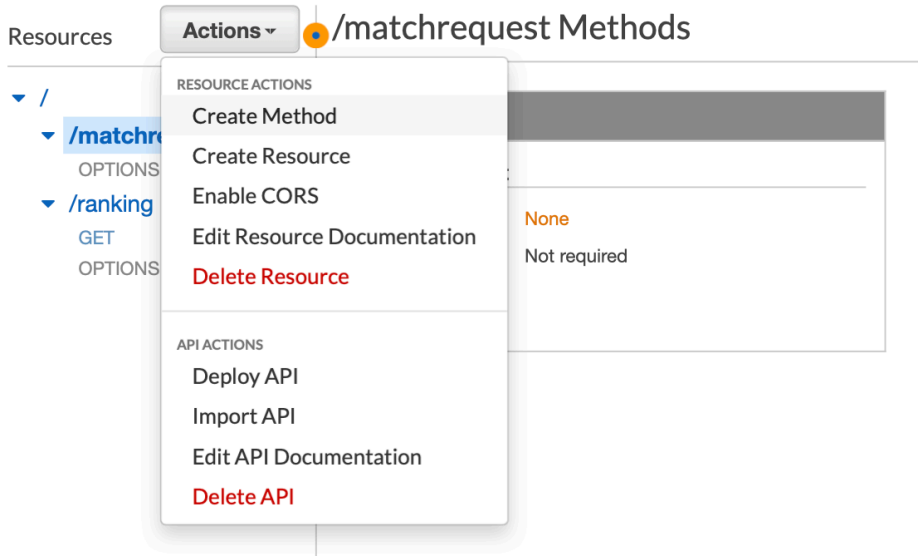
1. 우선 콘솔에서 API Gateway 메뉴로 이동합니다.
   https://console.aws.amazon.com/apigateway
2. 앞서 생성한 gomoku API를 선택하고 Create Resource를 선택합니다.



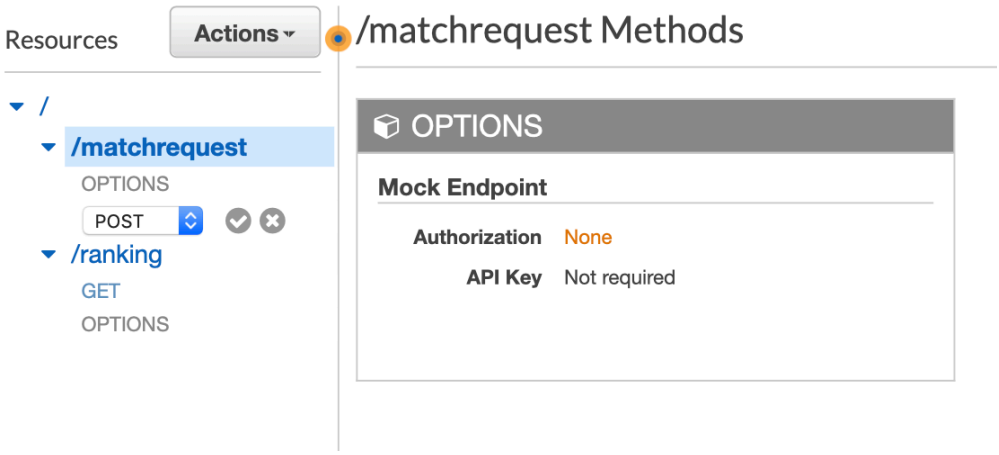
3. **Resource name**은 matchrequest를 입력하고 **Enable API Gateway CORS**를 체크 하고 **Create Resource** 버튼을 클릭하여 Resource를 생성합니다.



