

# WEEK3: NodeJS ile AWS Üzerinden Uygulama Ayağa Kaldırma

-Uğurkan Hoşgör

## Description:

Bu hafta temel olarak REST API protokolleri ile nodejs kullanarak uygulama geliştireceğiz. Bu uygulamaları AWS ekosisteminde ayağa kaldırıp, güvenlik tedbirlerini koyacağız.

## Tasks:

1. Temel bir öğrenci ve öğretim görevlisi görüntüleme platformu için API geliştirin, bu platformda öğrenciler, öğretim görevlileri ve dersler bulunacak. ✓
2. Bu platformun testlerini, Postman uygulamasını kullanarak yapabilirsiniz. ✓
3. Bu API servisinizi, bir docker container'ı içerisinde çalıştırmanız bekleniyor. ✓
4. Bu kurduğunuz docker container'ı, AWS ekosistemi içerisinde tercih ettiğiniz metod ile ayağa kaldırıp tercih ettiğiniz kişiler tarafından erişilebilir kılın. ✓
- a. Port'lerinizi açma konusunda araştırma yapın(Security grouplardan yapıyorsunuz). ✓
5. Yaptığınız tüm çalışmalarda, endüstri standartı best practice'leri kullanmanız bekleniyor. Özellikle AWS'nin best practice'lerini, kendi dokümantasyonlarında uygun araçları yaparak bulabilirsiniz. ✓
6. NodeJS ve mongoose dışında herhangi bir teknoloji sınırlaması bulunmamakta. Tek şart AWS ekosisteminden çıkmamanız. ✓
7. Opsiyonel olarak, API'nize çeşitli güvenlik protokolleri ekleyebilirsiniz(JWT gibi), veya HTTPS bağlantı protokolüne bağlayabilirsiniz(SSL sertifikası için herhangi bir ücret ödemeyin(Domain gibi.)). ⚠
8. Elastic beanstalk adındaki servis üzerinden ayağa kaldırma konusunda araştırma yapmanız tavsiyemizdir. ✓

## What I have done:

I created a web server using Express and Node.js in a Docker container, then connected it to MongoDB, which is in another container using the Mongoose library. I was planning to include a login page and security features, but time was not enough, so I didn't include them. Then I implemented API features for database access and modification. At the end, I got everything running on an AWS EC2 machine using an SSH connection. It has a simple interface for inserting and viewing data. The website and API are accessible from the internet. There is no restriction for now.

## Details:

Website interface for viewing and posting:

**Student and Course Management**

**Students**

Name:

Age:

[Add Student](#)

Test, Age: 5

**Instructors**

Name:

Expertise:

[Add Instructor](#)

Inst, Expertise: Devops

**Courses**

Title:

Instructor:

I kept it simple in order to focus on the API features and make all work.

## EC2 Machine details:

EC2 > Instances > i-069486a2d7cca299d

**Instance summary for i-069486a2d7cca299d (Week3-server)** [Info](#)

Updated less than a minute ago

[Refresh](#) [Connect](#) [Instance state ▼](#) [Actions ▼](#)

Instance ID i-069486a2d7cca299d (Week3-server)	Public IPv4 address 16.171.20.48   <a href="#">open address</a>	Private IPv4 addresses 10.0.1.106
IPv6 address -	Instance state <b>Running</b>	Public IPv4 DNS ec2-16-171-20-48.eu-north-1.compute.amazonaws.com   <a href="#">open address</a>
Hostname type IP name: ip-10-0-1-106.eu-north-1.compute.internal	Private IP DNS name (IPv4 only) ip-10-0-1-106.eu-north-1.compute.internal	Elastic IP addresses -
Answer private resource DNS name -	Instance type t3.micro	AWS Compute Optimizer finding <a href="#">Opt-in to AWS Compute Optimizer for recommendations.</a> <a href="#">Learn more</a>
Auto-assigned IP address 16.171.20.48 [Public IP]	VPC ID vpc-0b12ef41b128b8cde (ECS default - VPC)	Auto Scaling Group name -
IAM Role -	Subnet ID subnet-0e0975ccad9bdeed7 (ECS default - Public Subnet 2)	

