



Gaming on AWS

Serverless 로 게임 서비스 구현하기

2017 Oct 24th

Contents

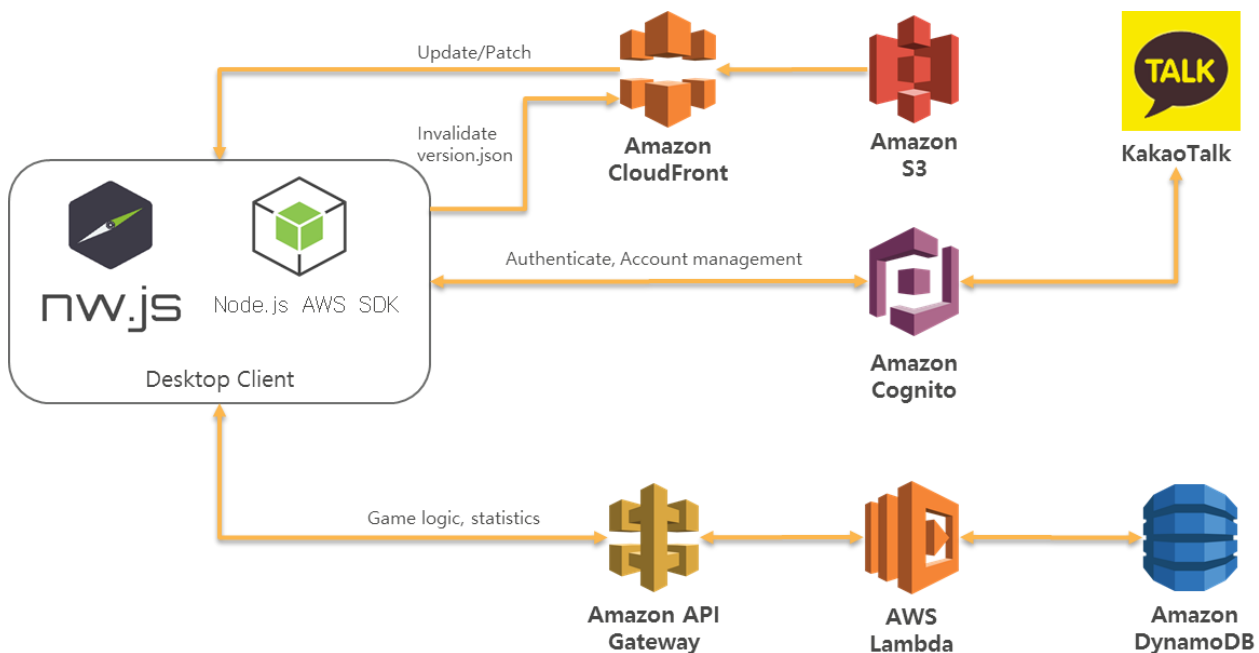
Serverless 로 게임 서비스 구현하기.....	3
필수 준비 사항	4
AWS 계정	4
카카오 디벨로퍼 계정	4
텍스트 에디터.....	4
NWJS.....	4
실습 모듈	5
리전 선택.....	5
Lab 1. Cognito 를 이용하여 카카오톡 연동하기	6
S3 및 Cloudfront 생성하기	7
S3 Bucket 생성하기	7
Cloudfront Distribution 생성하기.....	10
카카오 개발자 등록 및 애플리케이션 설정하기.....	11
Cognito 생성하기	15
Lambda 및 API Gateway 생성하기.....	18
Lambda 함수 생성하기.....	18
API Gateway 생성하기.....	24
카카오 인증 테스트하기	30
웹에서 테스트.....	30
NWJS 로 테스트.....	31
Lab2. S3 와 Cloudfront 로 클라이언트 배포 서비스 제작하기	33
시나리오	33
IAM Role 에 권한 추가하기	33
S3 버킷에 클라이언트 업데이트용 path 추가하기.....	36
테스트하기	36
Lab3. DynamoDB 와 stream 을 이용한 플레이데이터 저장 및 통계	40
IAM role.....	40
DynamoDB	41
Lambda 함수 설정하기	43
API Gateway.....	47
애플리케이션 설정 및 테스트.....	51
리소스 정리하기	52

Serverless 로 게임 서비스 구현하기

본 워크샵에서는 AWS 의 다양한 Serverless 서비스를 통해 클라이언트 배포(업데이트/패치), 인증, 게임 사용자 데이터의 저장 및 통계를 확인하는 등의 게임 서비스에 필요한 애플리케이션을 만들고 배포해 봅니다.

이 실습은 AWS Lambda, Amazon API Gateway, Amazon S3, Amazon DynamoDB, Amazon Cognito, Amazon Cloudfront 를 활용합니다. Amazon S3 는 정적 웹호스팅을 통해 KakaoTalk 과 Cognito 를 연계하는 proxy 역할 및 클라이언트 업데이트를 위한 origin 서버 기능을 제공합니다. NW.js 에서 실행되는 node.js javascript 는 AWS SDK 를 사용하여 AWS 서버리스 서비스들과 데이터를 주고 받으며 Lambda 및 API Gateway 와 연계하여 Application logic 을 구현합니다. Amazon Cognito 는 KakaoTalk 이 지원하는 OAuth 표준을 사용하여 사용자의 인증과 관리 기능, CognitoSync 를 통한 사용자 데이터 저장 기능을 제공합니다. Cloudfront 는 비용 절감 및 클라이언트 다운로드 성능 최적화를 위한 CDN 서비스를 제공합니다. 마지막으로 DynamoDB 및 DynamoDB Stream 을 활용하여 사용자 플레이 점수를 저장하고 통계 기능을 제공합니다.

전체 아키텍처 그림은 아래 다이어그램을 참조하십시오.



필수 준비 사항

AWS 계정

본 워크샵을 진행하려면 AWS 계정을 준비해야 합니다. AWS IAM, S3, DynamoDB, Lambda, API Gateway, Cloudfront 및 Cognito 에 접근할 수 있어야 하며, 본 가이드는 한명이 하나의 AWS 계정을 사용한다고 가정합니다. 다른 사람과 계정을 공유할 경우 특정 리소스에 대해 충돌이 발생하므로 권장하지 않습니다.

본 워크샵의 일환으로 사용하는 모든 리소스는 AWS 계정이 생성된지 12 개월 미만일 경우 제공되는 AWS 프리티어로 사용 가능합니다. 프리티어 사용량을 넘어서거나 프리티어가 만료될 경우 과금이 될 수 있습니다. 따라서 새로운 실습용 계정을 만드는 것을 권장합니다. 자세한 내용은 [AWS 프리티어 페이지](#)를 참조하십시오.

카카오 디벨로퍼 계정

카카오톡 인증 연동을 위하여 KakaoTalk Developer 계정이 필요합니다. 상세한 내용은 [Kakao Developers 페이지](#)를 참조하십시오.

텍스트 에디터

설정 파일의 업데이트를 하기 위하여 로컬 텍스트 편집기가 필요합니다.

NWJS

본 워크샵에서 사용하는 데스크탑 애플리케이션은 NWJS 로 구현되었습니다. 소스코드에 함께 포함된 NWJS 를 압축을 해제합니다. Windows 의 경우 nwjs-sdk-v0.25.0-win-x64.zip, Mac 의 경우 nwjs-sdk-v0.25.0-osx-x64.zip 파일입니다.

실습 모듈

이 워크샵은 세 가지 실습 모듈로 나뉩니다. 다음 단계로 진행하기 전에 각 모듈을 완료하여야 합니다. 워크샵을 마친 후에는 자원 삭제 가이드에 따라 생성된 모든 리소스를 삭제할 수 있습니다.

- Lab1. Cognito 를 이용하여 카카오톡 연동하기
- Lab2. S3 와 Cloudfront 로 클라이언트 배포 서비스 제작하기
- Lab3. DynamoDB 와 stream 을 이용한 플레이어데이터 저장 및 통계

리전 선택

이 실습은 다음 서비스를 지원하는 모든 AWS 리전에 배포할 수 있습니다.

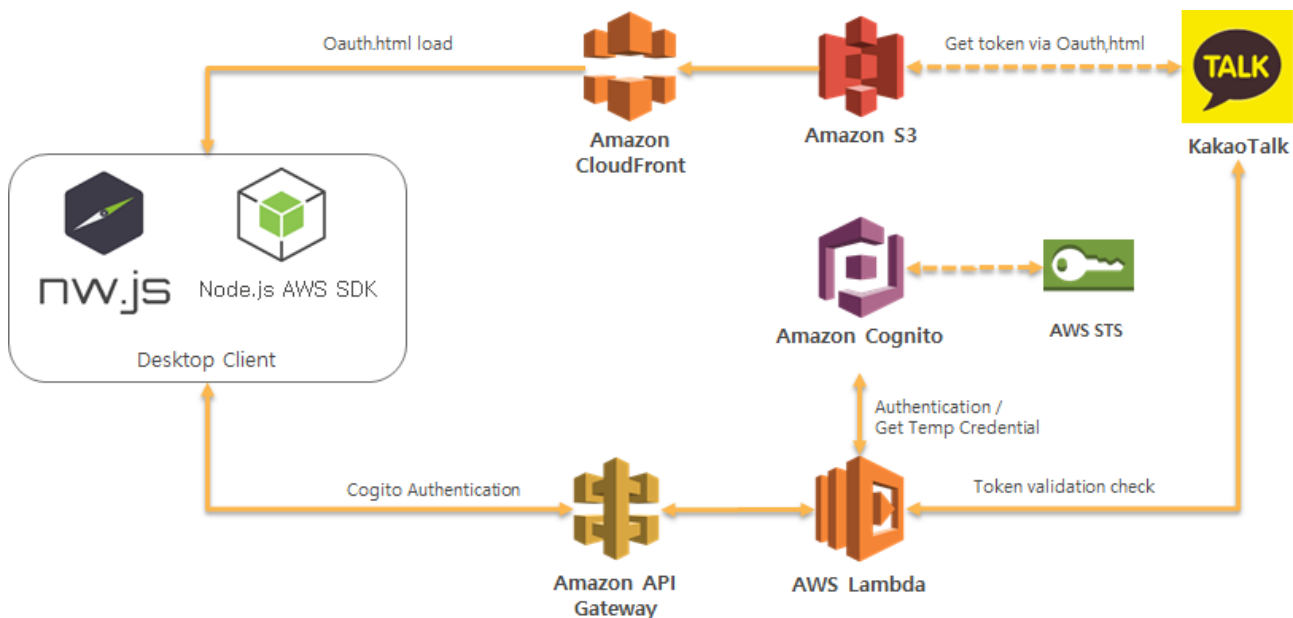
- Amazon Cognito
- AWS Lambda
- Amazon API Gateway
- Amazon S3
- Amazon DynamoDB
- Amazon Cloudfront

AWS 설명서에서 리전 표 를 참고하여 지원되는 서비스가 있는 지역을 확인할 수 있습니다. 지원되는 지역중에서는 N. Virginia, Ohio, Oregon, Ireland, Frankfurt, Tokyo, Sydney, Seoul 이 있습니다.

Lab 1. Cognito 를 이용하여 카카오톡 연동하기

이 모듈에서는 KakaoTalk 과 Cognito 를 이용하여 인증 서비스를 구성합니다. KakaoTalk 에서 OpenID 를 지원하지 않기 때문에, OAuth 2.0 표준을 통하여 Custom Source 로 Cognito 와 연동할 수 있습니다. OpenID 나 Google, Twitter, Facebook, Amazon, Digits 의 경우에는 Cognito Console 에서 더욱 간단하게 인증 연계가 가능합니다. KakaoTalk javascript API 는 webserver 를 통한 loading 만을 허용하기 때문에 Desktop application 에서 웹서버 없이 사용할 수 없고, RESTful API 역시 KakaoTalk API 의 CORS 설정으로 인해 Desktop application 에서 구현할 수는 없습니다. 따라서, 이 실습에서는 Amazon Simple Storage Service(S3)의 정적 웹페이지 호스팅 기능을 이용하여 Kakaotalk 인증을 중개하도록 구성합니다. 만일 Mobile Application 이나 웹앱이라면 S3 정적 웹페이지 호스팅 과정 없이 Kakao API 와 AWS SDK 를 사용하여 연계가 가능합니다.

Kakao Login 을 사용하는 로그인 과정은 다음과 같은 흐름을 가지게 됩니다.



사용자는 oauth.html 의 javascript 를 통하여 Kakao Login 을 요청하여 Key 를 발급받고, 필요정보를 AWS API Gateway 로 전달합니다. API Gateway 는 들어온 정보를 가지고 Lambda 를 호출합니다. Lambda 함수는 Kakao Login 에서 전달받은 Key 정보의 정합성을 확인하고 올바른 키의 경우 Cognito 에 Cognito Token 을 요청합니다. Token 이 발급되면 API Gateway 를 통해 response 를 반환하게 됩니다. 이제 사용자는 Cognito ID 와 Token 을 가지고 해당 세션에 Cognito 를 통한 Credential 을 할당 받고 정의된 Role 의 권한이 허용하는 AWS 서비스를 활용할 수 있게 됩니다.

S3 및 Cloudfront 생성하기

이 실습에서 Kakaotalk 연동을 위한 oauth.html 과 클라이언트 리소스 업데이트를 위한 origin 으로 S3 서비스를 사용합니다. 이 단계에서는 S3 bucket 과 Cloudfront Distribution 을 생성합니다.

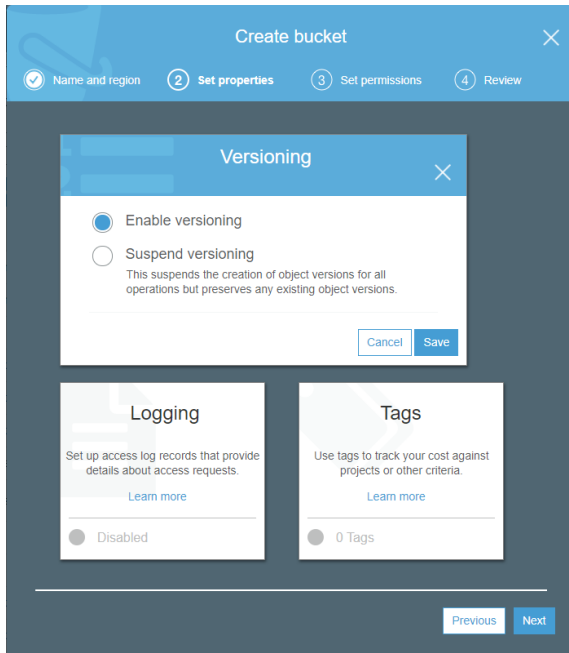
S3 Bucket 생성하기

1. AWS Management Console 에서 Services 를 선택한 다음 S3 를 선택하십시오.
2. **+Create Bucket** 을 선택하십시오.
3. gamingonaws-yourname 와 같은 전 세계적으로 고유한 이름을 설정하십시오.
4. 드롭다운 메뉴에서 이 실습에서 사용할 리전을 선택하십시오.

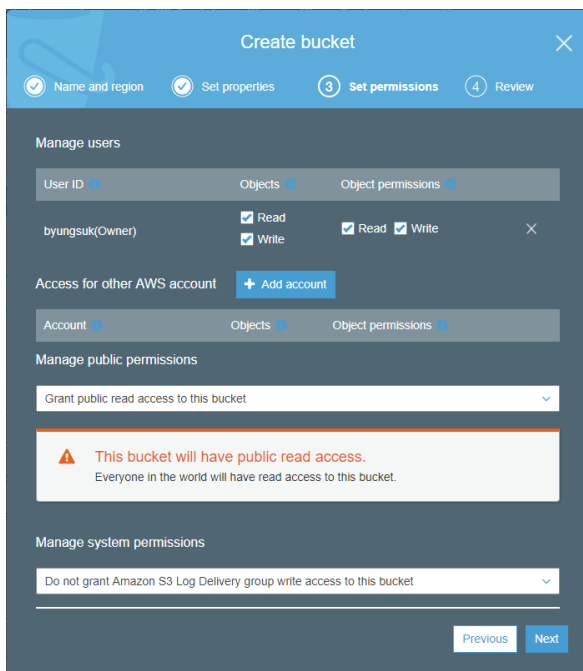
Gaming on AWS

Serverless 로 게임 서비스 구현하기

5. Next 를 누르고 Versioning 을 활성화합니다. (이후 Client update/patch 에서 S3 Versioning 기능을 활용합니다.)



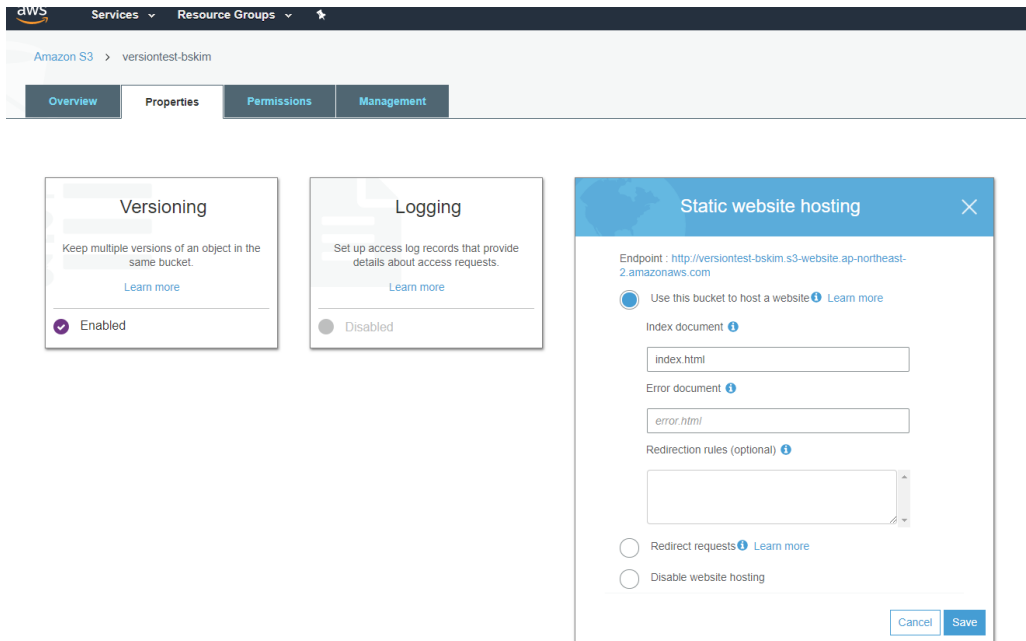
6. Next 를 누르고 Set Permissions 탭에서 "Grant public read access to this bucket"을 선택합니다. 이후 기본 값으로 Bucket 을 생성합니다.



