

배포계획서

배포 계획

1. 목적

스프링부트를 기반의 쿠버네티스 기반의 마이크로서비스를 구축하고 사용하여 배포하려고 하고 있으며, 기본적인 gis개발을 해서 주변의 와이파이 지역을 탐색하려고 한다. 혹시나 부족한 부분은 rest api나 api를 사용하여 보충하려고 하고 있음

2. 세부추진 계획

- 쿠비네티스(마이크로서비스)를 사용하여 구축하기!!
- 필요한 개발 서비스들을 설치하기
- 툴은 이클립스를 사용할 것
- 스프링 부트를 통해 개발하고 쿠비네티스로 구축한 곳에다 배포하기

2.1. 세부 추진 일정

- 2020/11/07 ~ 2020/11/15 : 쿠비네티스(마이크로서비스) 구축 및
필요한 개발 서비스들을 설치하기
- 2020/11/15 ~ 2020/11/21 : 스프링부트로 gis쪽을 웹페이지 화면에 나오도록
개발할 계획
- 2020/11/21 ~ 2020/11/22 : 스프링부트를 사용한 개발한 소스를 쿠비네티스로
구축한 곳에 배포하기(대략적인 계획)

배포 계획

- Restful api 배포

호스팅 REST API 를 사용하여 사이트에 배포

[Firebase 호스팅 REST API](#) 를 사용하면 **Firebase** 호스팅 사이트에 프로그래매틱 방식으로 맞춤설정 가능한 배포를 할 수 있습니다. 이 **REST API** 를 사용하여 새로운 또는 업데이트된 호스팅 콘텐츠 및 구성을 배포합니다.

배포를 위해 [Firebase CLI](#) 대신 **Firebase** 호스팅 **REST API** 를 사용하여 프로그래매틱 방식으로 사이트 애셋의 새 `version` 을 만들어서 파일을 버전에 업로드한 다음 이 버전을 사이트에 배포할 수 있습니다.

다음은 **Firebase** 호스팅 **REST API** 로 할 수 있는 작업의 몇 가지 예입니다.

- **배포 예약.** 크론 작업과 함께 **REST API** 를 사용하면 정기적인 일정으로 **Firebase** 호스팅 콘텐츠를 변경할 수 있습니다(예: 콘텐츠의 특별한 명절 또는 이벤트 관련 버전 배포).
- **개발자 도구와 통합.** 도구에서 단 한 번의 클릭만으로 웹 앱 프로젝트를 **Firebase** 호스팅에 배포하는 옵션을 만들 수 있습니다(예: **IDE** 에서 배포 버튼을 클릭).
- **정적 콘텐츠 생성 시 자동 배포.** 프로세스가 프로그래매틱 방식으로 정적 콘텐츠를 생성하면(예: 위키 또는 뉴스 기사와 같은 사용자 제작 콘텐츠) 생성된 콘텐츠를 동적으로 제공하지 않고 정적 파일로 배포할 수 있습니다. 이렇게 하면 비싼 컴퓨터 성능을 절약하고 보다 확장 가능한 방식으로 파일을 제공할 수 있습니다.

이 가이드에서는 먼저 **API** 를 사용 설정하고 인증하고 승인하는 방법을 설명합니다. 그 다음, **Firebase** 호스팅 버전을 만들고 필요한 파일을 버전에 업로드한 후 마지막으로 이 버전을 배포하는 예시가 제시되어 있습니다.

[전체 호스팅 REST API 참조 문서](#)에서도 이 **REST API** 에 대해 자세히 알아볼 수 있습니다.

시작하기 전: REST API 사용 설정

Google API 콘솔에서 **Firebase** 호스팅 **REST API** 를 사용 설정해야 합니다.

1. Google API 콘솔에서 [Firebase 호스팅 API 페이지](#)를 엽니다.
2. 메시지가 나타나면 **Firebase** 프로젝트를 선택합니다.

참고: 모든 **Firebase** 프로젝트는 **Google API** 콘솔에 해당 프로젝트가 있습니다.

3. **Firebase 호스팅 API** 페이지에서 **사용 설정**을 클릭합니다.

1 단계: API 요청을 인증하고 승인하는 액세스 토큰 가져오기

Firebase 프로젝트는 앱 서버 또는 신뢰할 수 있는 환경에서 **Firebase** 서버 API를 호출하는 데 사용할 수 있는 **Google 서비스 계정**을 지원합니다. 로컬에서 코드를 개발하거나 온프레미스에 애플리케이션을 배포하는 경우 이 서비스 계정을 통해 가져온 사용자 인증 정보를 사용하여 서버 요청을 승인할 수 있습니다.