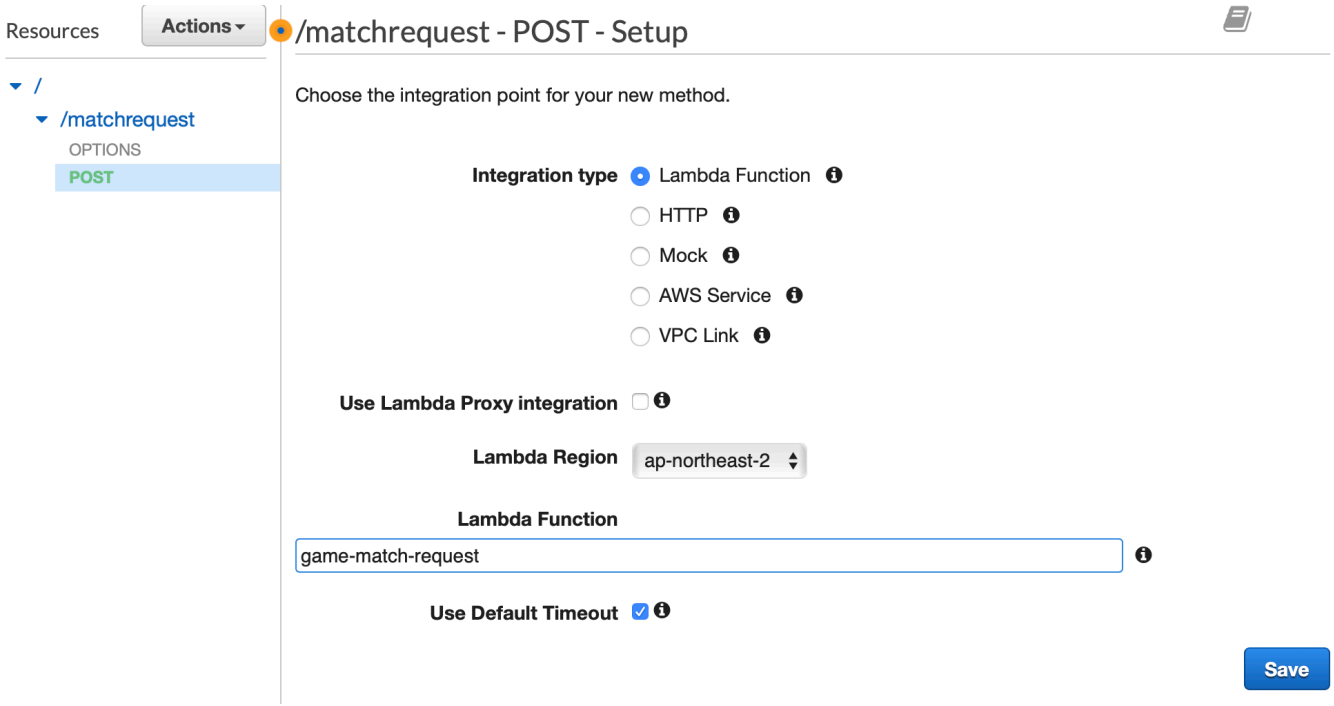
4. 생성한 Resource에 새로운 Method를 생성합니다. **Actions** 버튼을 클릭한 뒤 **Create Method** 메뉴를 선택합니다.



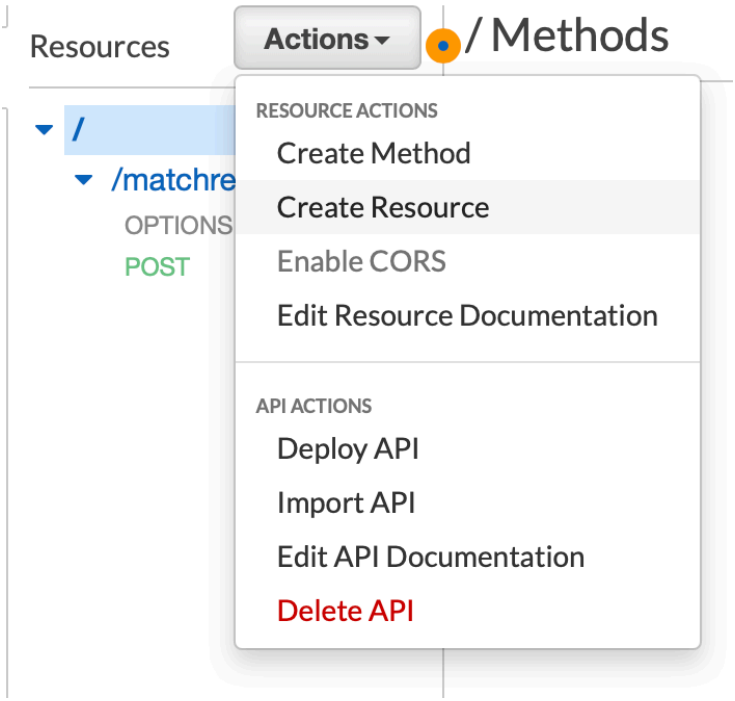
5. 아래 작은 리스트박스가 보일 것입니다. **POST**를 선택한 뒤 옆의 체크 버튼을 클릭합니다.