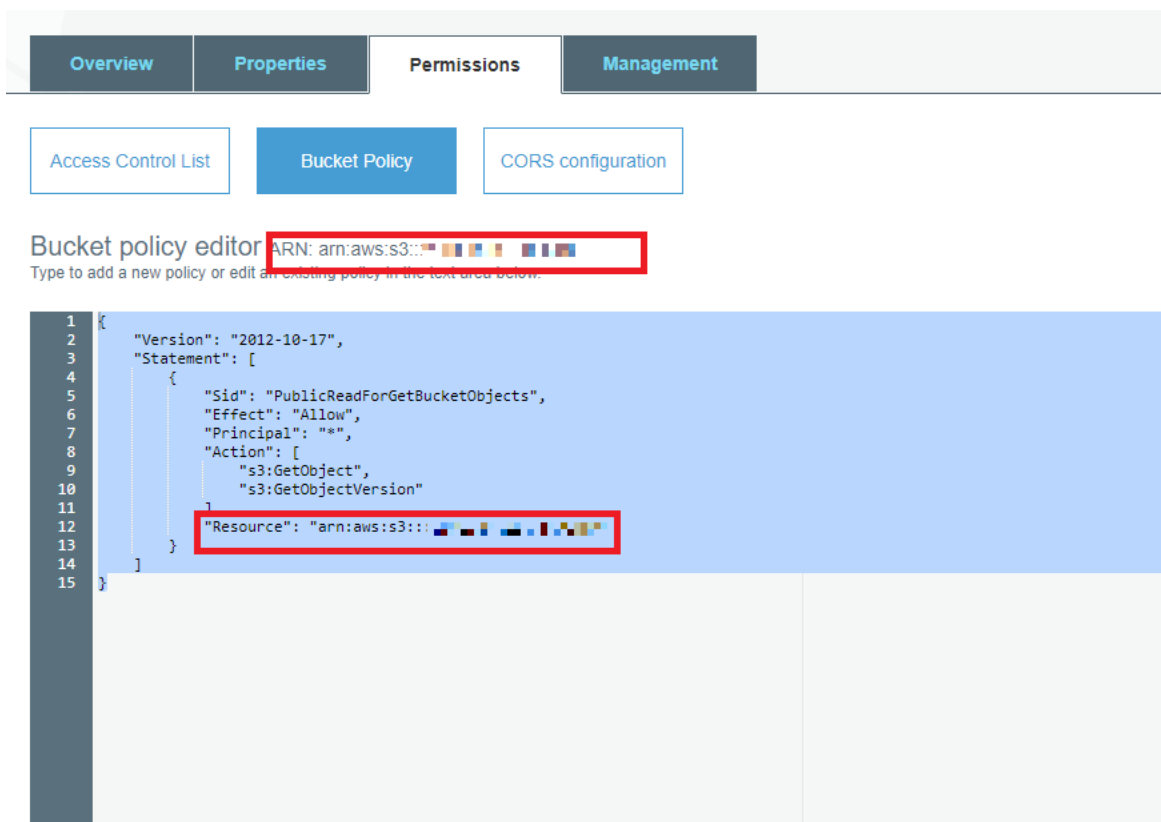
Gaming on AWS

Serverless 로 게임 서비스 구현하기

7. 생성된 버킷의 Properties Tab 에서 Static website hosting 을 활성화합니다. (이 때 index Document 에는 index.html 을 기입해둡니다.)



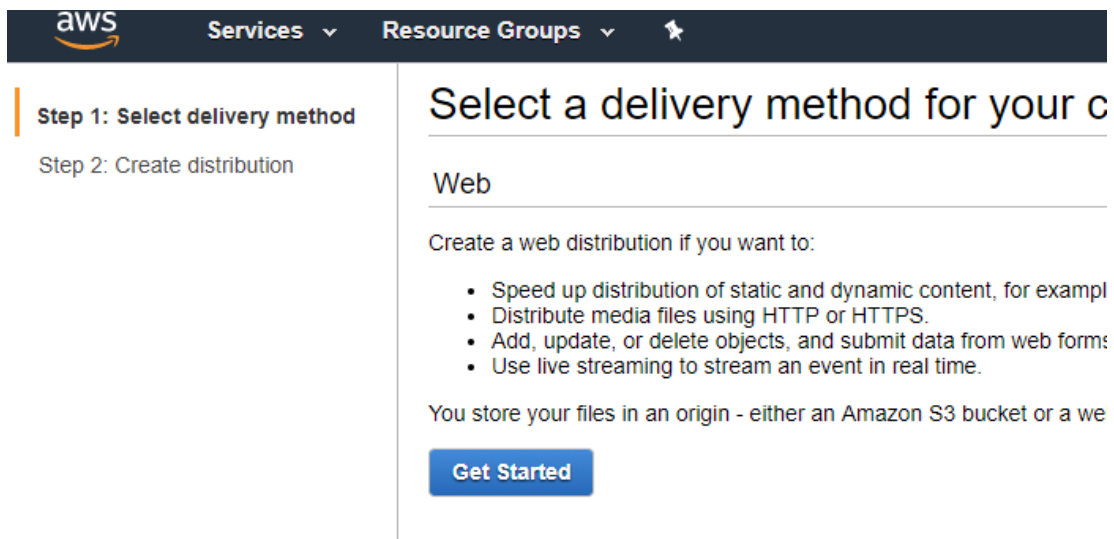
8. Permissions Tab 에서 Bucket 정책을 다음과 같이 추가합니다. <bucket_name>을 생성한 bucket 이름으로 대체합니다.



```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadForGetBucketObjects",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::<bucket_name>/*"
    }
  ]
}
```

Cloudfront Distribution 생성하기

1. AWS Management Console 에서 Services 를 선택한 다음 Cloudfront 를 선택하십시오.
2. Create Distribution 버튼을 클릭하여 Distribution 을 생성합니다.
3. Delivery method 로 Web 을 선택하십시오



4. Origin Domain Name 으로 위에서 생성한 Bucket 을 설정합니다.

Step 1: Select delivery method
Step 2: Create distribution

Create Distribution

Origin Settings

Origin Domain Name:

Origin Path:

Origin ID:

Restrict Bucket Access: ☐ Yes ☒ No

Origin Custom Headers: Header Name Value

Default Cache Behavior Settings

5. Query String Forwarding and Caching 을 Forward all, cache based on whitelist 로 선택하고, Query String Whitelist 에 **versionId** 라고 입력합니다. (이후 Client resource update/patch 에서 해당 기능을 사용합니다.)

[Learn More](#)

Minimum TTL:

Maximum TTL:

Default TTL:

Forward Cookies:

Query String Forwarding and Caching:

Query String Whitelist:

Valid characters: a-z, A-Z, 0-9, -, ., *, %

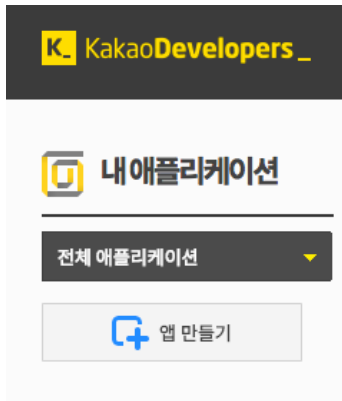
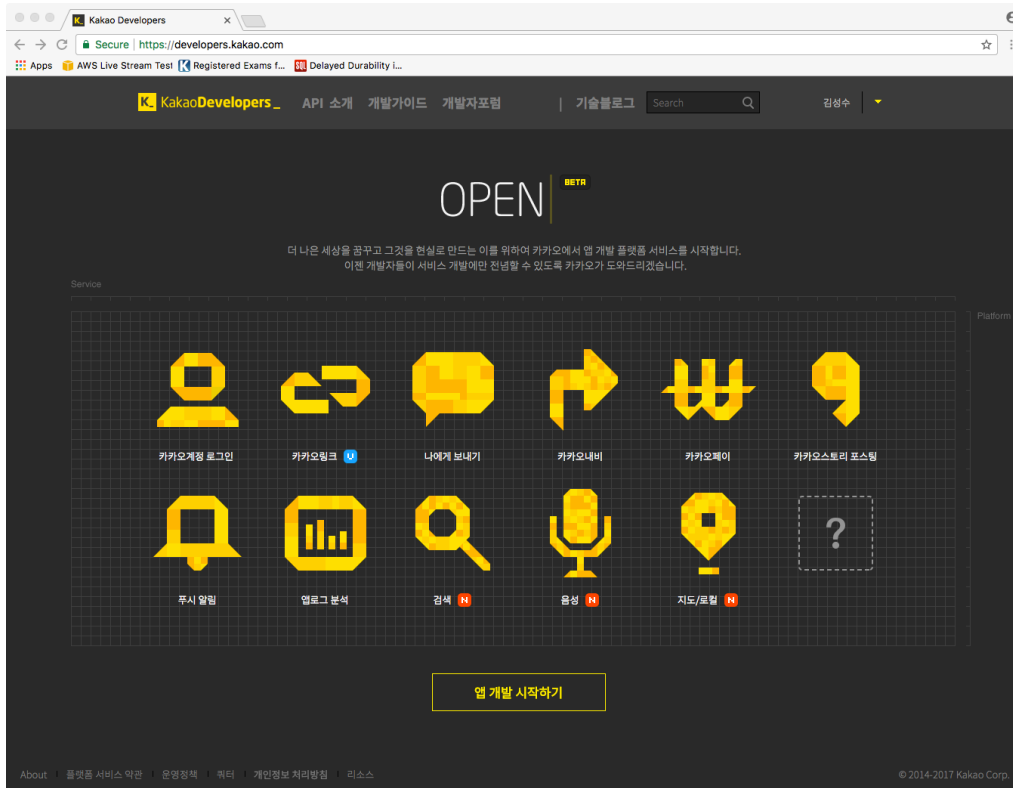
6. Create Distribution 버튼을 클릭하여 Cloudfront distribution 을 생성합니다. Distribution 생성에는 시간이 걸리므로 바로 다음 단계로 진행합니다.

카카오 개발자 등록 및 애플리케이션 설정하기

1. Kakao Login 을 비롯한 KakaoTalk 연동을 위해서는 Kakao 개발자 등록을 해야 합니다. Kakao 계정을 가지고 있으면 손쉽게 개발자 등록을 마칠 수 있습니다. 이 과정은 Kakao의 개발자 웹포털을 통해 진행할 수 있습니다. 개발자 등록을 마치고 개발자 포털에서 앱 개발 시작하기 버튼을 클릭하면 앱개발을 위한 페이지로 이동합니다.

Gaming on AWS

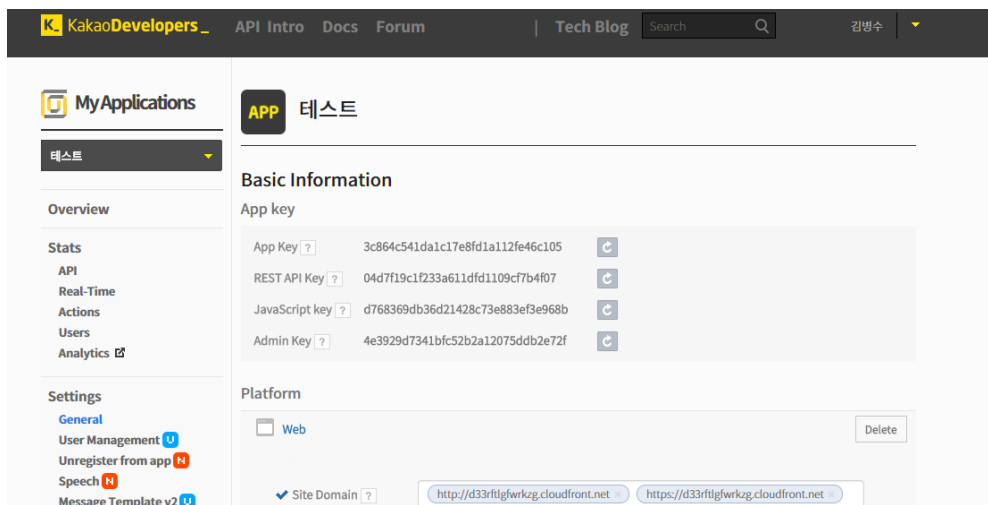
Serverless 로 게임 서비스 구현하기



2. 앱 만들기 버튼을 클릭하고 적절한 이름을 지정하면 Kakao 용 앱을 만들 준비가 완료됩니다.
3. App 을 생성한 뒤 애플리케이션을 선택하면 각종 애플리케이션 정보를 확인할 수 있습니다. 여기에서 앱정보 항목의 앱키를 선택해서 생성된 Kakao 의 Application Key 를 기록해둡니다. 이 키는 여러분의 애플리케이션 및 Lambda 함수에서 사용될 예정입니다.



4. 카카오쪽에서는 애플리케이션의 인증 도메인을 확인합니다. 이를 위하여 애플리케이션의 설정/개요 항목에서 앱정보 항목의 "설정된 플랫폼"을 통해 어떤 웹앱/모바일앱에서 인정 요청을 할 수 있는지 지정하게 됩니다. 사이트 도메인으로 위에서 생성한 Cloudfront distribution 의 domain name 을 입력합니다. 이때, http 와 https 두가지 모두 입력해 둡니다.

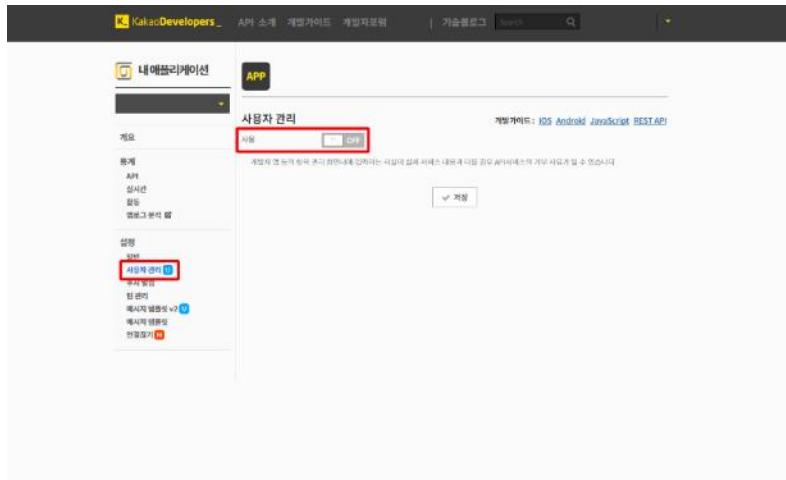


Gaming on AWS

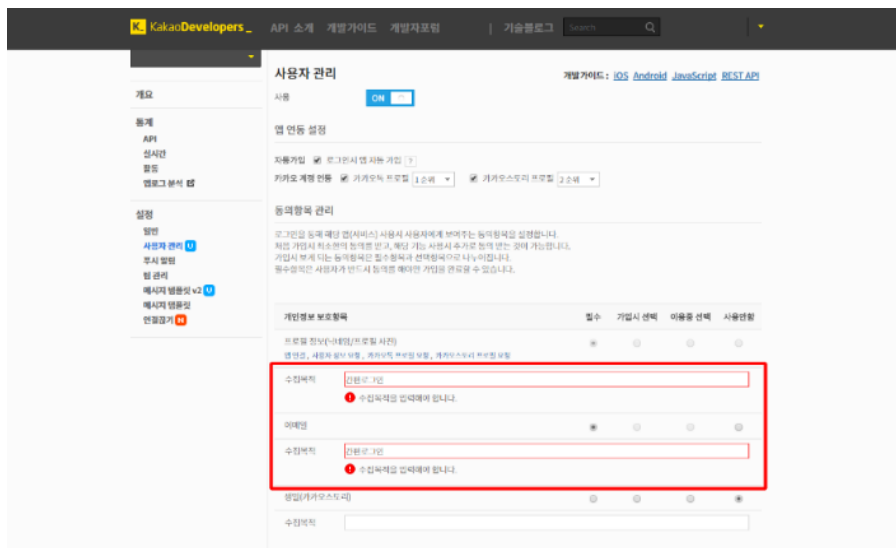
Serverless 로 게임 서비스 구현하기



5. 카카오 로그인을 활성화합니다. 좌측 설정 메뉴에서 사용자 관리 메뉴를 클릭하여 사용자 관리를 활성화합니다.