서비스 계정을 인증하고 **Firebase** 서비스에 액세스하도록 승인하려면 **JSON** 형식의 비공개 키 파일을 생성해야 합니다.

서비스 계정의 비공개 키 파일을 생성하려면 다음 안내를 따르세요.

1. **Firebase Console**에서 **설정 > 서비스 계정**을 엽니다.
2. 새 비공개 키 생성을 클릭한 다음 키 생성을 클릭하여 확인합니다.
3. 키가 들어 있는 **JSON** 파일을 안전하게 저장합니다.

원하는 언어의 [Google API 클라이언트 라이브러리](#)와 함께 **Firebase** 사용자 인증 정보를 사용하여 수명이 짧은 **OAuth 2.0** 액세스 토큰을 가져옵니다.

[node.js](#)[Python](#) [자바](#)

```
const {google} = require('googleapis');
function getAccessToken() {
  return new Promise(function(resolve, reject) {
    var key = require('./service-account.json');
    var jwtClient = new google.auth.JWT(
      key.client_email,
      null,
      key.private_key,
      SCOPES,
      null
    );
    jwtClient.authorize(function(err, tokens) {
      if (err) {
        reject(err);
      }
      return;
    });
  });
}
```

```

    }
    resolve(tokens.access_token);
  });
});
}

```

이 예시에서는 **Google API** 클라이언트 라이브러리가 **JSON** 웹 토큰, 즉 **JWT** 를 사용해 요청을 인증합니다. 자세한 내용은 [JSON 웹 토큰](#)을 참조하세요.

액세스 토큰이 만료되면 토큰 새로고침 메서드가 자동으로 호출되어 업데이트된 액세스 토큰이 발급됩니다.

참고: 서버 환경의 *자동화된* 태스크에는 서비스 계정을 사용하는 것이 적합하지만 다른 방법으로 승인을 받아 **Firebase** 호스팅 **REST API** 를 사용할 수도 있습니다. 예를 들어 프로젝트 구성원이 (1) 프로젝트의 관리자 또는 편집자 역할을 할당받고 (2) 적절한 범위의 액세스 토큰을 제공하는 경우 **API** 를 사용할 수 있습니다. **Firebase** 호스팅 **REST API** 의 사용 승인을 얻을 수 있는 다른 **OAuth** 흐름에 대해서는 [기타 시나리오](#)를 검토하세요.

2 단계: 사이트의 새 버전 만들기

첫 번째 **API** 호출은 사이트의 새 **Version**을 만들기 위한 것입니다. 이 가이드 뒷부분에서 파일을 이 버전에 업로드한 다음 사이트에 배포하게 됩니다.

1. 배포할 사이트의 **SITE_NAME**을 결정합니다.

참고: 기본 호스팅 사이트의 경우 **SITE_NAME**은 **Firebase** 프로젝트 ID(**Firebase** 하위 도메인 **SITE_NAME.web.app** 및 **SITE_NAME.firebaseio.com**을 만드는 데 사용)입니다.

Firebase 프로젝트에서 [여러 사이트](#)를 만든 경우 배포할 사이트의 **SITE_NAME**을 사용하고 있는지 확인합니다.

2. 호출 시 **SITE_NAME**을 사용하여 [versions.create](#) 엔드포인트를 호출합니다.

(선택사항) 호출 시 지정된 기간 동안 모든 파일을 캐싱하는 헤더 설정을 포함하여 [Firebase 호스팅 구성 객체](#)를 전달할 수도 있습니다.

예를 들면 다음과 같습니다.

[cURL 명령어](#)원시 **HTTP** 요청

```

curl -H "Content-Type: application/json" \
  -H "Authorization: Bearer ACCESS_TOKEN" \
  -d '{
    "config": {

```

```

    "headers": [{
      "glob": "**",
      "headers": {
        "Cache-Control": "max-age=1800"
      }
    }]
  }
}' \

```

`https://firebasehosting.googleapis.com/v1beta1/sites/SITE_NAME/versions`

`versions.create`에 대해 이 API를 호출하면 다음 JSON을 반환합니다.

```

{
  "name": "sites/SITE_NAME/versions/VERSION_ID",
  "status": "CREATED",
  "config": {
    "headers": [{
      "glob": "**",
      "headers": {
        "Cache-Control": "max-age=1800"
      }
    }]
  }
}

```

응답에는 `sites/SITE_NAME/versions/VERSION_ID` 형식으로 새 버전에 대한 고유 식별자가 포함됩니다. 이 버전을 참조하려면 가이드 전체에서 제공된 고유 식별자를 사용해야 합니다.