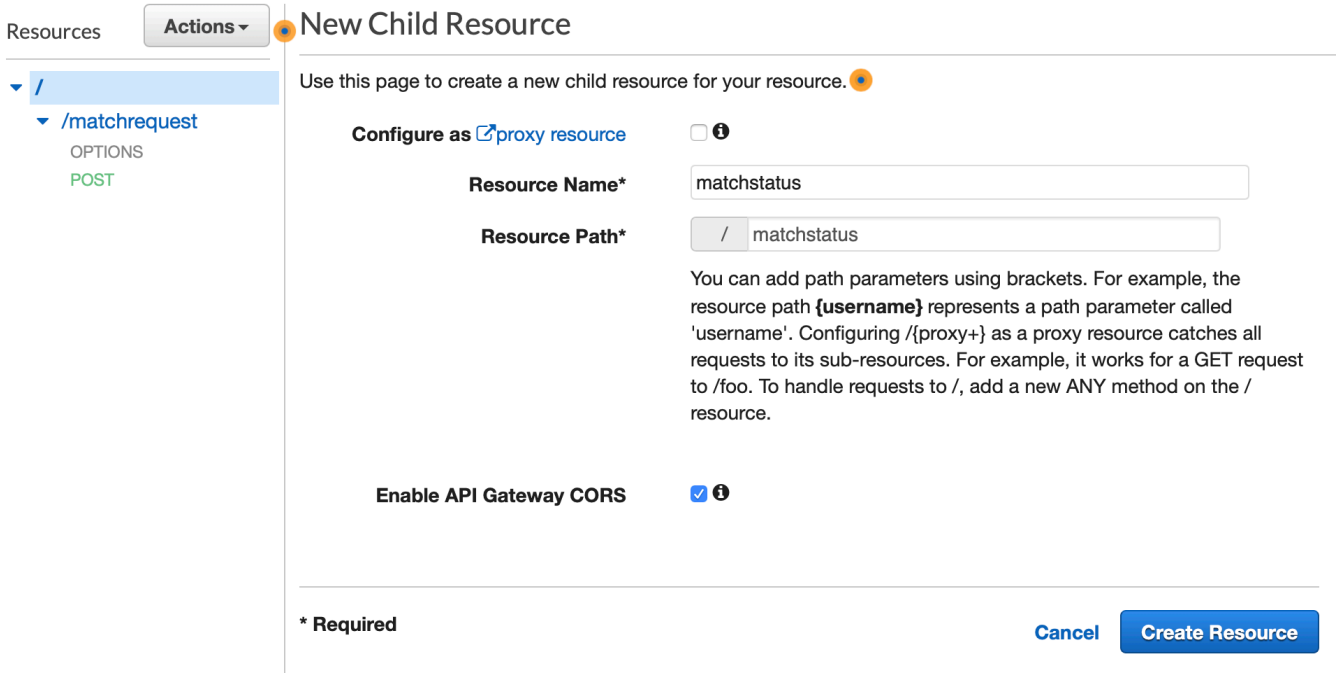
6. POST 메소드의 상세 설정에서 Integration type은 **Lambda Function**을 선택하고 Lambda Region에 실습을 진행 중인 Region을 선택합니다. Lambda Function에는 앞서 생성한 **game-match-request**를 선택한 뒤 **Save** 버튼을 클릭합니다.



7. 두 번째 람다함수를 연동하기 위해 생성한 API 에 두 번째 Resource 를 만듭니다. **API 의 root 를 선택하고** Actions 에서 Create Resource 를 선택합니다.

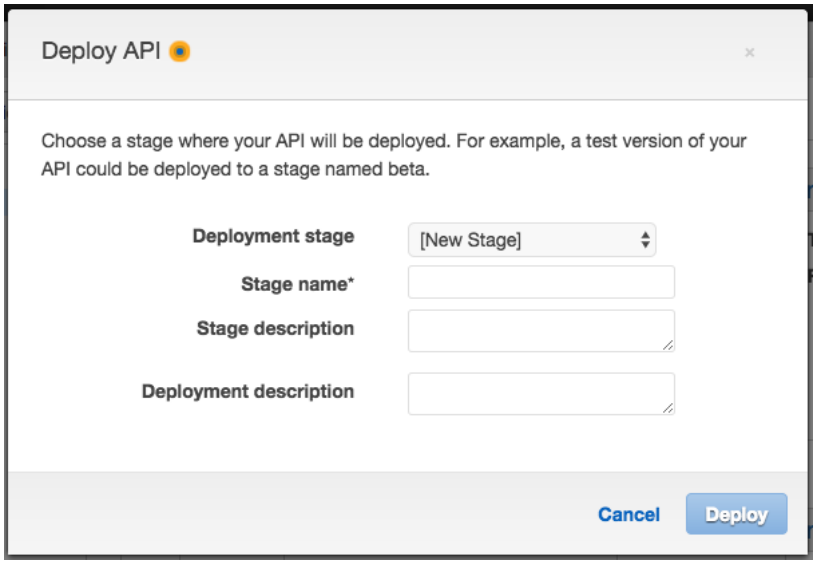


8. 앞서 첫번째 Resource 와 동일한 방식으로 생성합니다 **Resource Name** 은 matchstatus 로 입력합니다. **Enable API Gateway CORS** 를 체크합니다.

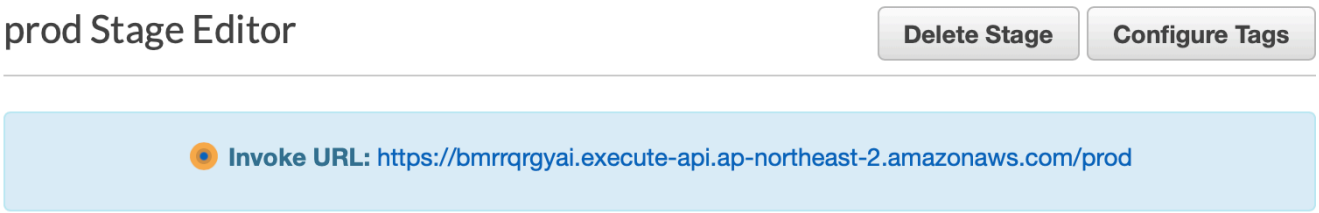


9. 앞서 첫번째 Resource 에 생성한 Method 와 동일한 방식으로 **Create Method** 를 수행합니다.

10. **POST** 메소드를 생성하고 **game-match-status** 람다를 연동하고 Save 버튼을 클릭하여 저장합니다.
11. API 구성이 되었습니다. 이제 배포를 수행합니다. **Actions** 버튼을 클릭하고 **Deploy API** 메뉴를 클릭합니다.



12. **[New Stage]** 를 선택하고 Stage name에는 **prod**를 입력합니다. **Deploy** 버튼을 클릭하여 진행합니다.
13. 완료되면 API의 Invoke URL이 표시됩니다. 아래 URL은 game client 설정에 사용됩니다.