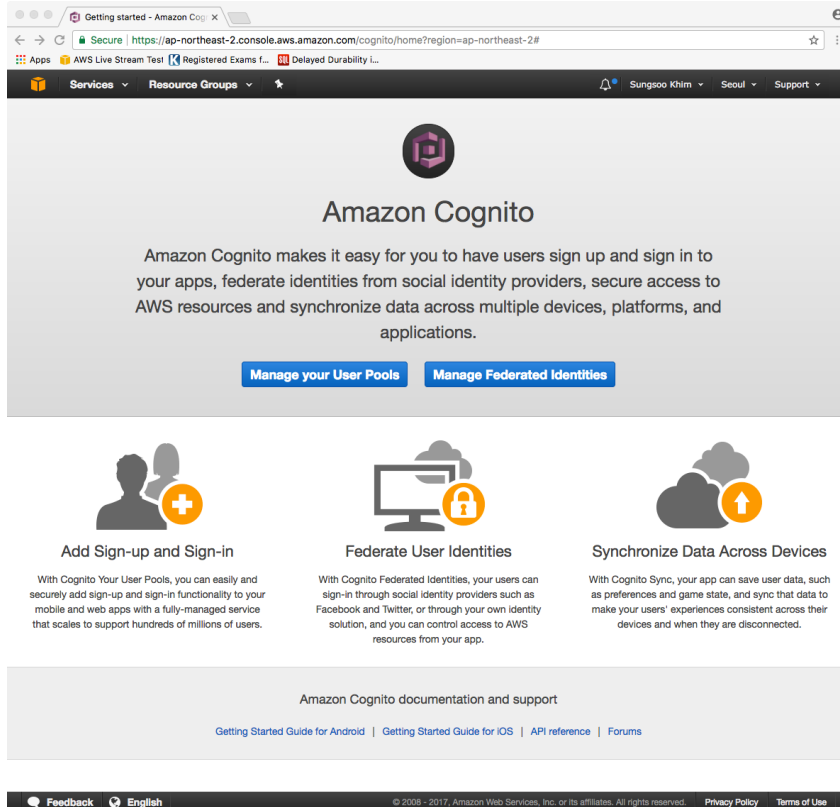


6. 개인 정보보호항목에서 수집목적을 입력해야 합니다. (간편로그인 연동 등.) 입력하지 않으면 활성화가 되지 않습니다.("사용자 관리 수정에 실패했습니다."라는 에러가 발생합니다.) 사용자 프로필 정보만 수집하는 것으로 하고 수집목적은 입력합니다.



Cognito 생성하기

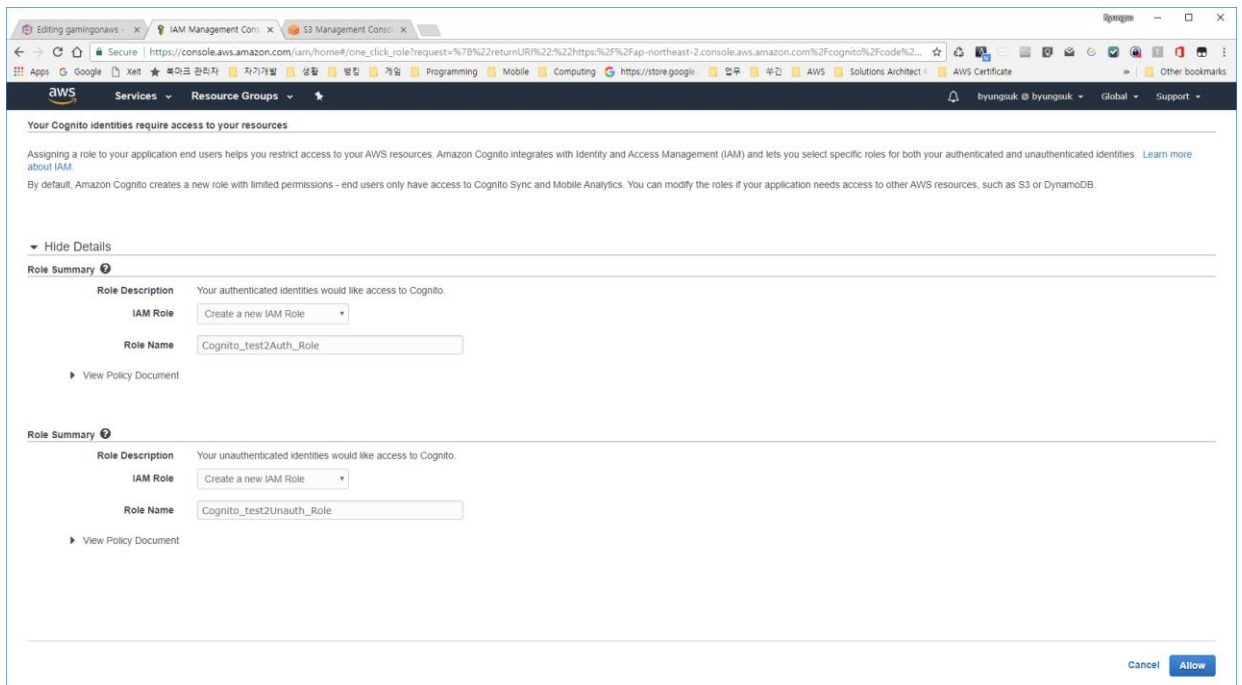
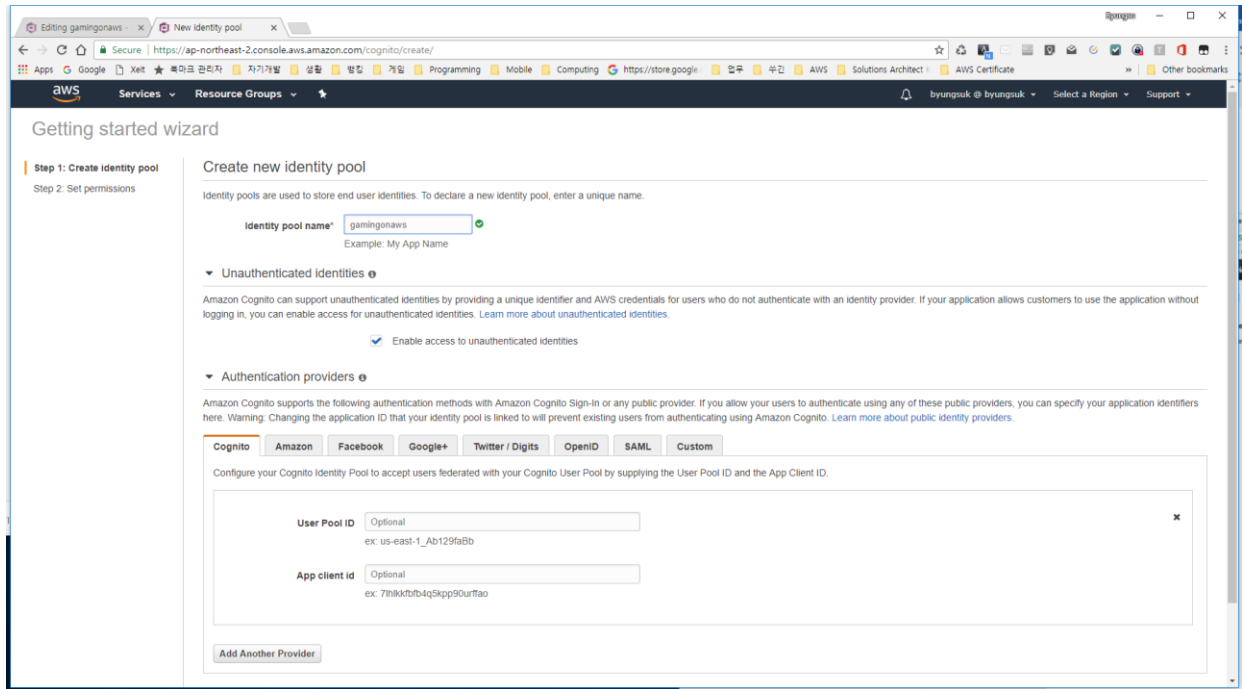
1. AWS Console 에서 Cognito 를 선택합니다.



2. Kakaotalk 연계를 위하여 Federated Identity 를 생성하여야 합니다. 오른쪽의 Manage Federated Identity 를 선택합니다.
3. 첫 페이지에서 Identity Pool name 을 gamingonaws 로 지정하고 Enable Access to Unauthenticated Identities 항목을 체크한 뒤, Create Pool 을 눌러 Identity Pool 을 생성해줍니다. Unauthenticated role, Authenticated role 권한에 맞는 IAM role 생성 화면이 나타나지만 여기서는 우선 Allow button 을 눌러 진행합니다. 이후 IAM Role 에서 필요한 권한을 추가할 것입니다.

Gaming on AWS

Serverless 로 게임 서비스 구현하기



4. Pool 을 생성하면 간단한 샘플 코드를 볼 수 있는 화면으로 이동합니다. 여기에서 화면 상단 우측의 Edit Identity Pool 링크를 클릭합니다. Dashboard 의 Edit identity Pool 로도 이동할 수 있습니다.

Gaming on AWS

Serverless 로 게임 서비스 구현하기

The first screenshot shows the 'Getting started with Amazon Cognito' page in the AWS IAM console. It includes sections for 'Download the AWS SDK', 'Get AWS Credentials', and 'Store User Data'. The 'Store User Data' section contains a code snippet for initializing the Cognito SDK and synchronizing data with the server.

```
// Initialize the Amazon Cognito credentials provider
CognitoCachingCredentialsProvider = new CognitoCachingCredentialsProvider(
    getApplicationContext(),
    "ap-northeast-2:ec9c3b33-e03e-45a0-a0c5-5abed1e774fb", // Identity pool ID
    Regions.AP_NORTHEAST_2 // Region
);

// Initialize the Cognito Sync client
CognitoSyncManager.syncClient = new CognitoSyncManager(
    getApplicationContext(),
    Regions.AP_NORTHEAST_2, // Region
    (credentialsProvider)
);

// Create a record in a dataset and synchronize with the server
Dataset dataset = syncClient.openOrCreateDataset("myDataset");
dataset.put("myKey", "myValue");
dataset.synchronize(new DefaultSyncCallback() {
    @Override
    public void onSuccess(Dataset dataset, List newRecords) {
        // Your handler code here
    }
});
```

The second screenshot shows the 'Identity pool' dashboard in the AWS IAM console. It displays statistics for the identity pool, including the number of identities, syncs, and storage. The dashboard also includes a chart showing the total identities over time and a section for resources and community links.

Metric	Value
Identities this month	0
Syncs this month	10
Current storage	0.3KB
Unauthenticated	0.0%
Developer Authenticated	100.0%

Filters: Total identities - Past 14 days -

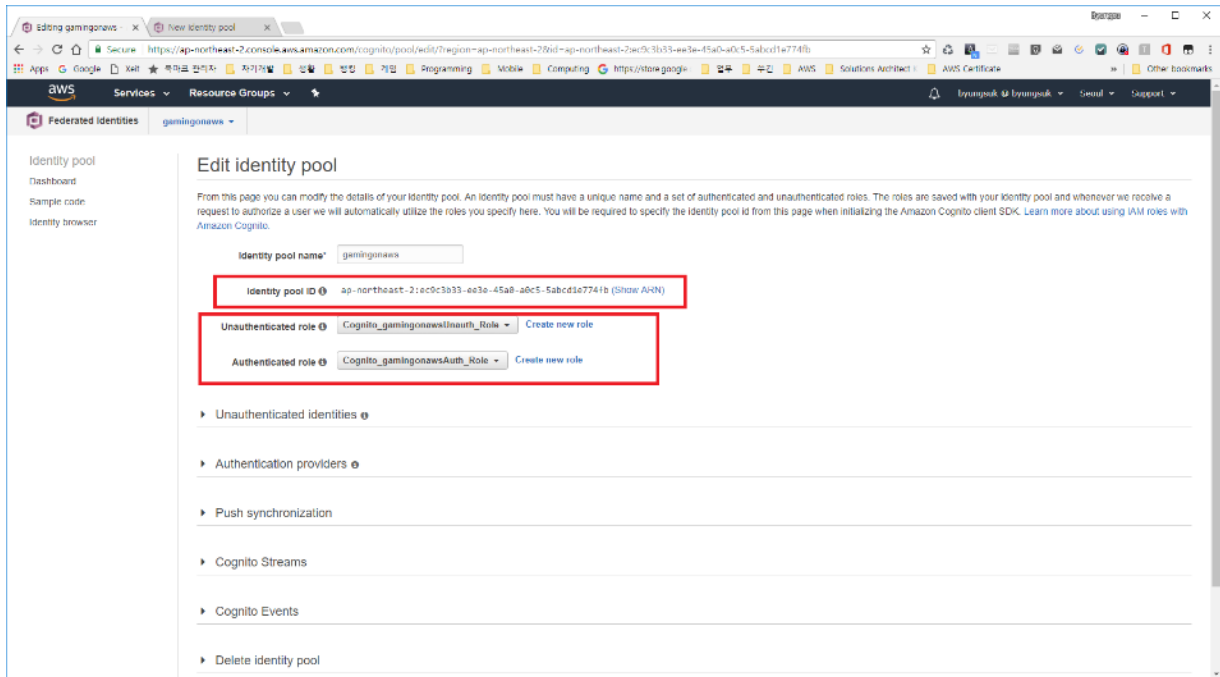
Resources:

- Getting started with Amazon Cognito
- Learn about the AWS Mobile SDKs
- Connect with the community

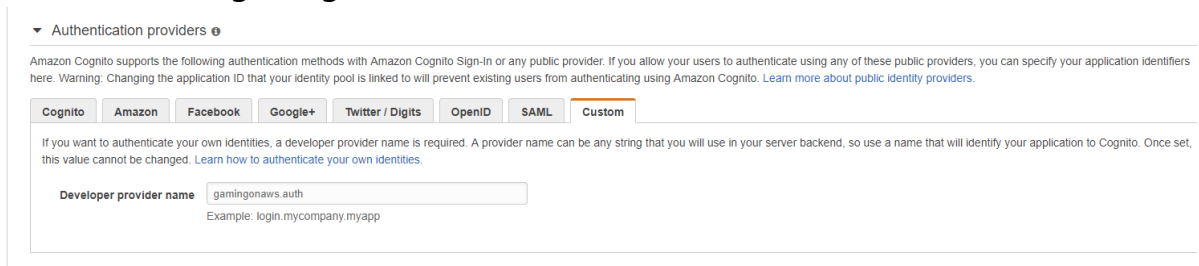
- 여기서 Identity Pool 의 각종 설정사항을 편집할 수 있습니다. 우선 Unauthenticated role, Authenticated role 의 이름을 확인합니다. 이 role 에 이후 필요한 권한들을 추가할 것입니다. 또한, Identity Pool ID 의 이름을 기록해 둡니다. 이후 설정에 필요합니다.

Gaming on AWS

Serverless 로 게임 서비스 구현하기



6. 아래 Authentication Providers 항목을 설정합니다. Kakao Login 은 Open ID 를 지원하지 않기 때문에 Custom 항목으로 인증을 처리해야 합니다. Custom Tab 을 열고 Developer Provide Name 을 gamingonaws.auth 로 지정한 후 저장합니다.



Lambda 및 API Gateway 생성하기

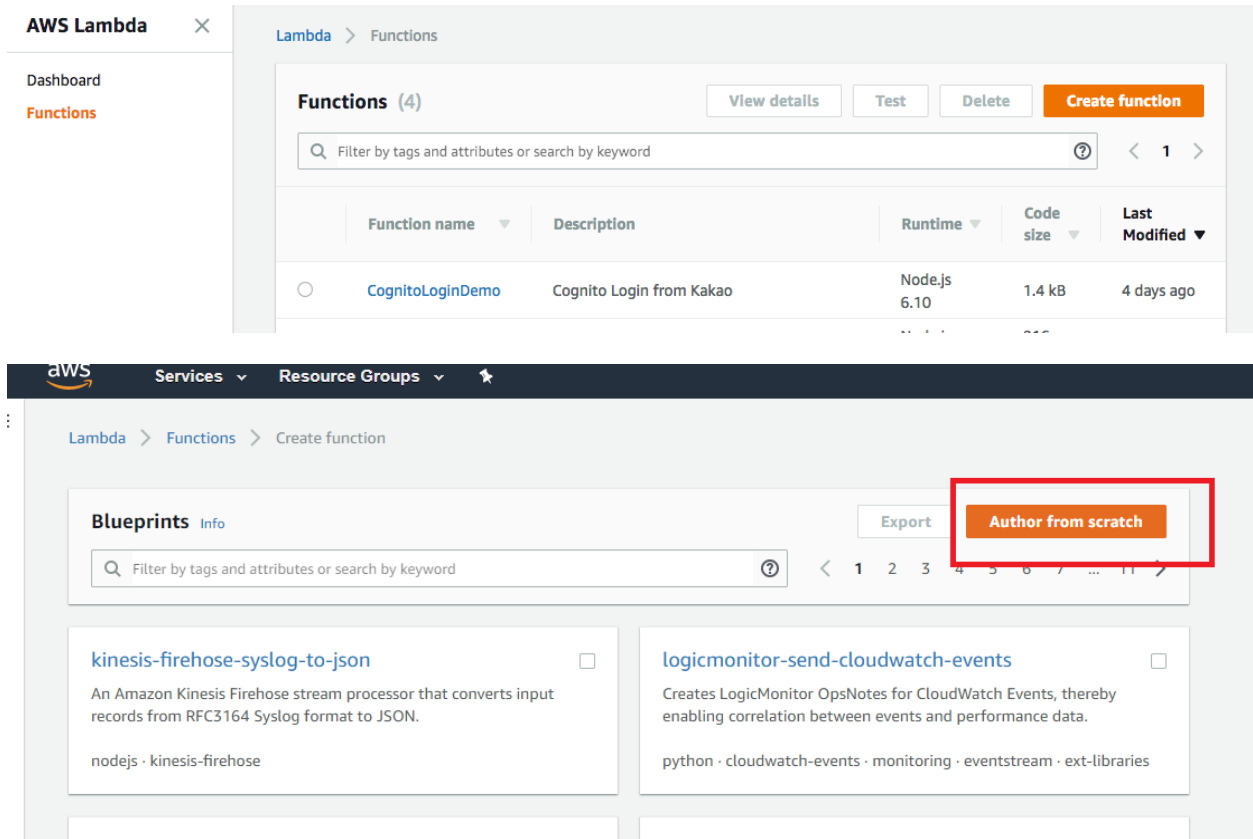
Cognito 및 Kakao 인증을 처리하기 위한 Lambda 함수와 API Gateway 를 생성합니다.