3 단계: 배포하려는 파일 목록 지정

이제 새로운 버전 식별자를 확보했으므로 이 새 버전에서 최종 배포하려는 파일을 **Firebase** 호스팅에 알려야 합니다.

중요: 이 호출에서 새 파일과 이전 버전의 기존 파일(수정 및 수정되지 않은 파일 모두)을 포함하여 해당 버전의 *모든* 파일을 나열해야 합니다.

이 API를 사용하려면 **SHA256** 해시로 파일을 식별해야 합니다. 따라서 API를 호출하기 전에 먼저 파일을 **Gzip**으로 압축한 후 새로 압축한 파일의 **SHA256** 해시를 가져와 각 정적 파일마다 해시를 계산합니다.

예를 들어 새 버전으로 `file1`, `file2`, `file3` 3 개의 파일을 배포하려고 한다고 가정해 봅시다.

1. 파일을 Gzip 으로 압축합니다.

```
gzip file1 && gzip file2 && gzip file3
```

이제 3 개의 압축 파일 `file1.gz`, `file2.gz`, `file3.gz` 가 생겼습니다.

2. 각 압축 파일의 SHA256 해시를 가져옵니다.

```
3. cat file1.gz | openssl dgst -sha256
4.
5. 66d61f86bb684d0e35f94461c1f9cf4f07a4bb3407bfbd80e518bd44368ff8f4
6. cat file2.gz | openssl dgst -sha256
7.
8. 490423ebae5dcd6c2df695aea79f1f80555c62e535a2808c8115a6714863d083
9. cat file3.gz | openssl dgst -sha256
10.
11. 59cae17473d7dd339fe714f4c6c514ab4470757a4fe616dfdb4d81400addf315
```

이제 3 개의 압축 파일의 SHA256 해시 3 개가 생겼습니다.

12. API 요청에서 이 3 가지 해시를 `versions.populateFiles` 엔드포인트로 보냅니다.

업로드된 파일에 대해 원하는 경로별로 각 해시를 나열합니다(이 예시에서는 `/file1`, `/file2`, `/file3` 파일이 사용됨).

참고: API 요청에서 최대 1,000 개의 파일 해시를 보낼 수 있지만 이 엔드포인트는 여러 번 호출할 수 있습니다. 각 호출 시 파일이 버전에 추가됩니다.

예를 들면 다음과 같습니다.

[cURL 명령어](#)원시 HTTP 요청

```
$ curl -H "Content-Type: application/json" \
  -H "Authorization: Bearer ACCESS_TOKEN" \
  -d '{
    "files": {
      "/file1":
"66d61f86bb684d0e35f94461c1f9cf4f07a4bb3407bfbd80e518bd44368ff8f4",
      "/file2":
"490423ebae5dcd6c2df695aea79f1f80555c62e535a2808c8115a6714863d083",
      "/file3":
"59cae17473d7dd339fe714f4c6c514ab4470757a4fe616dfdb4d81400addf315"
    }
  }' \
https://firebasehosting.googleapis.com/v1beta1/sites/SITE_NAME/versions/VERSION_ID:populateFiles
```

`versions.populateFiles`에 대해 이 API를 호출하면 다음 JSON을 반환합니다.

```
{
  "uploadRequiredHashes": [
    "490423ebae5dcd6c2df695aea79f1f80555c62e535a2808c8115a6714863d083",
    "59cae17473d7dd339fe714f4c6c514ab4470757a4fe616dfdb4d81400addf315"
  ],
  "uploadUrl": "https://upload-
firebasehosting.googleapis.com/upload/sites/SITE_NAME/versions/VERSION_
ID/files"
}
```

이 응답에는 다음이 포함됩니다.

- 업로드해야 하는 각 파일의 해시입니다. 예를 들어 이 예시에서 `file1`은 이전 버전에서 이미 업로드되었으므로 해시는 `uploadRequiredHashes` 목록에 포함되지 않습니다.

참고: 새 버전의 일부 파일은 업로드할 필요가 없습니다(예: 파일이 이전 버전에 이미 있고 새 버전에 대해 수정되지 않은 경우).

- 새 버전에 해당하는 `uploadUrl`입니다.

다음 단계에서 새 파일을 2개 업로드하려면 `versions.populateFiles` 응답의 해시 및 `uploadURL`이 필요합니다.