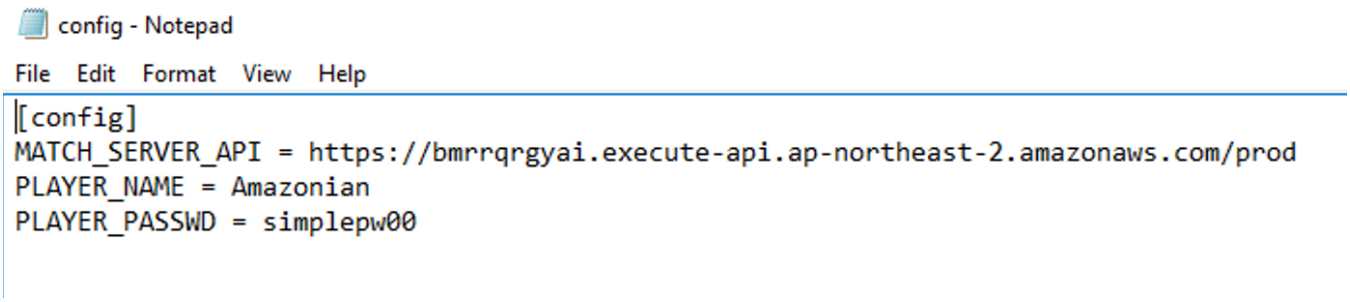


Section 6: Let's play the client

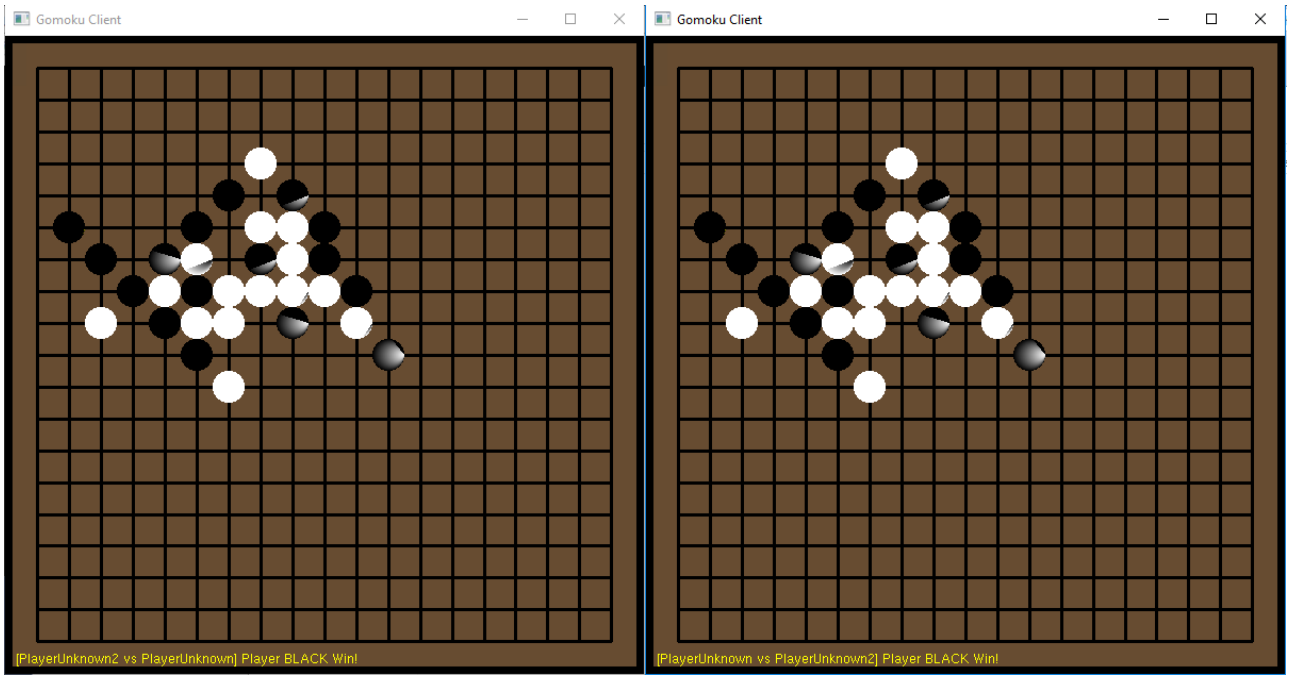
게임 클라이언트는 기본적으로 Windows OS에서 동작하도록 개발되었습니다. (Mac에서 실행할 수 있는 버전은 따로 제공됩니다. Mac 버전에서 사용을 위해서는 Xquartz를 미리 설치해야 합니다.)

게임 클라이언트는 Outbound TCP 커넥션을 2개 맺습니다. (방화벽 설정이 필요하다면 설정해줍니다.)

1. 다운로드 받은 실습 파일에서 Binaries/client_player1 폴더로 이동합니다.
2. 해당 client_player1폴더 내의 config.ini 파일의 수정이 필요합니다.
3. 앞선 Section 5의 마지막 단계에서 확인한 API Gateway URL을 MATCH_SERVER_API값에 입력합니다.