Lambda 함수 생성하기

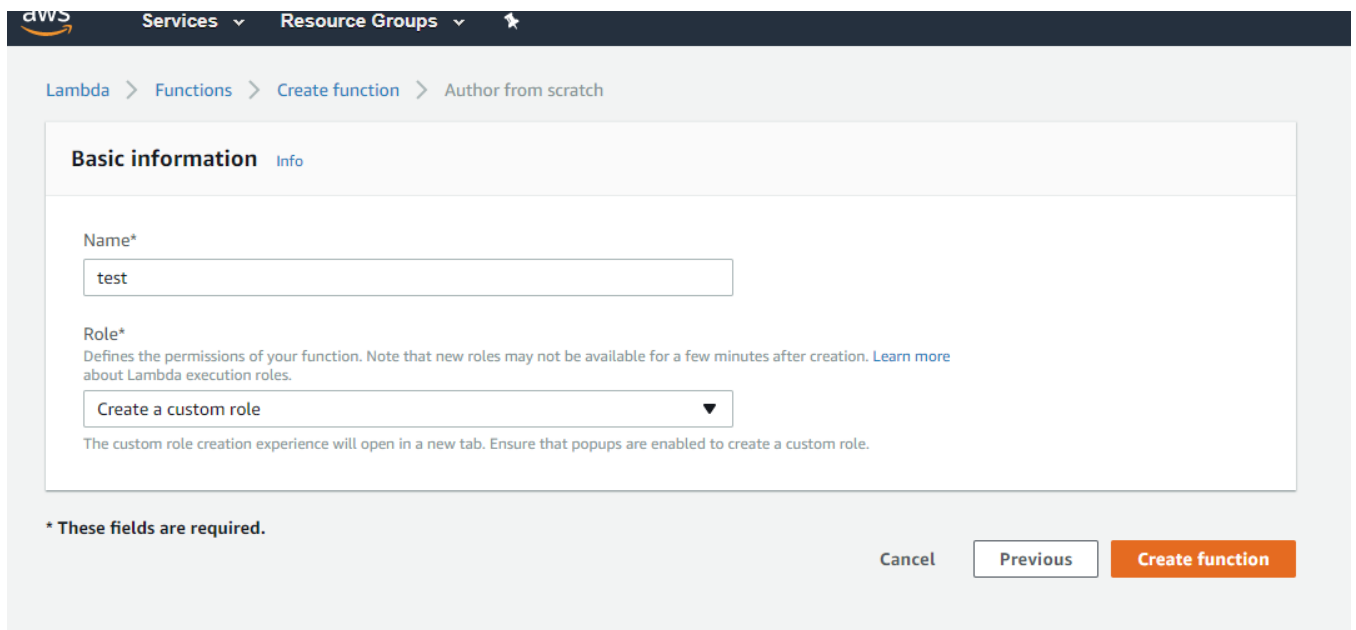
1. AWS Management Console 에서 Lambda 를 선택합니다. Author from scratch 로 빈함수를 생성합니다.

Gaming on AWS

Serverless 로 게임 서비스 구현하기



2. Role 은 Create a custom role 을 선택하여 Lambda 에 부여할 IAM role 을 설정합니다.



Serverless 로 게임 서비스 구현하기

AWS Lambda requires access to your resources

AWS Lambda uses an IAM role that grants your custom code permissions to access AWS resources it needs.

▼ Hide Details

Role Summary ?

Role Description Lambda execution role permissions

IAM Role

Policy Name

► View Policy Document

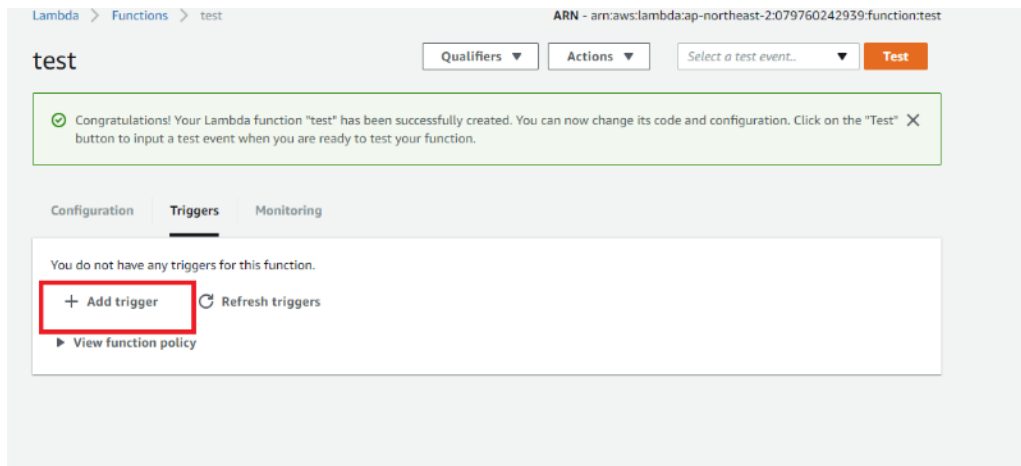
3. View Policy Document 를 클릭하고 Edit 를 선택하여 다음과 같이 Cognito 인증을 위한 권한을 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cognito-identity:*",
        "cognito-idp:*",
        "cognito-sync:*",
        "iam:ListRoles",
        "iam:ListOpenIdConnectProviders",
        "sns:ListPlatformApplications"
      ],
      "Resource": "*"
    }
  ]
}
```

Gaming on AWS

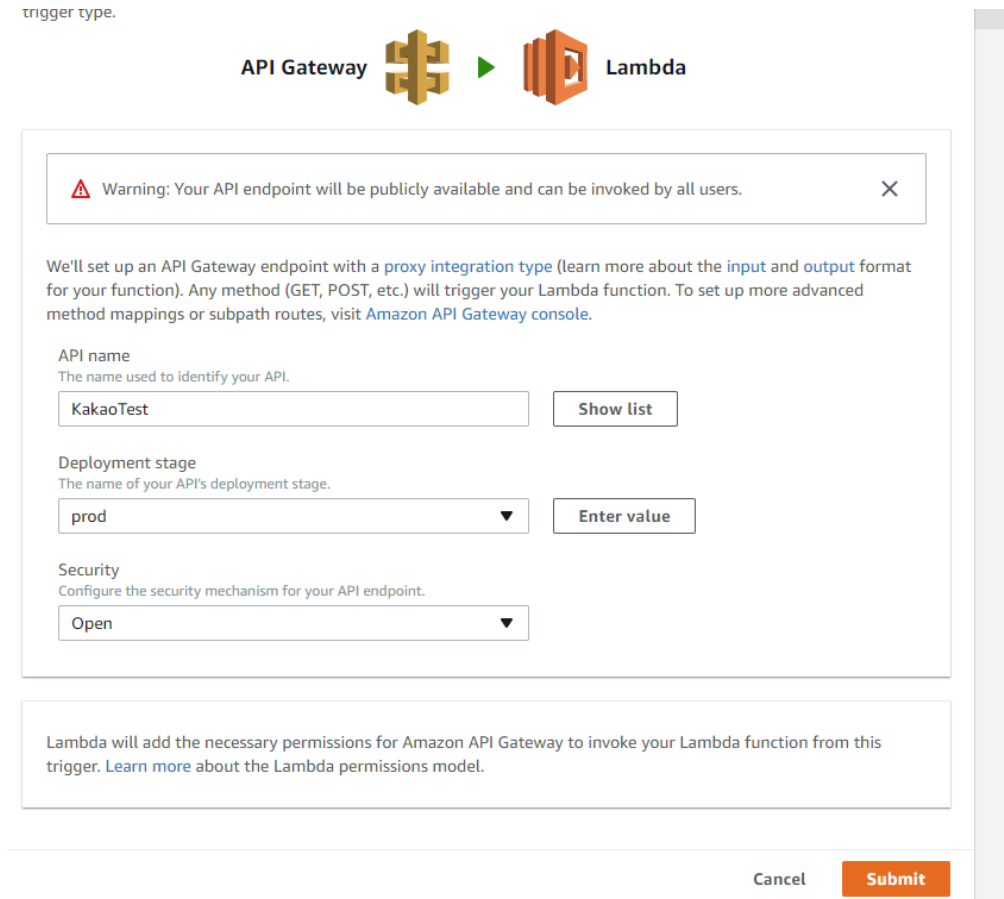
Serverless 로 게임 서비스 구현하기

4. 생성된 Lambda 함수에서 Triggers Tab 에서 Add trigger 로 API Gateway 를 지정합니다.



5. API name 을 Enter value 버튼을 클릭하여 지정하고, Security 는 Open 으로 지정합니다.
(실제 서비스에서는 보안 강화를 위하여 Secret Key 나 IAM 을 사용하도록 설정할 수 있습니다.)

trigger type.



6. 다시 Configuration Tab 으로 돌아와 아래 코드를 입력합니다. 아래 코드에서 <identity-pool-id>를 위 단계에서 기록해둔 ID 로 대체합니다. 만일 developer provider name 도 gamingonaws.auth 가 아닌 다른 이름으로 지정하였다면 해당 이름으로 변경합니다.

```
var AWS = require('aws-sdk');
var cognitoidentity = new AWS.CognitoIdentity();

exports.handler = (event, context) => {
  // Sample code to act as auth server with Kakao Login

  var authInfo = require('querystring').parse(event.querystring);

  var userid = authInfo['id'];
  var access_token = authInfo['access_token'];
  var expires = authInfo['expires_in'];

  var http = require('https');

  var options = {
    host: "kapi.kakao.com",
    path: "/v1/user/access_token_info",
    headers: {"Authorization": "Bearer " + access_token}
  };

  console.log("start Kakao auth confirm");

  http.get(options, function(res) {
    console.log("Got response:" + res.statusCode);
    if (res.statusCode == 200) {
      // Only enter here when Kakao gave us a confirmation

      var kakaoresponse = "";
      res.on("data", function(chunk) {
        kakaoresponse += chunk;
      });
    }
  });
}
```

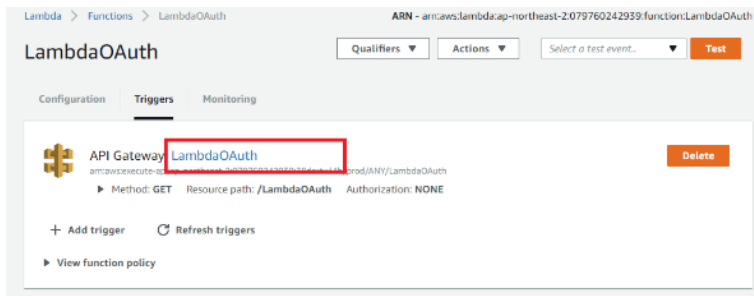
```
    })  
    res.on("end", function() {  
        var obj = JSON.parse(kakaoreponse);  
  
        var useridfromkakao = obj.id;  
        var expireinMillis = obj.expireinMillis;  
        var appld = obj.appld;  
  