4 단계: 필수 파일 업로드

각 필수 파일(이전

단계에서 `versions.populateFiles` 응답의 `uploadRequiredHashes`에 나열된 파일)은 개별적으로 업로드해야 합니다. 이러한 파일 업로드의 경우 파일 해시와 이전 단계의 `uploadUrl`이 필요합니다.

1. 슬래시와 파일의 해시를 `uploadUrl`에 추가하여 `https://upload-firebasehosting.googleapis.com/upload/sites/SITE_NAME/versions/VERSION_ID/files/FILE_HASH` 형식으로 파일별 URL을 만듭니다.
2. 일련의 요청을 사용하여 모든 필수 파일을 파일별 URL에 하나씩 업로드합니다(이 예시에서는 `file2.gz` 및 `file3.gz`만).

예를 들어 압축된 `file2.gz`를 업로드하는 방법은 다음과 같습니다.

[cURL 명령어](#) 원시 HTTP 요청

```
curl -H "Authorization: Bearer ACCESS_TOKEN" \
```



```
-H "Content-Type: application/octet-stream" \  
--data-binary @./file2.gz \  
https://upload-  
firebasehosting.googleapis.com/upload/sites/SITE_NAME/versions/VERSION_  
ID/files/FILE_HASH
```

업로드가 성공하면 200 OK HTTP 응답을 반환합니다.

5 단계: 버전 상태를 FINALIZED 로 업데이트

versions.populateFiles 응답에 나열된 모든 파일을 업로드한 후 버전의 상태를 FINALIZED 로 업데이트할 수 있습니다.

FINALIZED 로 설정된 API 요청의 status 필드를 사용하여 versions.patch 엔드포인트를 호출합니다.

예를 들면 다음과 같습니다.

cURL 명령어원시 HTTP 요청

```
curl -H "Content-Type: application/json" \  
-H "Authorization: Bearer ACCESS_TOKEN" \  
-X PATCH \  
-d '{"status": "FINALIZED"}' \  
https://firebasehosting.googleapis.com/v1beta1/sites/SITE_NAME/versions/  
VERSION_ID?update_mask=status
```

versions.patch 에 대해 이 API 를 호출하면 다음 JSON 을 반환합니다. status 가 FINALIZED 로 업데이트되었는지 확인합니다.

```
{  
  "name": "sites/SITE_NAME/versions/VERSION_ID",  
  "status": "FINALIZED",  
  "config": {  
    "headers": [{  
      "glob": "**",  
      "headers": {"Cache-Control": "max-age=1800"}  
    }]  
  },  
  "createTime": "2018-12-02T13:41:56.905743Z",  
  "createUser": {  
    "email": "SERVICE_ACCOUNT_EMAIL@SITE_NAME.iam.gserviceaccount.com"  
  },  
  "finalizeTime": "2018-12-02T14:56:13.047423Z",  
  "finalizeUser": {
```

```

    "email": "USER_EMAIL@DOMAIN.tld"
  },
  "fileCount": "5",
  "versionBytes": "114951"
}

```

6 단계: 배포 버전 출시

이제 최종 버전이 준비되었으므로 배포용으로 출시합니다. 이 단계에서는 호스팅 구성과 새 버전의 모든 콘텐츠 파일이 포함된 버전의 [Release](#)를 만듭니다.

[releases.create](#) 엔드포인트를 호출하여 출시 버전을 만듭니다.

예를 들면 다음과 같습니다.

[cURL 명령어](#)원시 HTTP 요청

```

curl -H "Authorization: Bearer ACCESS_TOKEN" \
  -X POST
https://firebasehosting.googleapis.com/v1beta1/sites/SITE_NAME/releases
?versionName=sites/SITE_NAME/versions/VERSION_ID

```

[releases.create](#)에 대해 이 API를 호출하면 다음 JSON을 반환합니다.

```

{
  "name": "sites/SITE_NAME/releases/RELEASE_ID",
  "version": {
    "name": "sites/SITE_NAME/versions/VERSION_ID",
    "status": "FINALIZED",
    "config": {
      "headers": [{
        "glob": "**",
        "headers": {"Cache-Control": "max-age=1800"}
      }]
    }
  },
  "type": "DEPLOY",
  "releaseTime": "2018-12-02T15:14:37Z"
}

```

이제 호스팅 구성과 새 버전의 모든 파일을 사이트에 배포하고 다음 URL을 사용하여 파일에 액세스할 수 있습니다.