4. PLAYER_NAME과 PLAYER_PASSWD를 (임의로) 지정합니다.
5. 게임 클라이언트(GomokuClient.exe)를 실행합니다. 아래의 스크립 캡처와 같이 실행될 것입니다.
6. 다른 플레이어를 실행해야 합니다. 다운로드 받은 실습 파일에서 Binaries/client_player2 폴더로 이동하여 config.ini 파일을 앞서와 같이 수정하고 player2를 실행합니다.
7. 두 개의 게임 화면에서 우클릭한 뒤 Start 하세요!
8. 이미 졌다고 판단된다면 우클릭한 뒤 Give up을 할 수 있습니다.
9. DynamoDB의 Score를 임의로 변경하여 Matchmaking을 테스트해 보십시요.
10. 수고하셨습니다!



참고: MatchMaker는 기본적으로 PLAYER_NAME가 이미 DynamoDB에 존재하면 PLAYER_PASSWD를 비교합니다. 패스워드가 맞다면 로그인이 자동으로 완료 됩니다. 만일 PLAYER_NAME에 해당하는 데이터가 없다면 해당 플레이어를 새로 생성하고 자동으로 로그인까지 하도록 구현되어 있습니다. 자세한 구현은 따로 제공된는 MatchMaker 소스 코드를 참고하세요.

게임 클라이언트는 친구, 가족에게 배포하여 Gomoku(Omok, FiveStones)를 함께 즐기세요. (**바이너리를 재배포 할 때는 법적인 제한 사항을 확인해야 합니다)

Appendix A Deployment Package with Python cheat sheet

How to create a deployment package for Lambda using python
For detailed information, refer to our web page at http://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html

For a quick cheat sheet, as python environment in lambda does not have all the library user can possibly use, it is recommended that you use a customized deployment package for any function that extends beyond built-in AWS SDK.

In our case, we created the project directory, and using pip, installed the necessary library into the project directory, and zipped the directory as is.

From CLI, the sequence of commands should look like this.

```
] mkdir project
] cd project
] vi Scoring.py
] pip install redis –t .
] pip install boto3 –t .
] zip –r LambdaDeploy.zip *
```

For more information on creating a proper deployment package, please refer to the link above.

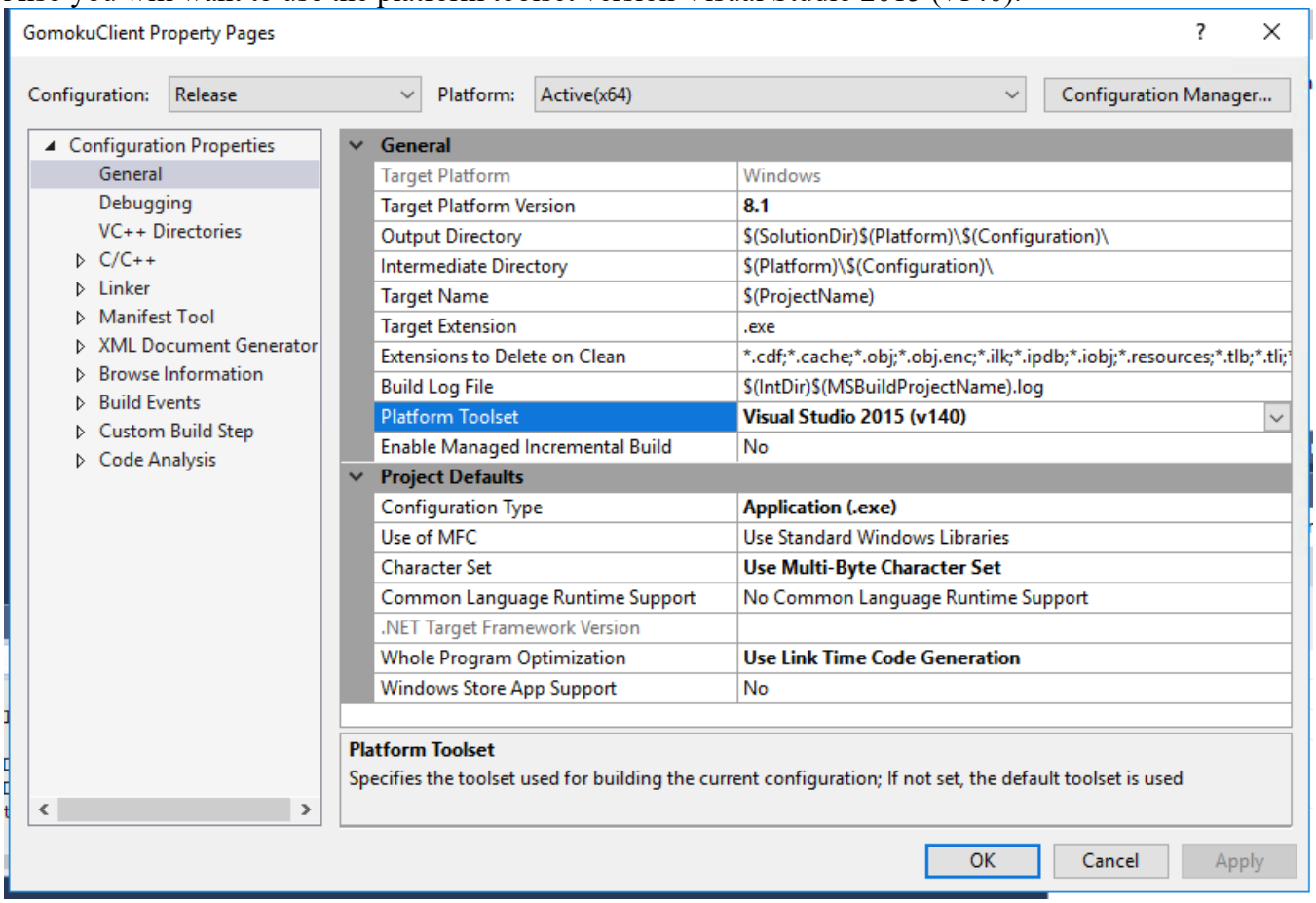
Appendix B Notes on compiling the source binary