        var params = {  
            IdentityPoolId: '<identity-pool-id>', /* required */  
            Logins: { /* required */  
                'gamingonaws.auth': useridfromkakao.toString()  
            },  
            TokenDuration: 1000  
        };  
        console.log("userid" + useridfromkakao);  
        console.log(expireinMillis);  
        console.log(appld);  
        cognitoidentity.getOpenIdTokenForDeveloperIdentity(params, function(err, data) {  
            if (err) console.log(err, err.stack); // an error occurred  
            else {  
                console.log(data); // successful response  
                context.done(null, data);  
            }  
        });  
        // End of Cognito call  
    });  
    } else {  
        // Because KAKAO returned 400 or 401 error - means auth check failed  
    }  
    });  
};
```

- 이 코드는 사용자의 Kakao 정보를 전달 받으면, 해당 정보의 Validity 를 Kakao 에 확인하는 코드입니다. 만일 인증이 되면 Cognito 에 사용자 Token 을 발급하고, 해당 정보를 클라이언트에 전달합니다. 실패하는 경우에는 Fail 을 돌려주게 됩니다.

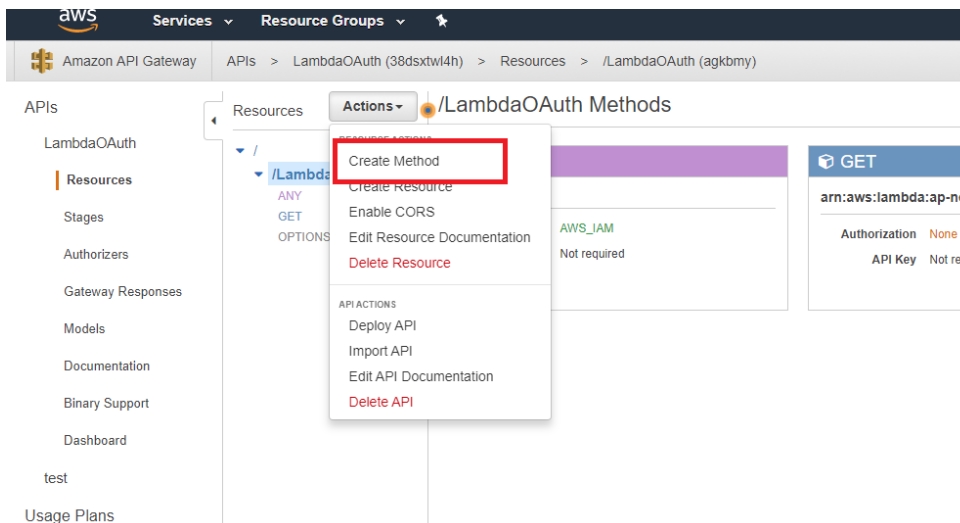
만일 함수의 실행에 관련된 로그를 확인하고 싶을 경우에는 `console.log()` 함수를 통해 로그를 하시고 CloudWatch 를 통하여 로그 내용을 확인 하실 수 있습니다. 이 함수는 실습을 위한 최소 기능만 구현한 함수이므로 참조로만 활용하실 것을 권합니다.

API Gateway 생성하기

1. Lambda 함수 생성시 Trigger 로 생성한 API Gateway 항목을 설정하기 위하여 AWS Management Console 에서 API Gateway 항목으로 이동하거나 Lambda 의 Triggers Tab 에서 링크를 클릭하여 이동합니다.



2. API Gateway 항목의 Resource 의 Dropdown 메뉴에서 Create method 항목을 선택합니다.



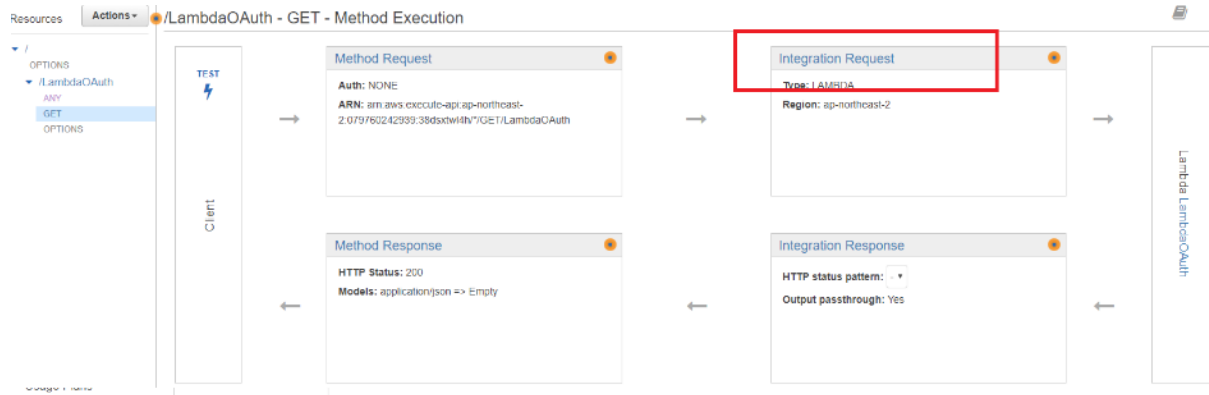
3. Dropdown 메뉴에서 먼저 OPTIONS 메소드를 선택한 후 옆의 체크버튼을 클릭하여 OPTIONS 메소드를 추가합니다. OPTIONS 에서는 Integration Type 을 Mock 으로 선택합니다. Mock 형태이므로 그대로 저장하면 됩니다.

4. 마찬가지로 GET method 를 추가합니다. Integration Type 을 Lambda 로 선택하고, Region 은 Lambda 함수를 생성한 Region 을 선택합니다. Seoul 을 선택하였다면 ap-northeast-2 가 됩니다. Lambda Function 은 위에서 생성한 함수 이름을 입력하고 저장합니다. (자동완성을 지원합니다.)

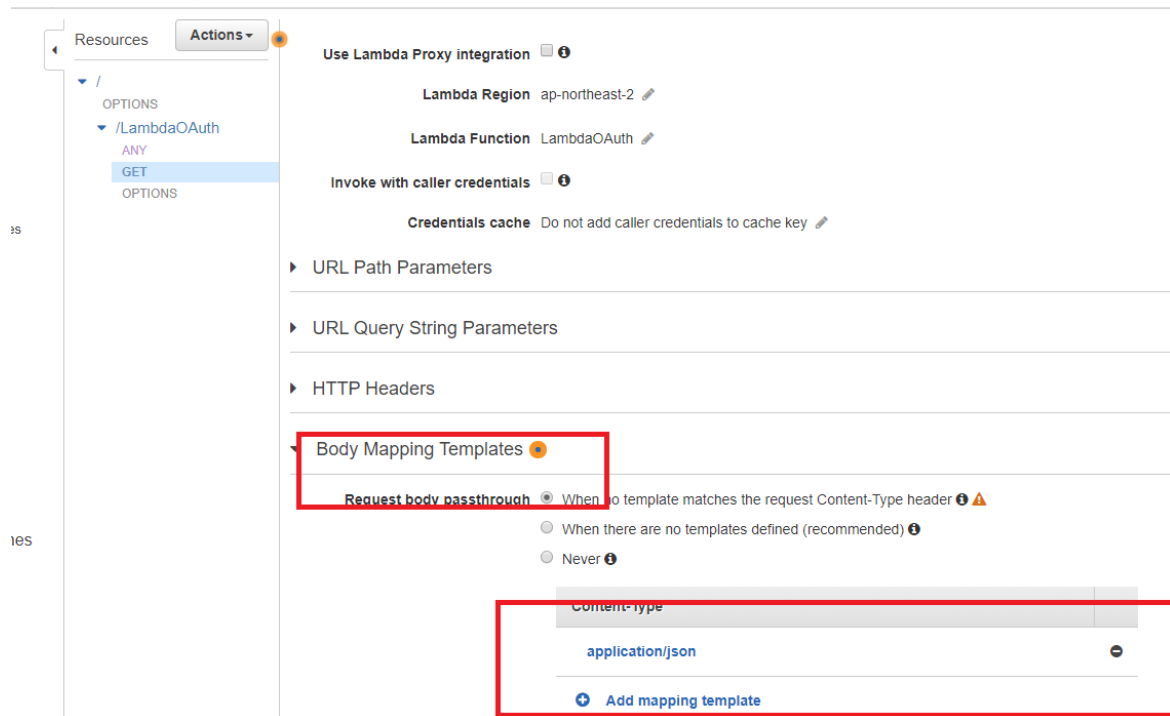
5. 이제 GET method 의 상세 사항을 설정하겠습니다. GET method 를 선택하여 Integration Request 항목을 클릭합니다.

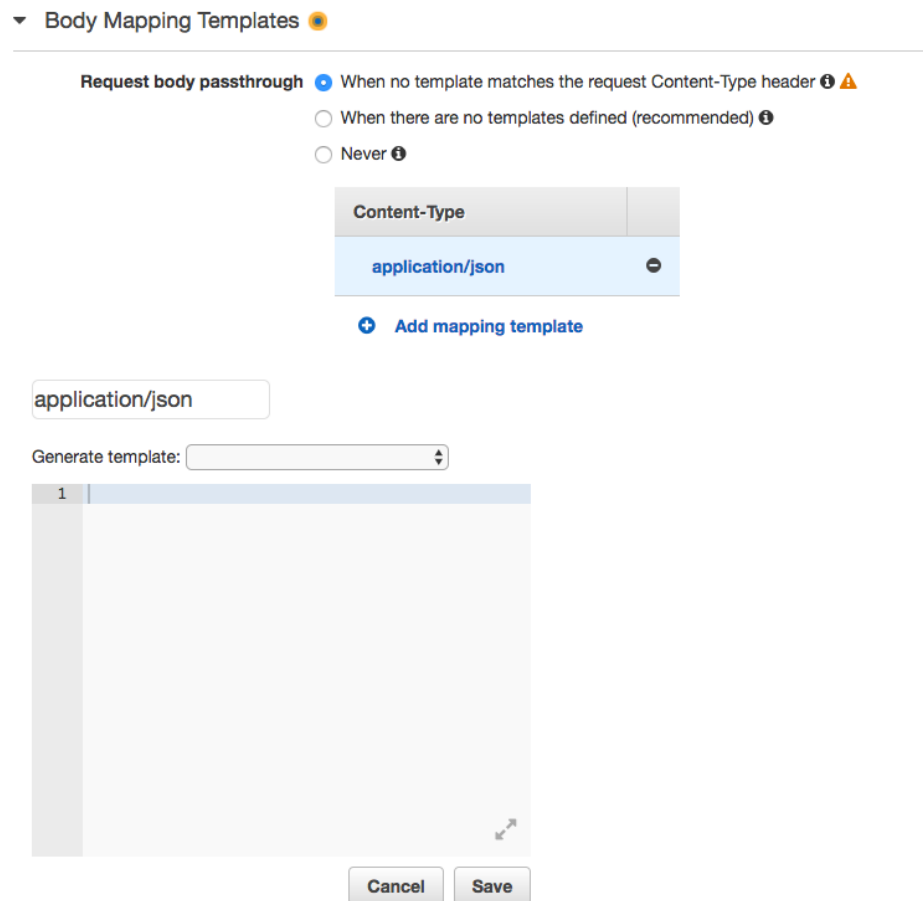
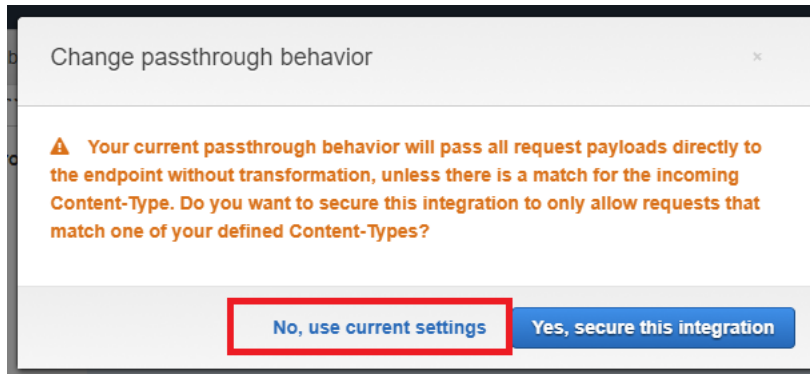
Gaming on AWS

Serverless 로 게임 서비스 구현하기



6. 하단의 Body Mapping Template 항목을 열고 Add mapping template 을 눌러 새로운 Mapping 을 추가합니다. Content-Type 은 application/json 을 입력합니다. 추가할때 아래와 같은 경고창이 나오면 **No, use current settings** 를 선택해줍니다. 그러면 아래와 같이 Template 를 입력하는 창이 나옵니다.



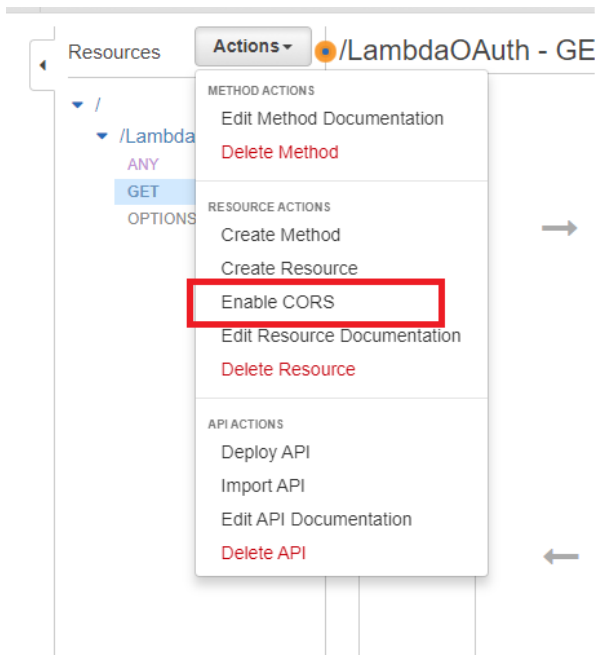


7. 여기에 아래 코드를 입력하고 Save 를 누릅니다.

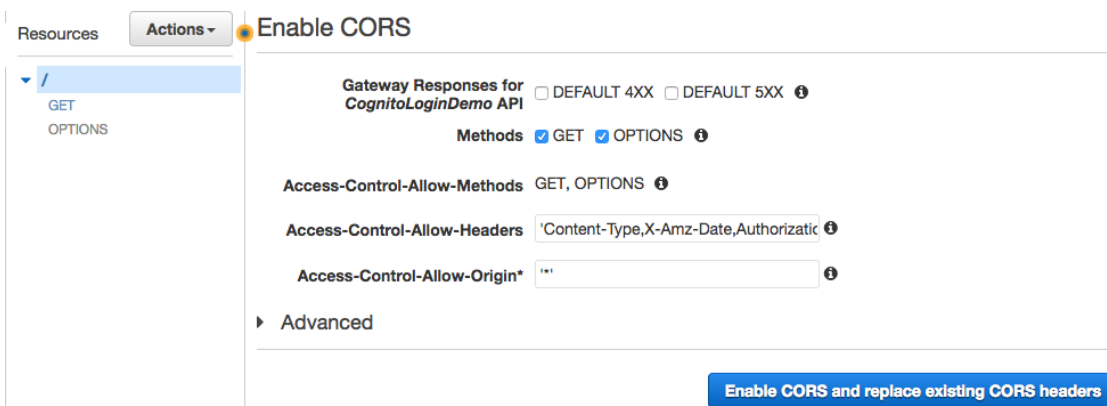
```
{
  "querystring" : "#foreach($key in $input.params().querystring.keySet())#if($foreach.index > 0)&#end$util.urlEncode($key)=$util.urlEncode($input.params().querystring.get($key))#end",
```

```
"body" : $input.json('$')
}
```

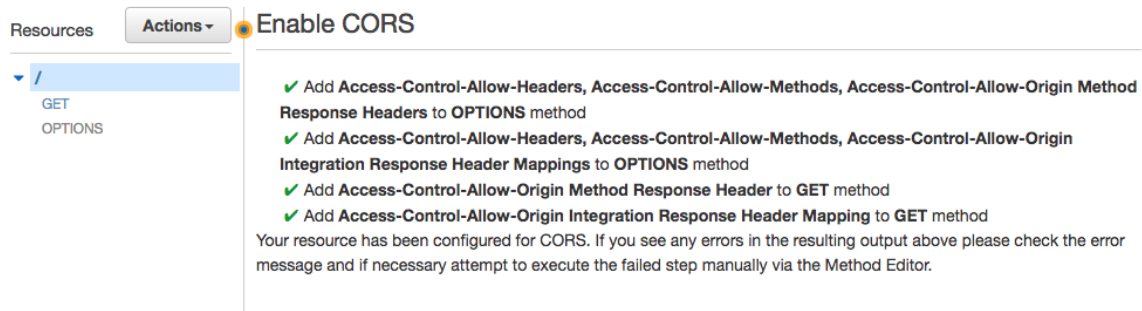
- 이 코드는 API Gateway 로 전달되는 GET Request 에 대하여 Lambda 에 querystring 이라는 변수로 전달하는 Template 입니다. 이후 조금 더 정교하게 수정을 하고 싶을 경우에는 Template 언어가 Apache 의 Velocity Project 의 Velocity Template Language 이므로 관련 정보를 참고하시면 됩니다.
- 8. 마지막으로 이 API Gateway 를 애플리케이션에서 호출할 수 있도록 CORS(Cross Origin Resource Sharing)설정을 하도록하겠습니다. Actions 메뉴에서 Enable CORS 를 선택합니다.



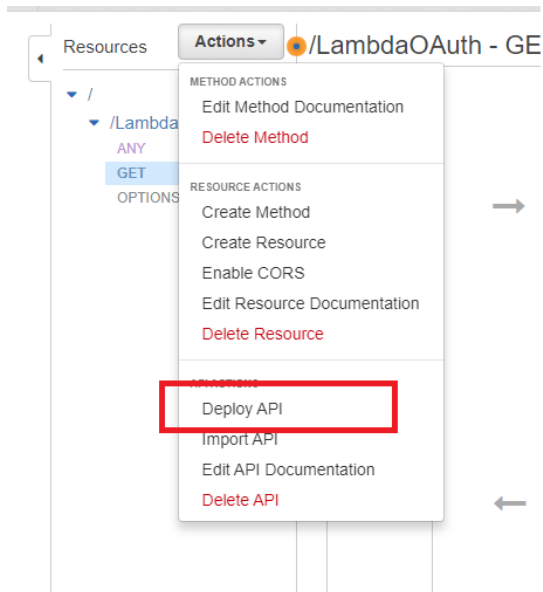
- 9. 기본값으로 Enable CORS and replace existing CORS headers 버튼을 눌러 설정합니다.



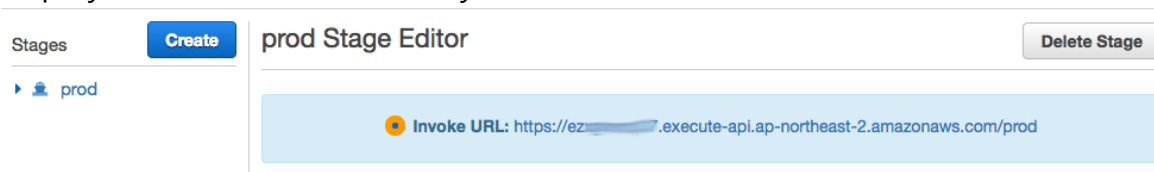
Serverless 로 게임 서비스 구현하기



10. 이제 API 를 Deploy 합니다. Action menu 에서 Deploy API 항목을 선택하고 Deployment stage 는 prod 를 선택하여 Deploy 합니다.



11. Deploy 가 완료되면 API Gateway 의 Invoke URL 이 나옵니다.



12. Invoke URL 을 기록해둡니다. (만약 API 를 root 가 아닌 별도의 이름으로 설정하였다면, /prod 뒤에 해당 API 의 resource path 까지 붙여서 기록해둡니다.)

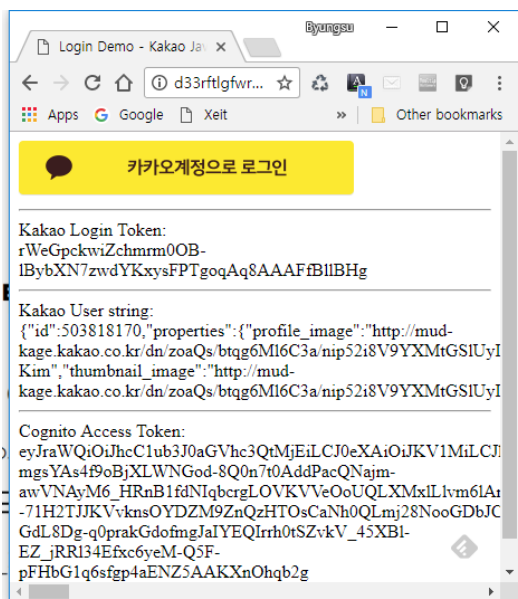
카카오 인증 테스트하기

웹에서 테스트

1. 제공된 Source code 의 root directory 의 oauth.html 과 script 폴더의 OAuth.js 를 위에서 생성한 S3 Bucket 에 업로드하여야 합니다. 업로드하기전에 OAuth.js 를 먼저 편집합니다. API_URL 은 API Gateway Invoke URL + resource path 형태로 설정하시거나 Lambda 함수의 trigger 항목에서 method 항목을 펼치면 나오는 invoke URL 을 사용하여 설정합니다.