- `https://SITE_NAME.web.app/file1`
- `https://SITE_NAME.web.app/file2`
- `https://SITE_NAME.web.app/file3`

이러한 파일은 `SITE_NAME.firebaseapp.com` 도메인과 연결된 URL 을 통해서도 액세스할 수 있습니다.

- 쿠버네티스 api 배포

쿠버네티스 api를 배포하는 방법은 googlecloud플랫폼 api로 클러스터를 생성 후 cloud shell을 사용하는 방법입니다.

클러스터에 애플리케이션(마이크로서비스) 배포하기

클러스터가 생성되었으므로 이제 컨테이너화된 애플리케이션을 배포할 수 있습니다. 이번 실습에서는 `hello-app` 을 클러스터에서 실행합니다.

Kubernetes Engine에서는 Kubernetes 객체를 사용해 클러스터의 리소스를 만들고 관리합니다. Kubernetes에서는 웹 서버와 같은 스테이트리스(Stateless) 애플리케이션의 배포를 위한 배포 객체를 제공합니다. 서비스 객체에서는 인터넷에서 애플리케이션에 액세스하기 위한 규칙 및 부하 분산 방식을 정의합니다.

Cloud Shell에서 다음과 같은 `kubectl create` 명령어를 실행하여 `hello-app` 컨테이너 이미지에서 새로운 배포 `hello-server` 를 생성합니다.

1. kubectl 설치 (cloud shell에서는 설치 안해도 됨.)

```
$ sudo apt-get install kubectl
```

2. hello-server 배포

```
$ kubectl create deployment hello-server --image=gcr.io/google-samples/hello-app:1.0
```

이 Kubernetes 명령어를 사용하면 hello-server 을 나타내는 배포 객체가 생성됩니다. 이 명령어에서 --image 를 통해 배포할 컨테이너 이미지가 지정됩니다. 이 경우에는 명령어를 통해 Google Container Registry 버킷에서 예시 이미지를 가져옵니다. gcr.io/google-samples/hello-app:1.0 은 가져올 이미지 버전을 지정합니다. 버전이 지정되지 않은 경우 최신 버전이 사용됩니다.

이제 다음과 같은 kubectl expose 명령어를 실행하여 Kubernetes 서비스를 생성합니다. 이 서비스는 애플리케이션을 외부 트래픽에 노출할 수 있게 해주는 Kubernetes 리소스입니다.

```
$ kubectl expose deployment hello-server --type=LoadBalancer --port 9005
```

이 명령어에서는:

- --port 를 통해 컨테이너가 노출될 포트가 지정됩니다.
- type="LoadBalancer"를 전달하면 컨테이너의 Compute Engine 부하 분산기가 생성됩니다.

kubectl get 을 실행하여 hello-server 서비스를 검사합니다.

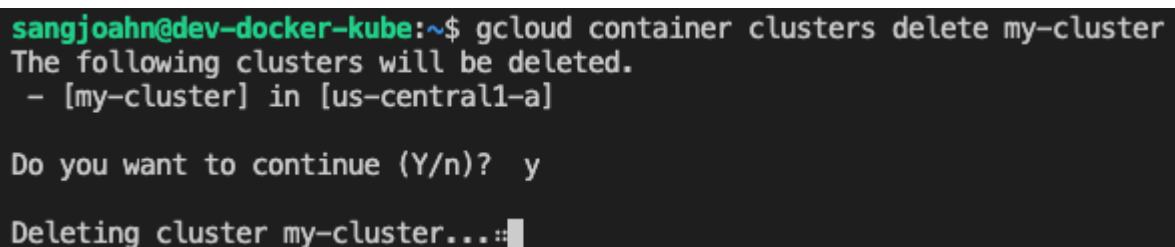
```
$ kubectl get service
```

참고: 외부 IP 주소가 생성되는 데는 몇 분이 걸릴 수 있습니다. EXTERNAL-IP 열이 아직 출력되지 않았으면 위 명령어를 다시 실행하세요.

* 외부 ip 주소 - http://{EXTERNAL-IP}:{port}

클러스터를 삭제하려면 다음과 같은 명령어를 실행합니다.

```
$ gcloud container clusters delete [CLUSTER-NAME]
```



```
sangjoahn@dev-docker-kube:~$ gcloud container clusters delete my-cluster
The following clusters will be deleted.
- [my-cluster] in [us-central1-a]

Do you want to continue (Y/n)? y

Deleting cluster my-cluster...:
```

* 메시지가 표시되면 Y 를 입력하여 확인합니다. 클러스터를 삭제하는 데는 몇 분이 걸릴 수 있습니다.