Compiling the binaries used in this example.

This lab has 3 different binaries used throughout. 1 Game client, and 2 game server binaries are used. For your convenience, each of them are precompiled and supplied within the starting package, however if you want to compile the binary yourself, here's the simple instruction on compiling.

You need to have Visual Studio 2015 Community Edition installed. If you can reconfigure VS 2017 to match that of 2015, you might try, however due to the NuGet package dependency, it is recommended to use VS 2015 edition instead.

Also you will want to use the platform toolset version Visual Studio 2015 (v140).



The each of the project files, GomokuServer.sln, GomokuMatchMaker.sln, GomokuClient.sln are prepared within the project directory.

As for NuGet, please refer to this link for detailed information on how to update the package maneger on Visual Studio 2015.

Appendix C AWS Cli environment notes

Creating a AWS Cli environment.

In this lab, we need to create a cli environment for GameLift binary upload.

Fortunately, we can easily create a AWS CLI environment following below steps.

For installing on each OS, refer to this page:
http://docs.aws.amazon.com/cli/latest/userguide/installing.html

Once you have the environment installed, you will need to create a IAM user with programmatic access, and setup your command line interface with the created IAM credentials.

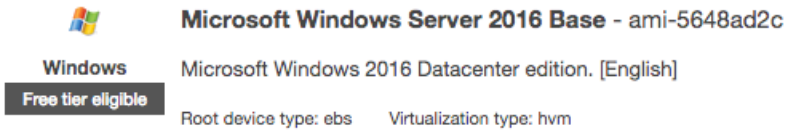
Please refer to this page for detailed instructions:

http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

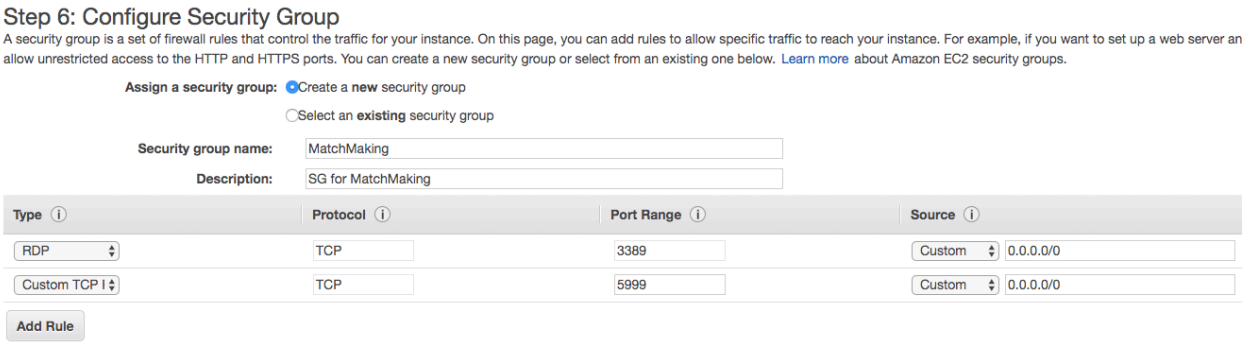
Appendix D Setting up Windows notes

EC2 Windows Server 생성 및 설정

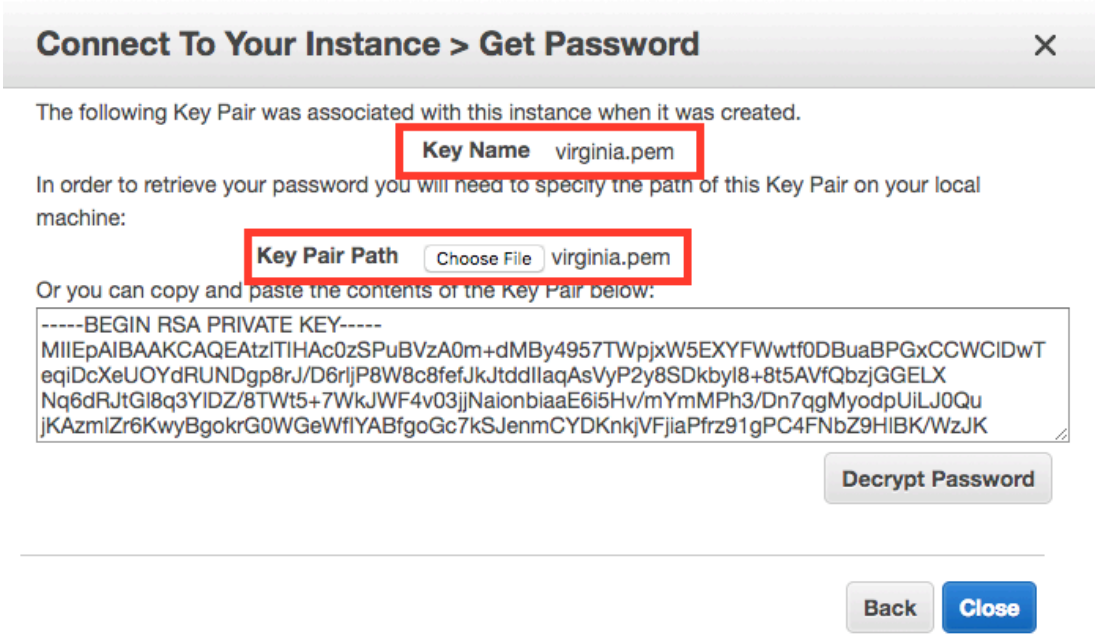
1. 콘솔에서 EC2 서비스 선택 후 Launch Instance 버튼을 클릭하여 EC2 생성을 시작합니다.
2. Amazon Machine Image 는 Microsoft Windows Server 2016 Base 를 선택합니다.



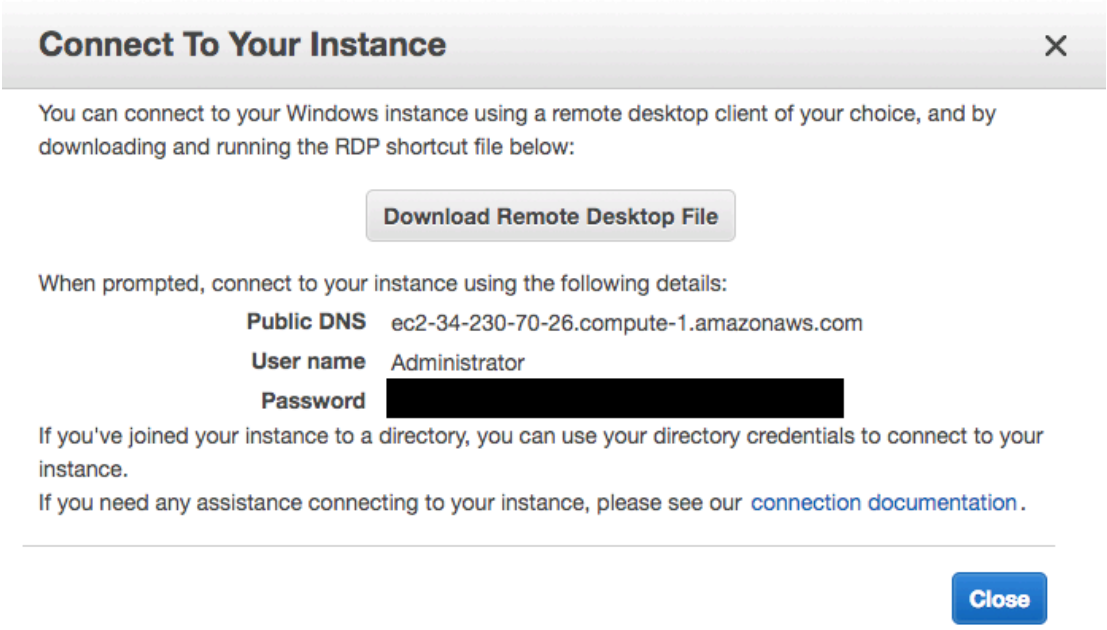
3. Instance Type 은 t2.large 혹은 더 큰 타입을 선택합니다.
4. 보안 그룹은 다음과 같이 설정합니다.



5. 갖고 있는 Key Pair(.pem 파일이 있다면)가 있다면 그대로 사용합니다. Key Pair 가 없는 경우 Create a new key pair 옵션을 선택하여 생성 후 다운로드합니다. 생성한 Key Pair 는 인스턴스의 패스워드 복호화에 사용되기 때문에 패스워드가 지정되지 않은 Key Pair 를 사용해야합니다.
6. Windows Server 의 경우 접속 가능 상태까지 약 4 분이 소요됩니다.
7. EC2 서비스의 Instances 메뉴로 이동하여 생성한 인스턴스를 선택한 뒤 상단의 Connect 버튼을 클릭합니다.
8. 기본 계정은 Administrator 가 할당되며 Get Password 버튼을 클릭하여 접속 패스워드를 얻습니다.
9. 패스워드는 Public Key 를 통해 생성되며 암호화 되어 있습니다.
10. 다음과 같이 인스턴스 생성할 때 사용한 Key Pair 와 동일한 Key 를 선택한 뒤 Decrypt Password 버튼을 클릭하여 패스워드를 복호화합니다.