```
var CONFIG = {
  "KakaoAppId": "<Kakao Javascript AppKey>",
  "API_URL": "API Gateway prod endpoint URL(+resource path if not root) "
}
```

2. S3 bucket 의 root 경로에 oauth.html 을, script 폴더를 생성하고 해당 폴더에 수정한 OAuth.js 를 업로드합니다.
3. browser 에서 cloudfront domainname/oauth.html 로드합니다.
4. 카카오계정으로 로그인 버튼을 눌러 Token 발급 및 필요정보들이 출력되는지 확인합니다.



5. 만약 설정 등의 문제로 OAuth.js 파일을 수정하여 다시 업로드 할 경우 Cloudfront 에 이미 caching 된 OAuth.js 로 인해 수정사항이 바로 적용되지 않습니다. Cloudfront distribution detail 에서 Invalidation 항목에서 Create invalidation 버튼을 클릭하고 /script/OAuth.js 를 입력하여 cache 된 OAuth.js 를 invalidate 시키고 invalidation 이 완료되면 다시 테스트를 진행합니다.

NWJS 로 테스트

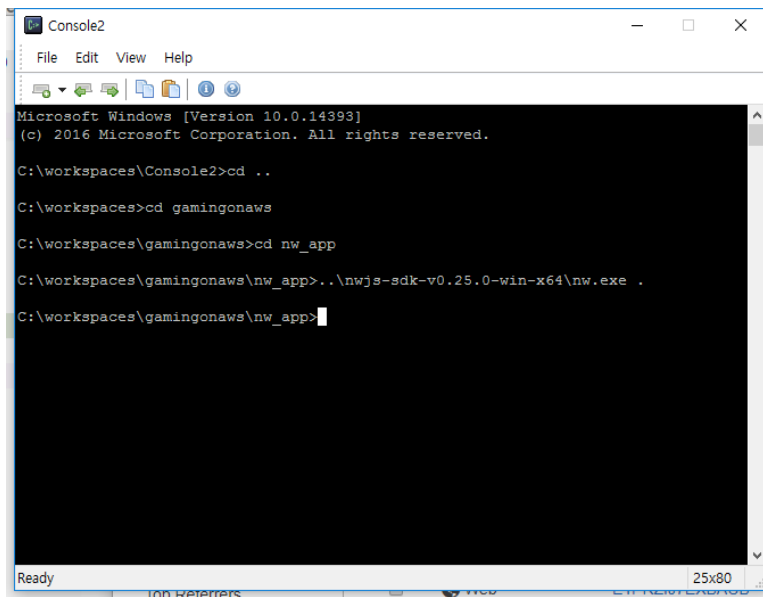
1. nw_app 폴더로 이동합니다.
2. Script 폴더의 config.json 을 열어 필요항목을 편집합니다.

```
{
  "bucketname": "<s3 bucket name>",
  "uploadprefix": "update",
  "region": "ap-northeast-2",
  "api_url": "<api_gateway prod endpoint url(+method name if specified)>",
  "IdentityPoolId": "cognito Identity Pool ID",
  "cloudfrontDistributionId": "cloudfrontDistributionId(not domain name randomized string id) ",
  "cloudfrontURL": "http://<cloudfrontdomainname>"
}
```

3. Cmd / terminal 창에서 nw_app 폴더로 이동하여 다음과 같이 커맨드를 입력합니다.
(여기서는 nw_app 과 같은 depth 의 nwjs-sdk 폴더에 nwjs 파일을 압축해제하였다고 가정합니다. Mac 의 경우, nwjs 를 nwjs.app/Contents/MacOS/nwjs . 와 같이 실행합니다.)
 - Windows

Gaming on AWS

Serverless 로 게임 서비스 구현하기



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\workspaces\Console2>cd ..

C:\workspaces>cd gamingonaws

C:\workspaces\gamingonaws>cd nw_app

C:\workspaces\gamingonaws\nw_app>.\nwjs-sdk-v0.25.0-win-x64\nw.exe .

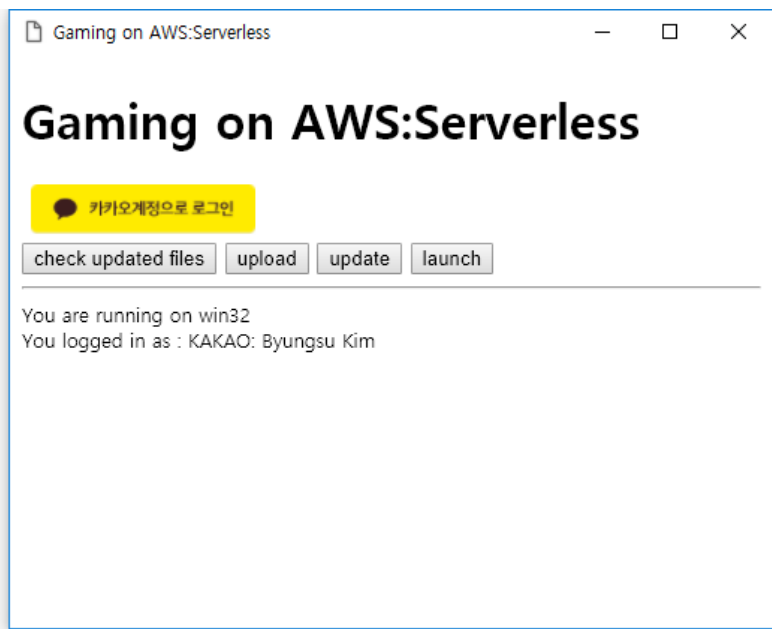
C:\workspaces\gamingonaws\nw_app>
```

○ MacOS



```
186590d03d3d:nw_app $ pwd
/Users/186590d03d3d/Documents/Amazon/GamingOnAWS/nw_app
186590d03d3d:nw_app $ ./nwjs-sdk-v0.25.0-osx-x64/nwjs.app/Contents/MacOS/nwjs .
```

4. Desktop Application 에서 카카오 인증을 진행하고 결과를 확인합니다.



Lab2. S3 와 Cloudfront 로 클라이언트 배포 서비스 제작하기

시나리오

게임서비스에서는 지속적인 게임 클라이언트의 배포 및 업데이트가 필요합니다. 이 실습에서 클라이언트 업데이트 과정은 다음과 같습니다.

1. nw_app 폴더에 새 클라이언트 리소스를 추가하면, nw_app 에서 file 들의 checksum 을 확인하여 업데이트 여부를 체크하고 업데이트된 파일들을 S3 에 업로드합니다.
2. 이미 S3 생성시에 versioning 기능을 설정하였으므로 각 object 들의 변경사항은 version 별로 S3 에 기록되게 됩니다.
3. nw_app 에서는 S3 업로드시 반환받은 version_id 를 metadata 로 생성하여 S3 에 업로드합니다.
4. 업로드가 완료되면 cloudfront distribution 의 version.json 파일(만) invalidation 하여 클라이언트 배포를 마치게 됩니다.
5. 클라이언트에서 버전을 체크할 때 version.json 을 받아 각 object 의 version_id 를 포함하여 cloudfront 에 요청합니다.
6. Cloudfront distribution 생성시에 이미 versionId query string 을 origin 에 포워딩하고 query string 을 포함하여 캐싱하도록 설정하였으므로 cloudfront 는 각 object 의 최신 version_id 로 object 를 다운로드 받게 됩니다.

여기서는 실습의 편의를 위하여 KakaoTalk 로그인 사용자가 클라이언트 패치를 생성하고 업로드할 수 있는 관리자인 것으로 상정합니다. 실제 게임 서비스에서는 클라이언트 패치 생성 및 업로드 권한을 분리하거나 별도의 애플리케이션으로 제작하여야 합니다. 또한 실제 서비스에서 비용 절감 및 성능 향상을 위하여 사용할 수 있는 파일 버전별 binary diff 생성을 통한 differential patch 등의 기능이나 리소스를 강제 삭제하거나 클라이언트 업데이트의 자체 패치 등 고급기능은 여기서 다루지 않습니다.

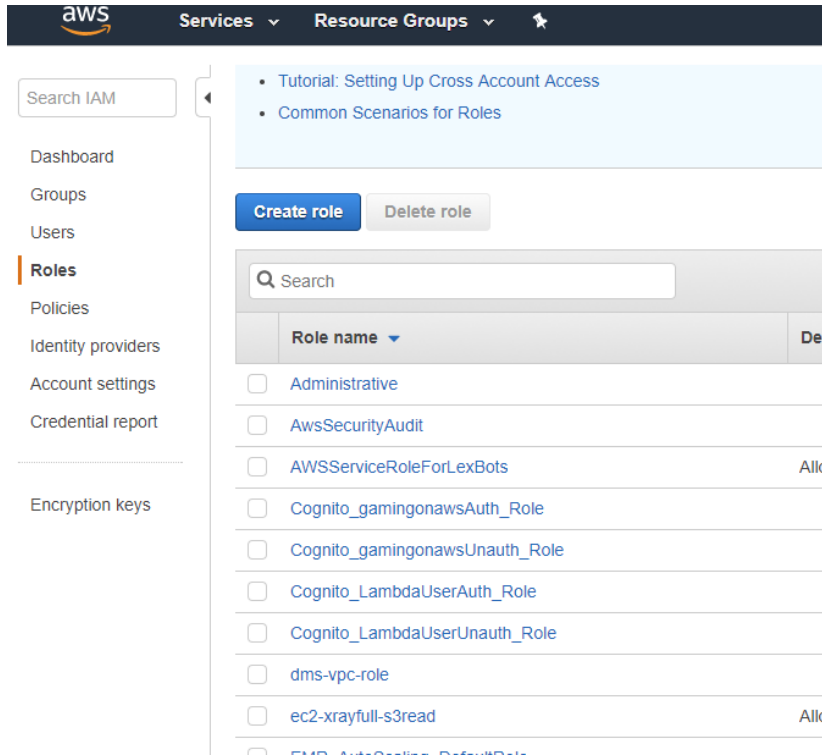
IAM Role 에 권한 추가하기

Cognito 인증 사용자에게 필요한 권한을 추가하는 작업이 필요합니다. 단계별 지침은 다음과 같습니다.

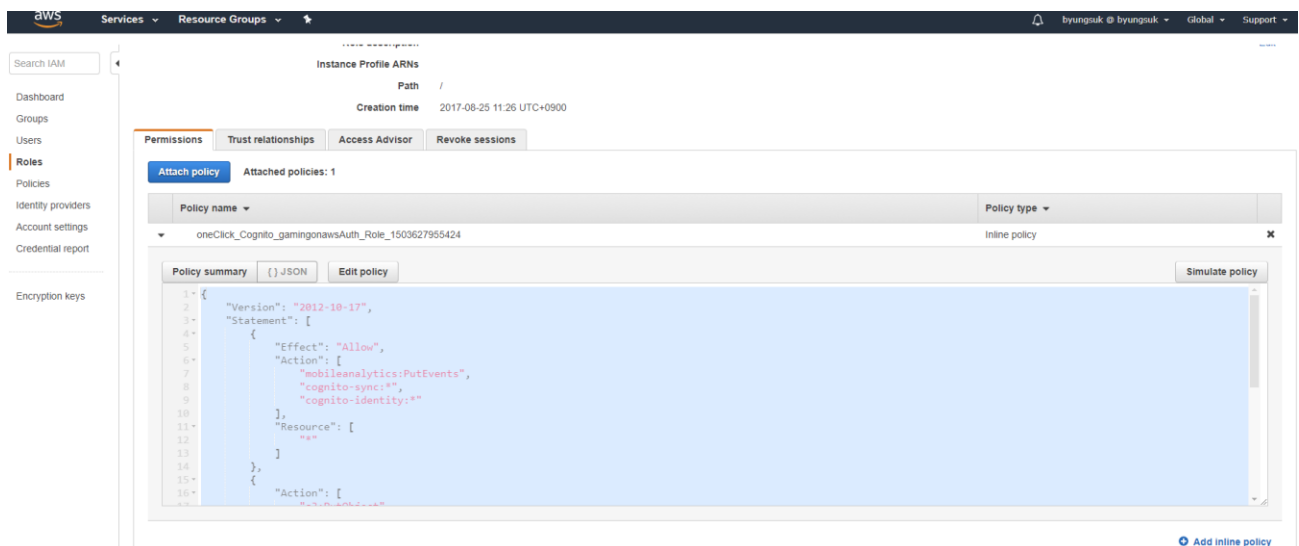
Gaming on AWS

Serverless 로 게임 서비스 구현하기

1. AWS management console 의 Services 에서 IAM 을 선택합니다.
2. 왼쪽 메뉴에서 Roles 메뉴를 선택합니다.
3. Role name 중 Cognito 생성시에 함께 생성된 Coginto_XXX_Auth_role 을 선택합니다.



4. Permission Tab 에서 policy 를 확장하고 Edit policy 버튼을 눌러 다음 코드를 입력합니다.
코드 입력시에 <bucket_name> 항목을 생성한 bucket 이름으로 설정합니다.

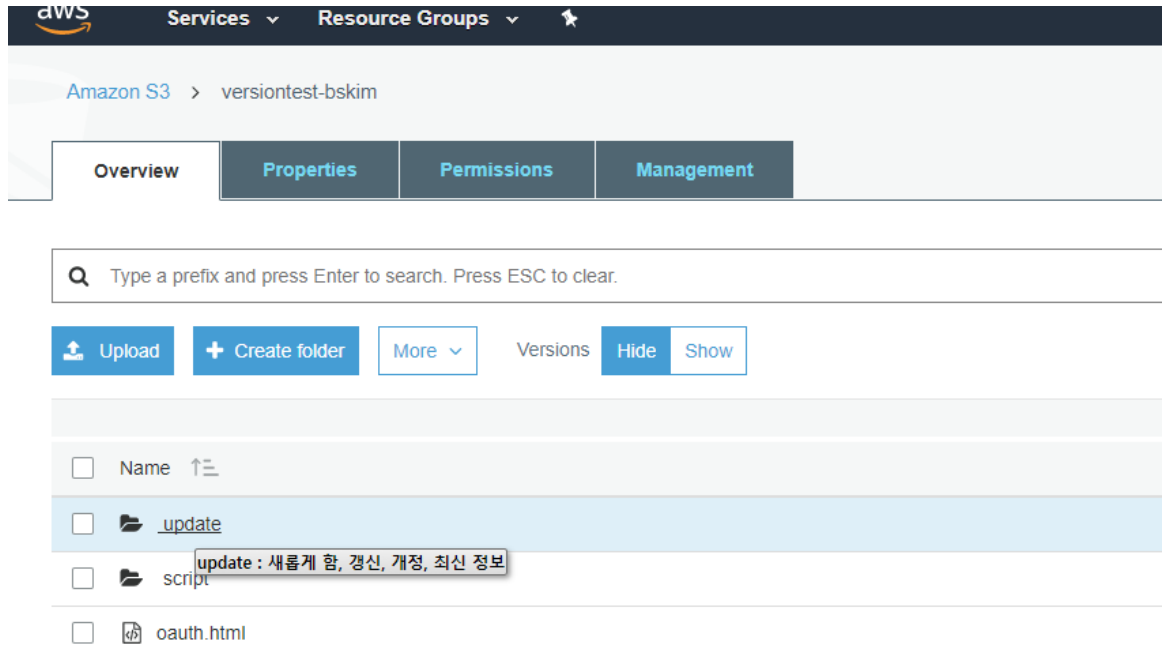


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "mobileanalytics:PutEvents",
        "cognito-sync:*",
        "cognito-identity:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::<bucket_name>/*"
      ]
    },
    {
      "Action": [
        "cloudfront:CreateInvalidation",
        "cloudfront:GetInvalidation"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

5. Validate policy 버튼을 눌러 문제가 없는지 확인하고 저장합니다.

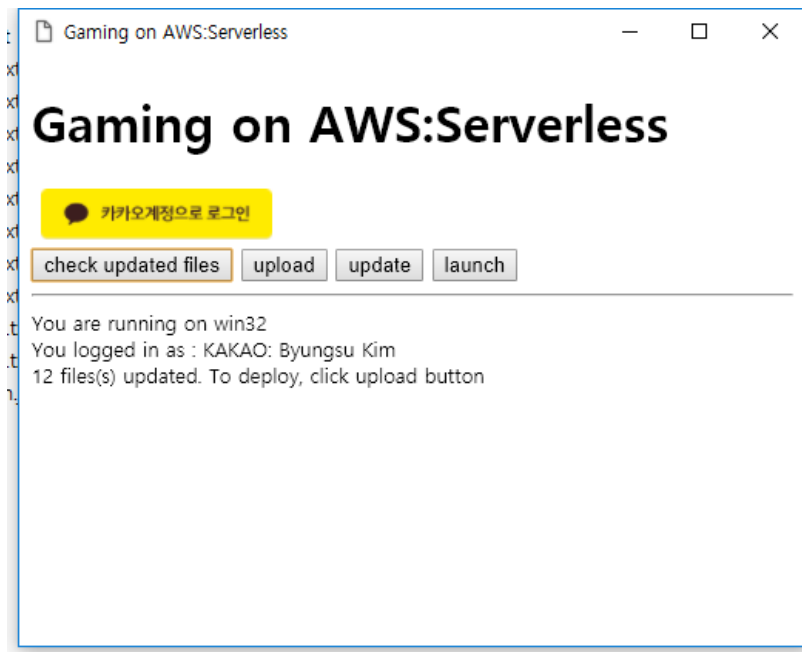
S3 버킷에 클라이언트 업데이트용 path 추가하기

위에서 생성한 S3 bucket 에 클라이언트 배포를 위한 folder 를 추가합니다. 여기서는 update 라는 path 를 사용합니다.



테스트하기

1. 위에서 설정한 nw_app 을 실행합니다. Config.json 에 cloudfront distribution id 및 domain name, s3 bucket name 이 잘 설정되어 있는지 다시 확인합니다.
2. nw_app 의 patch 폴더에 client resource 파일 테스트를 위한 임의의 파일 및 폴더를 구성합니다. 여기서는 test1.txt, test 폴더 생성 후 test2.txt 를 테스트로 사용한다고 가정합니다. 각 test1.txt 에는 임의의 내용을 입력합니다.
3. nw_app 카카오 계정 로그인인 된 상태에서 check updated files 버튼을 클릭하면 애플리케이션이 patch 폴더 및의 파일과 subfolder 들을 탐색하여 업데이트 내용을 체크합니다. patch 폴더에 version.json 파일이 생성된 것을 확인합니다.