## Inbound rules:

### ▼ Inbound rules

<input type="text" value="Filter rules"/>				
Name	Security group rule ID	Port range	Protocol	Source
-	sgr-0f934fd2e0ca141be	27017	TCP	0.0.0.0/0
-	sgr-07256ede4283d8775	80	TCP	0.0.0.0/0
-	sgr-0a2906fab8a97aba3	443	TCP	0.0.0.0/0
-	sgr-04cf3bc0c4c342329	22	TCP	0.0.0.0/0
-	sgr-09bfce7bd769d12e7	80	TCP	::/0
-	sgr-04dd325191cb92732	8080	TCP	0.0.0.0/0
-	sgr-0f85acc153e0a45d6	443	TCP	::/0

I kept some of them opened for future use. Only ports 22, 8080 and 27017 in use for now.

Docker compose up result when it was ready to run on EC2.

```
Desktop — ubuntu@ip-10-0-1-106: ~/docker-mongo-app1 — ssh...
=> => extracting sha256:0d27a8e861329007574c6766fba946d48e20d2c8e 0.0s
=> [app internal] load build context 0.0s
=> => transferring context: 76.08kB 0.0s
=> [app 2/4] COPY package*.json ./ 0.2s
=> [app 3/4] RUN npm install 8.2s
=> [app 4/4] COPY . . 0.1s
=> [app] exporting to image 0.7s
=> => exporting layers 0.7s
=> => writing image sha256:7fa89b50ea39a76f795c160ae9d62cc167e1c2 0.0s
=> => naming to docker.io/library/docker-mongo-app1-app 0.0s
[+] Running 5/5
✔ Network docker-mongo-app1_default Created 0.1s
✔ Volume "docker-mongo-app1_app-data" Created 0.0s
✔ Volume "docker-mongo-app1_mongodb-data" Created 0.0s
✔ Container docker-mongo-app1-db-1 Started 0.1s
✔ Container docker-mongo-app1-app-1 Started 0.0s
ubuntu@ip-10-0-1-106:~/docker-mongo-app1$
```

Get students API result:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://ec2-16-171-20-48.eu-north-1.compute.amazonaws.com:8080/students
- Status:** 200 OK
- Time:** 138 ms
- Size:** 301 B
- Response Body (JSON):**

```
{
  "_id": "64e2b5df762d6b78a5a223ad",
  "name": "Test",
  "age": 5,
  "__v": 0
}
```

It also support Post student too:

The screenshot shows a REST client interface with a collection named "post student". A POST request is configured to the URL `http://ec2-16-171-20-48.eu-north-1.compute.amazonaws.com:8080/students`. The request body is in the "Body" tab, showing a JSON object with the following fields:

Key	Value	Description
<input checked="" type="checkbox"/> name	Test User	
<input checked="" type="checkbox"/> age	23	

The response is shown in the "Body" tab, indicating a 200 OK status with a response time of 140 ms and a body size of 305 B. The response body is a JSON object:

```
1 {
2   "name": "Test User",
3   "age": 23,
4   "_id": "64e2b7af762d6b78a5a223bc",
5   "__v": 0
6 }
```

As a result, new student is added as seen in a new get request:

The screenshot shows a REST client interface with a collection named "aws students". A GET request is configured to the URL `http://ec2-16-171-20-48.eu-north-1.compute.amazonaws.com:8080/students`. The response is shown in the "Body" tab, indicating a 200 OK status with a response time of 137 ms and a body size of 373 B. The response body is a JSON array containing two student objects:

```
1 [
2   {
3     "_id": "64e2b5df762d6b78a5a223ad",
4     "name": "Test",
5     "age": 5,
6     "__v": 0
7   },
8   {
9     "_id": "64e2b7af762d6b78a5a223bc",
10    "name": "Test User",
11    "age": 23,
12    "__v": 0
13  }
14 ]
```

Deleting a student using id:

The screenshot shows a REST client interface with a new collection named "delete students". The request method is "DELETE" and the URL is "http://ec2-16-171-20-48.eu-north-1.compute.amazonaws.com:8080/students/:id". The "Path Variables" table shows the variable "id" with the value "64e2b5df762d6b78a5a223ad". The "Body" tab is selected, showing a JSON response in "Pretty" format:

```
1 {
2   "_id": "64e2b5df762d6b78a5a223ad",
3   "name": "Test",
4   "age": 5,
5   "__v": 0
6 }
```

The status bar indicates a 200 OK response with 148 ms latency and 299 B of data.