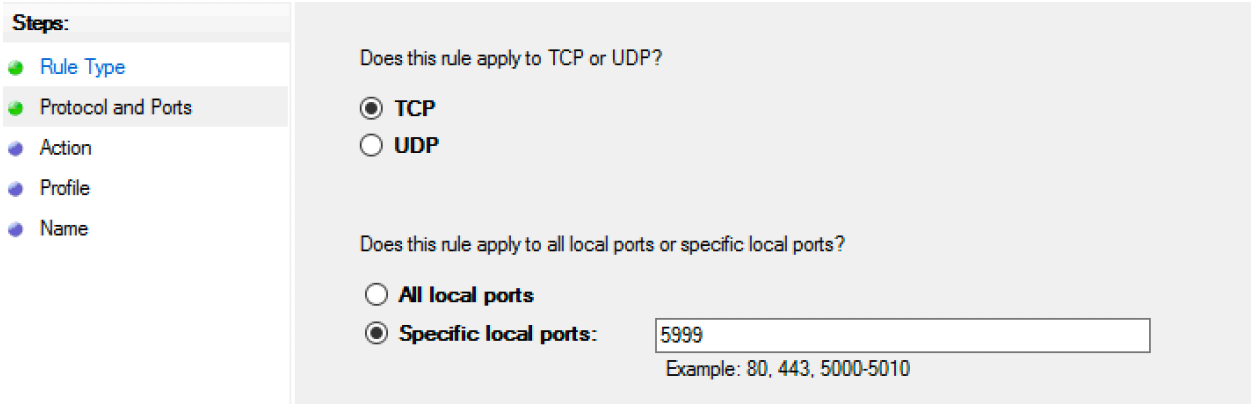
11. 다음과 같이 인스턴스 접속 정보가 화면에 나타납니다.



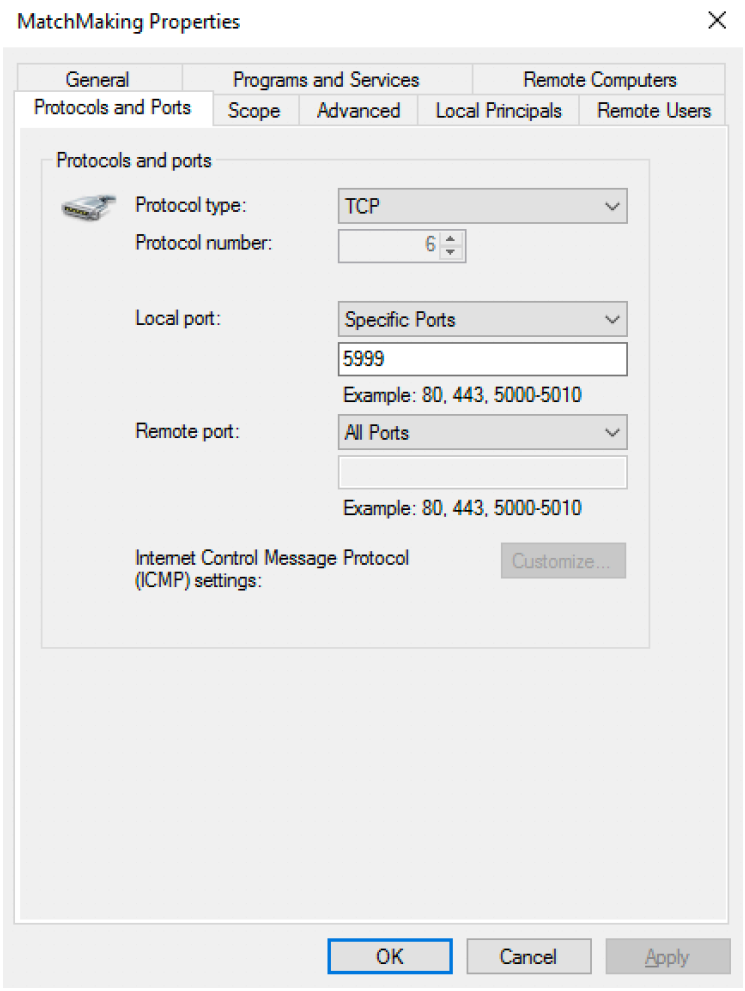
12. 화면의 정보를 이용하여 인스턴스에 원격접속합니다. (Public IP 를 할당하지 않으셨다면 Public DNS 가 제공되지 않습니다. 이 경우 Elastic IP 를 생성하여 인스턴스에 할당한 뒤 EIP 를 통해 접속합니다.)
13. Windows 사용자는 원격데스크톱 (실행 -> mstsc 입력)을 사용하고 Mac 사용자는 App Store 에서 Microsoft Remote Desktop 을 설치하여 접속합니다.

14. 방화벽 설정을 해줍니다. Control Panel -> System and Security -> Windows Firewall 메뉴로 이동합니다.

15. 왼쪽 메뉴의 Advanced settings 로 이동합니다.
16. 왼쪽 메뉴의 Inbound Rules 를 선택하고 오른쪽의 New Rule... 메뉴를 클릭합니다. New Inbound Rule Wizard 가 시작됩니다.
17. Rule Type 은 Port 를 선택하고 Next 를 클릭합니다. Specific local ports 에 5999 를 입력한 뒤 Next 를 클릭합니다.



18. 이후는 기본값으로 진행합니다. 마지막으로 정책 이름을 입력하신 뒤 Finish 를 클릭하여 방화벽 설정을 완료합니다.



19. 이전 실습으로 돌아가서 Matchmaker 서버 구성을 완료합니다.