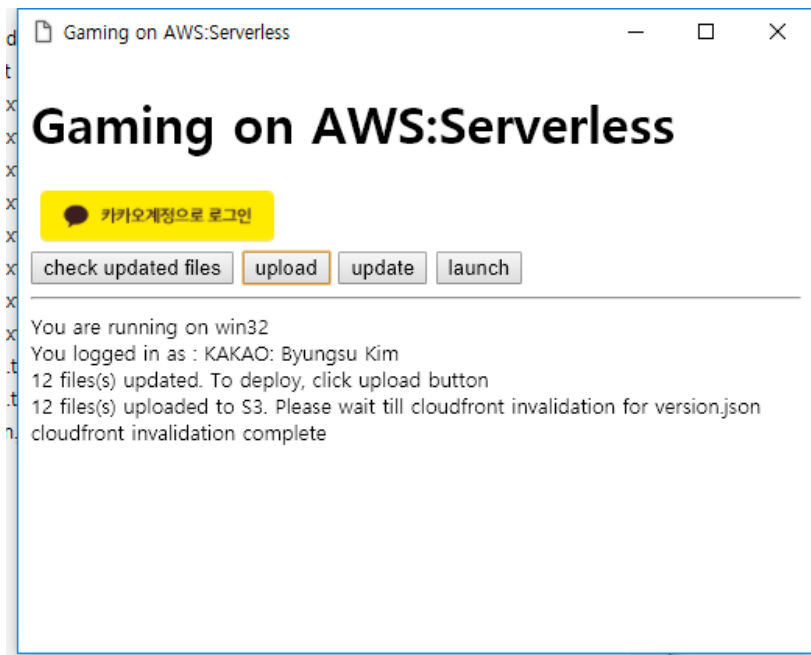


구성		새로 만들기
workspaces > gamingonaws > nw_app > patch		
이름	수정	
test	2017	
test.txt	2017	
test2.txt	2017	
test3.txt	2017	
test4.txt	2017	
test5.txt	2017	
test6.txt	2017	
test7.txt	2017	
test8.txt	2017	
test9.txt	2017	
test10.txt	2017	
test11.txt	2017	
version.json	2017	

4. 체크가 완료되면 upload 버튼을 눌러 s3 에 업데이트된 파일들을 업로드 합니다. 업로드가 완료되면 Patch 폴더에 filedb.db 파일이 업데이트 된것을 확인합니다. 이 filedb.db 에는 s3 에 object 를 업로드하면서 확인받은 s3 version_id 가 포함되어 있습니다.

이름	수용
test	201
filedb.db	201
test.txt	201
test2.txt	201
test3.txt	201
test4.txt	201
test5.txt	201
test6.txt	201
test7.txt	201
test8.txt	201
test9.txt	201
test10.txt	201
test11.txt	201
version.json	201

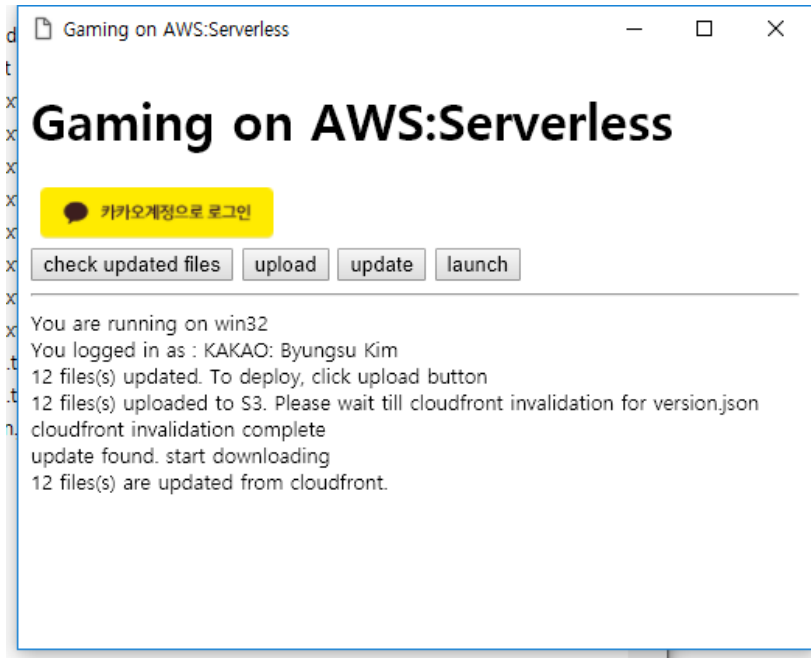
5. 애플리케이션이 자동으로 cloudfront distribution 에 version.json 파일을 invalidate 합니다. 이 과정은 10 초에서 1 분 정도 걸릴 수 있습니다. Invalidation 이 완료되면 application 에 완료 메시지가 출력됩니다.



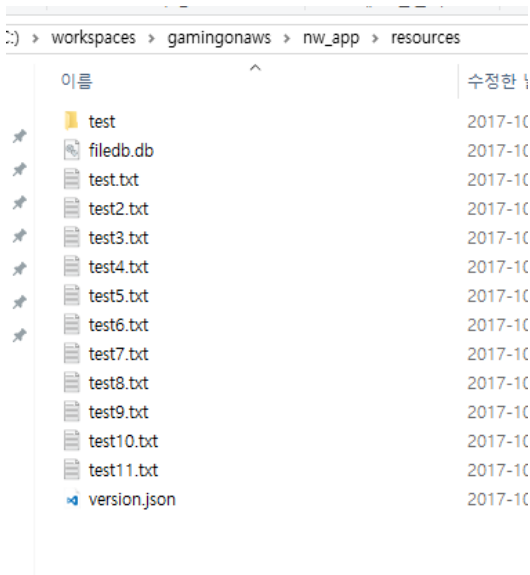
6. Update 버튼을 클릭하여 version 을 체크하고 업데이트된 파일을 다운받습니다.

Gaming on AWS

Serverless 로 게임 서비스 구현하기



7. nw_app 의 resources folder 에서 파일이 업데이트된 것을 확인합니다.



8. Patch 폴더에 파일을 추가하거나 파일 내용을 변경하여 업데이트된 파일만 패치가 이루어지는지 확인합니다.

Lab3. DynamoDB 와 stream 을 이용한 플레이어데이터 저장 및 통계

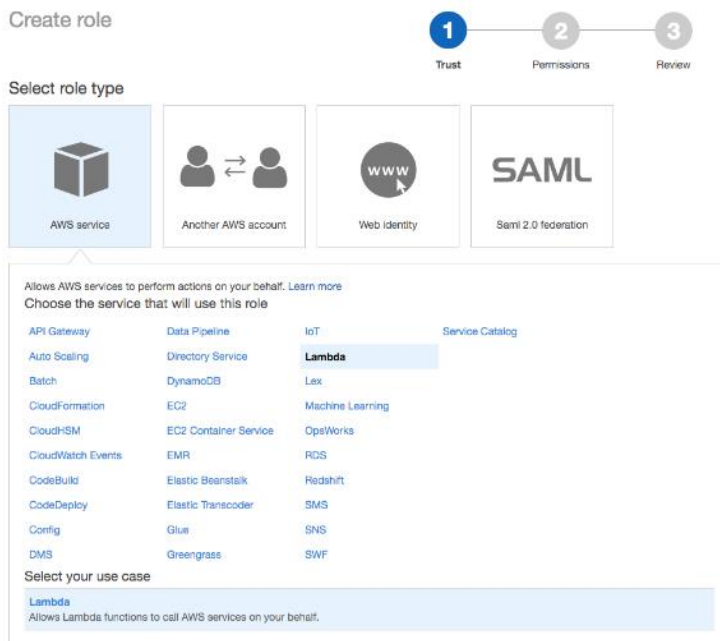
이제 사용자는 게임 클라이언트 업데이트를 마치고 게임을 시작할 수 있게 되었습니다. 사용자의 플레이어데이터를 기록하고 DynamoDB Stream 을 통해 score 통계를 내는 기능을 구현할 차례입니다. Lambda function 과 DynamoDB, API Gateway 를 사용하여 기능을 구현해보겠습니다.

IAM role

우선 DynamoDB 와 Lambda 함수를 만들기 전에 Lambda 함수에 적절한 권한을 부여합니다. 앞서 Lambda 함수를 만들 때에는 Lambda 함수를 생성하면서 IAM Role 도 생성하였지만, 이미 생성한 role 을 사용할 수도 있습니다. 여기서는 IAM role 부터 생성해보도록 합니다.

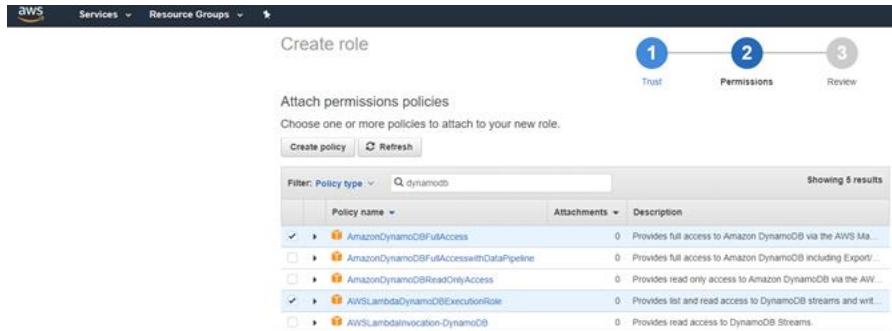
게임 스코어를 Put, Get 할 수 있는 2 가지 IAM role 을 각각 생성합니다.

1. AWS Management Console 에 로그인 한 뒤 IAM 서비스에 접속합니다.
2. 왼쪽 메뉴에서 Roles 를 선택합니다.
3. Create role 버튼을 클릭하여 역할 추가 페이지로 들어갑니다.

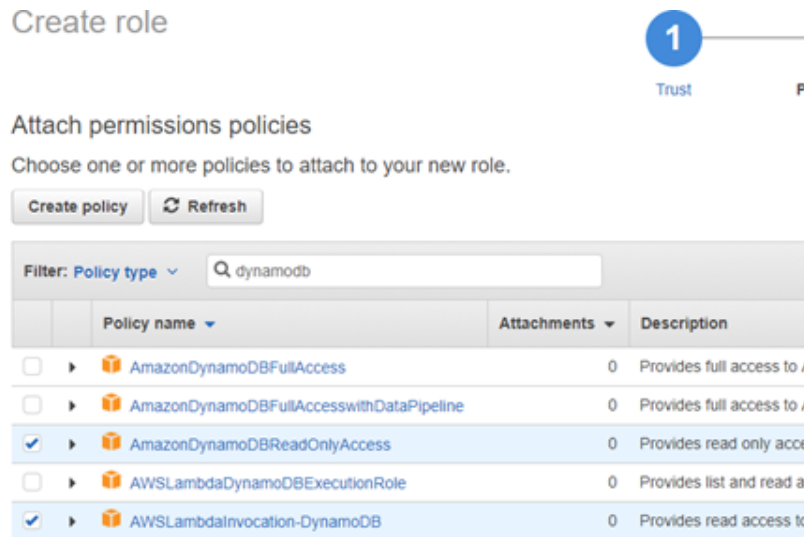


4. Lambda 서비스를 선택하고, **Next: Permissions** 버튼을 클릭합니다.

5. **AmazonDynamoDBFullAccess, AWSLambdaDynamoDBExecutionRole** 2 가지 역할을 선택합니다. **Next: Review** 를 클릭하여 진행합니다



6. **Role name** 에는 gamescore-update-role 을 입력한 뒤 **Create role** 을 클릭하여 완료합니다.
7. 위와 같은 방식으로 gamescore-read-role 을 생성합니다. 역할은 **AmazonDynamoDBReadOnlyAccess, AWSLambdaInvocation-DynamoDB** 를 선택합니다.



DynamoDB

게임 스코어를 기록하고 히스토리를 확인할 2 개의 테이블을 생성합니다.

1. AWS Management Console 에서 DynamoDB 서비스에 접속합니다.

- 우선 게임 스코어를 기록할 테이블을 생성합니다. Create table 을 선택하고 Table name 은 DemoUserScore, Primary key 에는 username 을 입력하고 Create 를 클릭합니다.

Create DynamoDB table Tutorial ?

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* DemoUserScore ⓘ

Primary key* Partition key

username String ⓘ

☐ Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☒ Use default settings

- No secondary indexes.
- Auto Scaling capacity set to 70% target utilization, at minimum capacity of 5 reads and 5 writes [NEW](#)

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel Create

- Table 생성이 완료되면 Manage Stream 을 활성화합니다. Manage Stream 을 선택한 뒤 View type 은 New and old images 를 선택하고 Enable 을 클릭합니다. 다음과 같이 Stream 이 활성화됩니다.

Stream details

Stream enabled Yes

View type New and old images

Latest stream ARN arn:aws:dynamodb:ap-northeast-2:584778767920:table/DemoUserScore/stream/2017-09-27T05:03:38.242

Manage Stream

- 샘플데이터를 생성해줍니다. Items 에서 Create item 을 클릭합니다.

Create item

Filter by table name X

Name DemoUserScore

Scan: [Table] DemoUserScore: username

Add filter

Start search

- + 버튼을 클릭한 뒤 Append 를 선택하고 Number 를 선택합니다. 다음과 같이 입력한 뒤 Save 합니다.

Create item

Tree

Item {2}

- username String : DefaultUser
- score Number : 123

6. 동일한 방법으로 게임 스코어 히스토리를 기록할 두 번째 테이블을 생성합니다. Table name 은 DemoScoreHistory 로 지정하고 Primary key 는 score 를 Number 타입으로 생성합니다. Stream 을 활성화 할 필요는 없습니다.

Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* DemoScoreHistory

Primary key* Partition key

score Number

☐ Add sort key

7. 다음의 정보로 Item 을 생성해줍니다.

Create item

Tree

Item (2)

- score Number : 123
- totalcount Number : 1

Lambda 함수 설정하기

1. AWS Management Console 에서 Lambda 서비스에 접속합니다.
2. 총 4 개의 Lambda function 을 만들어야 합니다.
3. Create function 을 선택한 뒤 Author from scratch 버튼을 클릭합니다.

Lambda > Functions > Create function

Step 1: Select blueprint

Step 2: Configure triggers

Step 3: Configure function

Step 4: Review

Select blueprint

Blueprints are sample configurations of event sources and Lambda functions. Choose a blueprint that best aligns with your desired scenario and customize as needed, or click on **Author from scratch** if you want to author a Lambda function and configure an event source separately. Except where otherwise noted, blueprints are licensed under [CC0](#).

Welcome to AWS Lambda! You can get started on creating your first Lambda function by choosing one of the blueprints below.

Blueprints

Export Author from scratch

Filter by tags and attributes or search by keyword

4. Trigger 는 현재 단계에서는 추가하지 않습니다. 바로 Next 를 클릭하여 다음 단계로 진행합니다
5. 첫 번째로 PutUserScore 를 생성합니다. 이 함수는 DynamoDB 에 게임 스코어를 업데이트 합니다. Role 은 이전 단계에서 생성한 gamescore-update-role 을 사용합니다.

Name	PutUserScore
Runtime	Node.js 6.10
Role	gamescore-update-role

Timeout	1 min
<pre> var AWS = require('aws-sdk'); var DOC = require('dynamodb-doc'); var dynamo = new DOC.DynamoDB(); exports.handler = function(event, context) { console.log(JSON.stringify(event.username)); var cb = function(err, data) { if(err) { console.log(err); context.fail('unable to update users at this time'); } else { console.log(data); context.done(null, data); } }; var item = { username: event.username, score: event.score }; dynamo.putItem({TableName:"DemoUserScore", Item:item}, cb); }; </pre>	

6. 두 번째로 GetUserScore 를 생성합니다. 이 함수는 DynamoDB 의 게임스코어 데이터를 읽어옵니다. Role 은 gamescore-read-role 을 사용합니다.

Name	GetUserScore
Runtime	Node.js 6.10
Role	gamescore-read-role
Timeout	1 min
<pre> var AWS = require('aws-sdk'); var DOC = require('dynamodb-doc'); var dynamo = new DOC.DynamoDB(); exports.handler = function(event, context) { var cb = function(err, data) { if(err) { console.log('error on GetDemoSAInfo: ',err); </pre>	

```

        context.done('Unable to retrieve information', null);
    } else {
        if(data.Items) {
            data.Items.forEach(function(elem) {
                console.log( elem.username + ": ",elem.score);
            });
            data.Items.sort(function(a,b) { return parseInt(b.score) -
parseInt(a.score) } );
            context.done(null, data.Items);
        } else {
            context.done(null, {});
        }
    }
};

dynamo.scan({TableName:"DemoUserScore"}, cb);
};

```

7. 세 번째는 GetScoreHistory 입니다. 이 함수는 지금까지 플레이했던 게임 스코어 기록을 가져옵니다.

Name	GetScoreHistory
Runtime	Node.js 6.10
Role	gamescore-read-role
Timeout	1 min

```

var AWS = require('aws-sdk');
var DOC = require('dynamodb-doc');
var dynamo = new DOC.DynamoDB();

exports.handler = function(event, context) {
    var cb = function(err, data) {
        if(err) {
            console.log('error on DemoScoreHistory: ',err);
            context.done('Unable to retrieve information', null);
        } else {
            if(data.Items) {
                context.done(null, data.Items);
            } else {
                context.done(null, {});
            }
        }
    }
};

```

```
dynamo.scan({TableName:"DemoScoreHistory"}, cb);



};
```

8. 마지막입니다. 게임 스코어를 히스토리에 기록할 CatchScoreUpdate 를 생성합니다. **Runtime** 은 Python 3.6 을 사용하고 Trigger 는 앞서 생성했던 DynamoDB 를 활용합니다.
9. DynamoDB 를 **Trigger** 로 설정합니다. **DynamoDB table** 은 **DemoUserScore** 를 선택하고 **Starting position** 은 **Trim horizon** 을 선택합니다. **Enable trigger** 에 체크한 뒤 Next 를 클릭합니다.

Configure triggers

You can choose to add a trigger that will invoke your function.

Add trigger
Remove



DynamoDB
Lambda

DynamoDB table
Please select a DynamoDB table. The Lambda function will be invoked whenever this table is updated.

DemoUserScore

Batch size
The largest number of records that AWS Lambda will retrieve from your table at the time of invoking your function. Your function receives an event with all the retrieved records.

100

Starting position
The position in the stream where AWS Lambda should start reading. For more information, see [ShardIteratorType](#) in the Amazon Kinesis API Reference.

Trim horizon

In order to read from the DynamoDB trigger, your execution role must have proper permissions.

Enable trigger
Enable the trigger now, or create it in a disabled state for testing (recommended).

☒

Cancel
Previous
Next

10. 다음의 정보를 입력하여 생성합니다. 이번에는 Python 으로 작업해봅니다.

Name	CatchScoreUpdate
Runtime	Python 3.6
Role	gamescore-update-role
Timeout	1 min
<pre>from __future__ import print_function import boto3</pre>	

```

import json
import decimal
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError

dynamodb = boto3.resource("dynamodb")
table = dynamodb.Table('DemoScoreHistory')

def increment_score(score_key):
    response = table.update_item(
        Key={'score': score_key },
        UpdateExpression="set totalcount = totalcount + :val",
        ExpressionAttributeValues={':val': decimal.Decimal(1) },
        ReturnValues="UPDATED_NEW"
    )
    return response

def lambda_handler(event, context):
    for record in event['Records']:
        print(record)
        newScore = int(record['dynamodb']['NewImage']['score']['N'])
        try:
            table.put_item(
                Item={ 'score': newScore, 'totalcount': 1 },
                ConditionExpression=Attr("score").ne(newScore)
            )
        except ClientError as e:
            if e.response['Error']['Code'] == "ConditionalCheckFailedException":
                print(e.response['Error']['Message'])
                increment_score(newScore)
            else:
                raise
        else:
            print("PutItem succeeded:")
    return 'Successfully processed {} records.'.format(len(event['Records']))

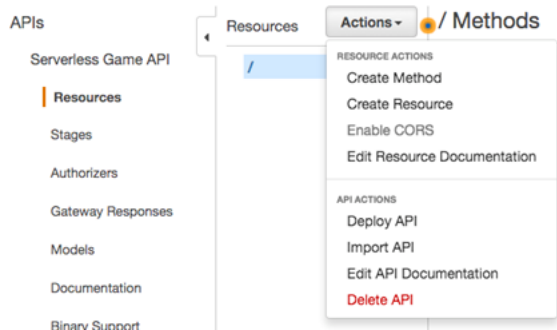
```

API Gateway

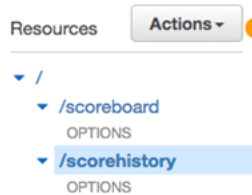
이제 생성한 lambda 함수들을 API Gateway 에 추가합니다.

1. AWS Management Console 에서 **API Gateway** 서비스에 접속합니다.

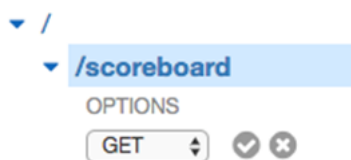
- 실습에는 scoreboard 와 scorehistory 두 가지 리소스가 더 필요합니다. Lab.1 에서 생성했던 API 에서 Actions 드롭다운메뉴에서 **Create Resource** 메뉴를 선택합니다.



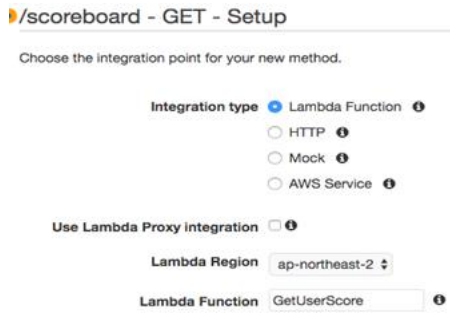
- Resource Name 에 scoreboard 를 입력하고 **Enable API Gateway CORS** 를 체크한 뒤 **Create Resource** 를 진행합니다.
- 동일한 방법으로 scorehistory 도 생성합니다. 이 때 Resource Path 에 주의합니다. 다음과 같이 두 개의 리소스를 생성한 뒤 다음으로 진행합니다.



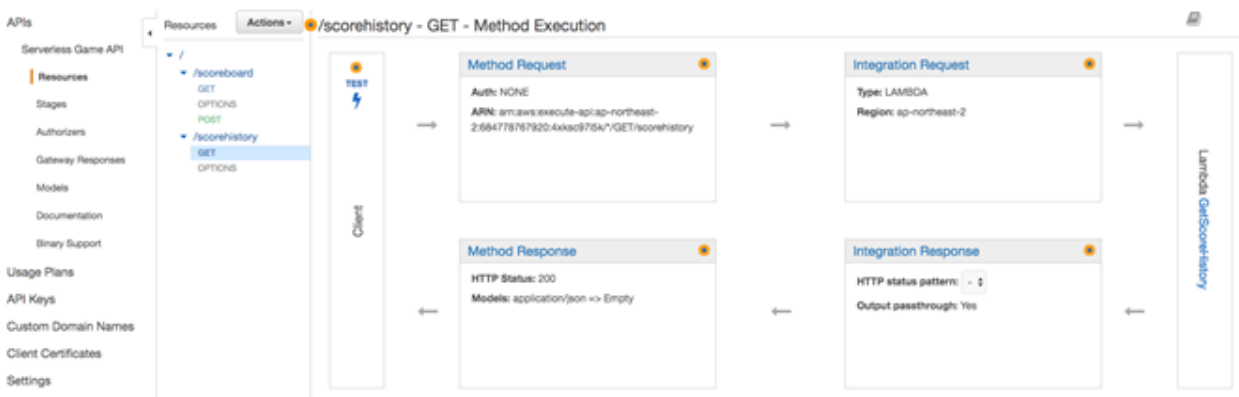
- Method 생성을 진행합니다. /scoreboard 리소스를 선택하고 Actions 의 Create Method 를 선택합니다.
- GET 을 선택한 뒤 체크 버튼을 클릭합니다.



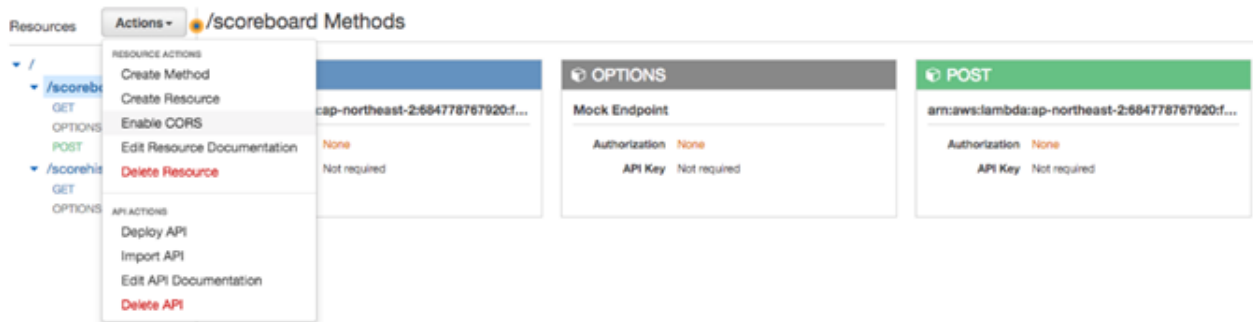
- Lambda Region** 은 Lambda 를 생성한 리전을 선택하고 **Lambda Function** 은 앞서 생성한 GetUserScore 를 입력합니다. **Save** 를 클릭합니다.



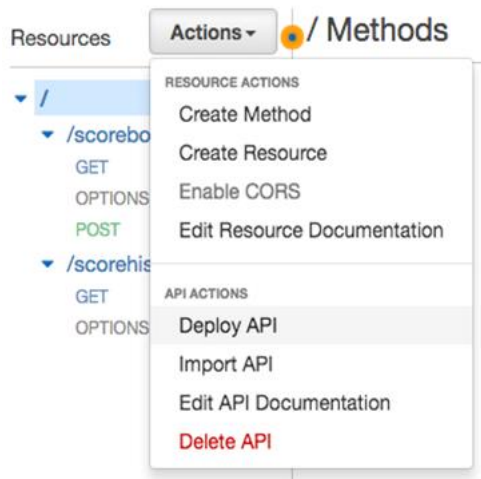
8. 위와 같은 방식으로 `/scoreboard`의 **POST**에는 `PutUserScore` 함수를 `/scorehistory`의 **GET**에는 `GetScoreHistory`를 설정해줍니다.



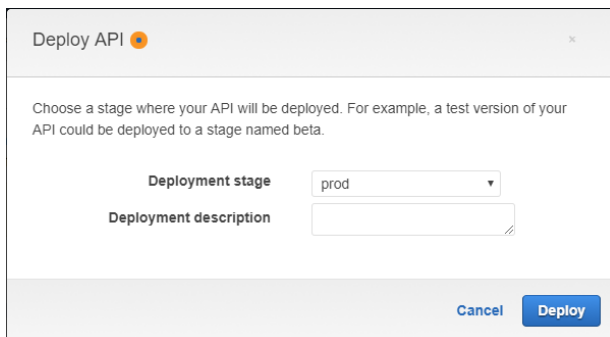
9. scoreboard 리소스를 선택한 뒤 Actions의 Enable CORS 메뉴를 선택합니다. 옵션은 변경하지 않고 **Enable CORS and replace existing CORS headers** 버튼을 클릭하고 **Yes, replace existing values** 버튼을 차례로 클릭합니다. 동일한 방법으로 scorehistory 리소스도 진행합니다.



10. 생성한 API를 배포해줍니다. 상위 경로 /를 선택한 뒤 Actions의 Deploy API 메뉴를 클릭합니다.

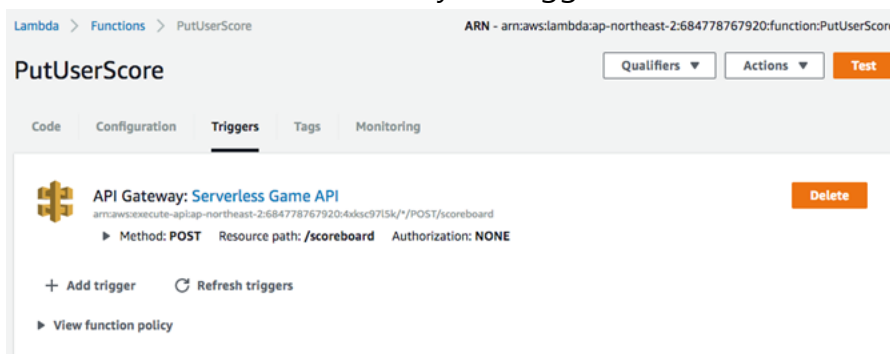


11. Deployment stage 는 Lab1 에서 사용하였던 prod 를 선택한 뒤 Deploy 버튼을 클릭합니다.



12. 생성된 Invoke URL 은 뒤의 게임 client 설정에 필요합니다.

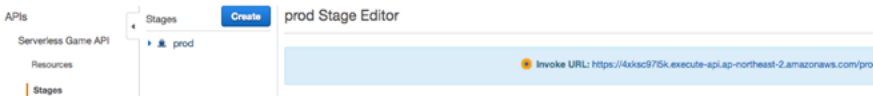
13. 진행을 완료하면 Lambda 에서 앞서 Trigger 를 설정하지 않은 PutUserScore, GetUserScore, GetScoreHistory 에 Trigger 가 추가된 것을 확인할 수 있습니다.



애플리케이션 설정 및 테스트

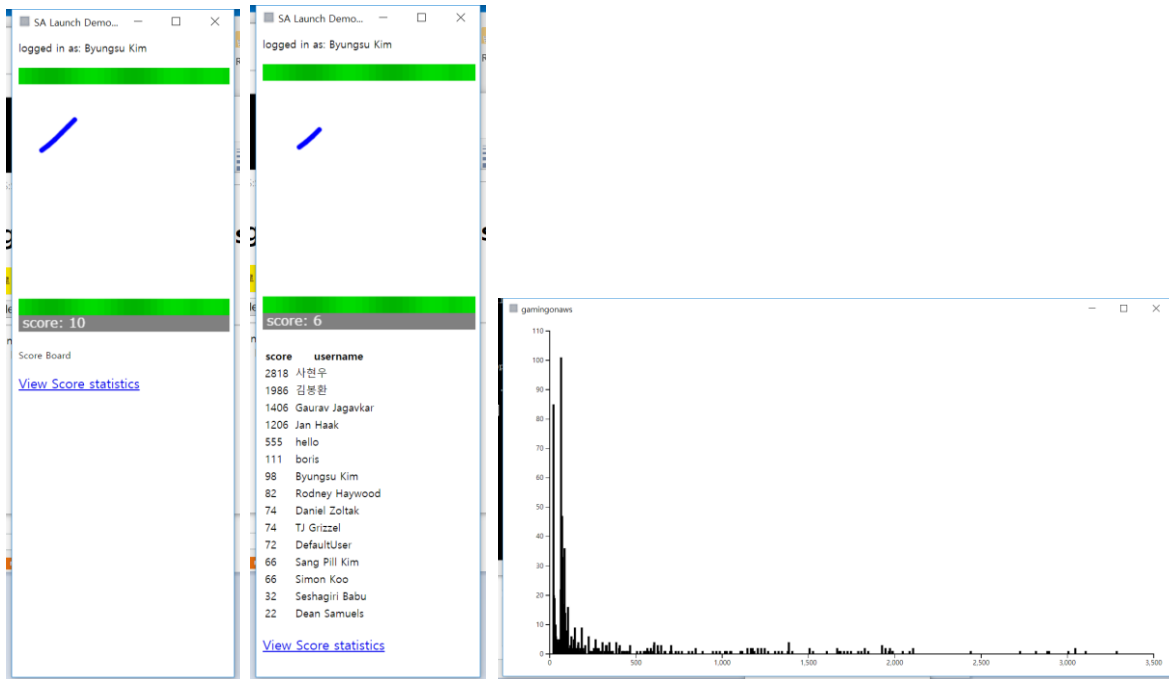
이제 위 단계에서 설정한 API Gateway API 들을 게임에서 사용하도록 설정하고 게임을 테스트해 볼 차례입니다.

1. nw_app 의 script 폴더의 config.json 파일에 scorehistory, scoreboard API 의 Endpoint URL 을 설정합니다. (API resource path 까지 포함하여 설정하여야 합니다. 혹은 Lambda 함수의 Trigger 항목에서 method 를 클릭하면 나오는 Invoke URL 을 사용하셔도 됩니다.)



```
{
  "bucketname": "<s3 bucket name>",
  "uploadprefix": "update",
  "region": "ap-northeast-2",
  "api_url": "<api_gateway prod endpoint url(+method name if specified)>",
  "IdentityPoolId": "<cognito Identity Pool ID >",
  "cloudfrontDistributionId": "<cloudfrontDistributionId(not domain name, randomized string id)>",
  "cloudfrontURL": "http://<cloudfrontdomainname>",
  "scorehistory_api_url": "<invoke_URL>/scorehisto",
  "score_api_url": "<invoke_URL> /scoreboard"
}
```

2. nw_app 을 실행하고, Kakaotalk 에 로그인한 후 launch 버튼을 누르면 게임이 시작됩니다.
3. 마우스 클릭이나 space 바를 눌러 게임을 진행할 수 있습니다. 게임이 진행된 후 score board 가 출력되는지, View Score statistics 링크를 클릭하여 score history 가 정상적으로 출력되는지 확인합니다.



리소스 정리하기

수고하셨습니다! 이제 서버리스로 게임 서비스 구현하기 Hands on Lab 이 모두 완료되었습니다. 오늘 사용한 모든 서비스들은 Free Tier 로 계정 생성후 1 년간은 비용이 발생하지 않지만, 1 년 후에는 작은 금액이나마 과금이 일어날 수 있으므로 리소스를 정리하여야 합니다. 예상하지 않은 비용이 발생하지 않도록 오늘 실습에서 사용한 다음 리소스들을 삭제하시기 바랍니다.

1. S3 Bucket 삭제
2. Cloudfront Distributions 삭제
3. API Gateway API 삭제
4. Lambda Function 삭제
5. DyanmoDB Table 삭제 (Stream 은 Table 을 삭제하면 함께 삭제됩니다.)