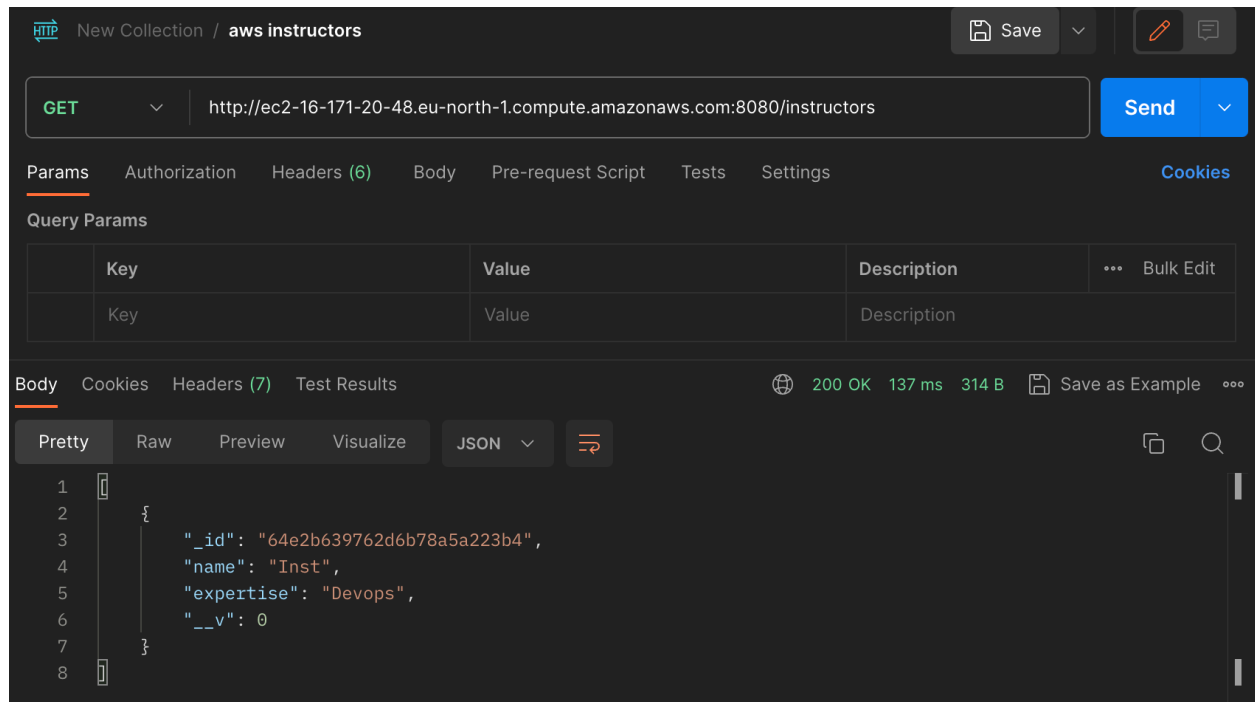
The deletion result is as expected:

The screenshot shows a REST client interface with a new collection named "aws students". The request method is "GET" and the URL is "http://ec2-16-171-20-48.eu-north-1.compute.amazonaws.com:8080/students". The "Query Params" table is empty. The "Body" tab is selected, showing a JSON response in "Pretty" format:

```
1 {
2   "_id": "64e2b7af762d6b78a5a223bc",
3   "name": "Test User",
4   "age": 23,
5   "__v": 0
6 }
```

The status bar indicates a 200 OK response with 137 ms latency and 307 B of data.

All requests are also valid for instructors and courses too:



HTTP New Collection / aws instructors

GET http://ec2-16-171-20-48.eu-north-1.compute.amazonaws.com:8080/instructors Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

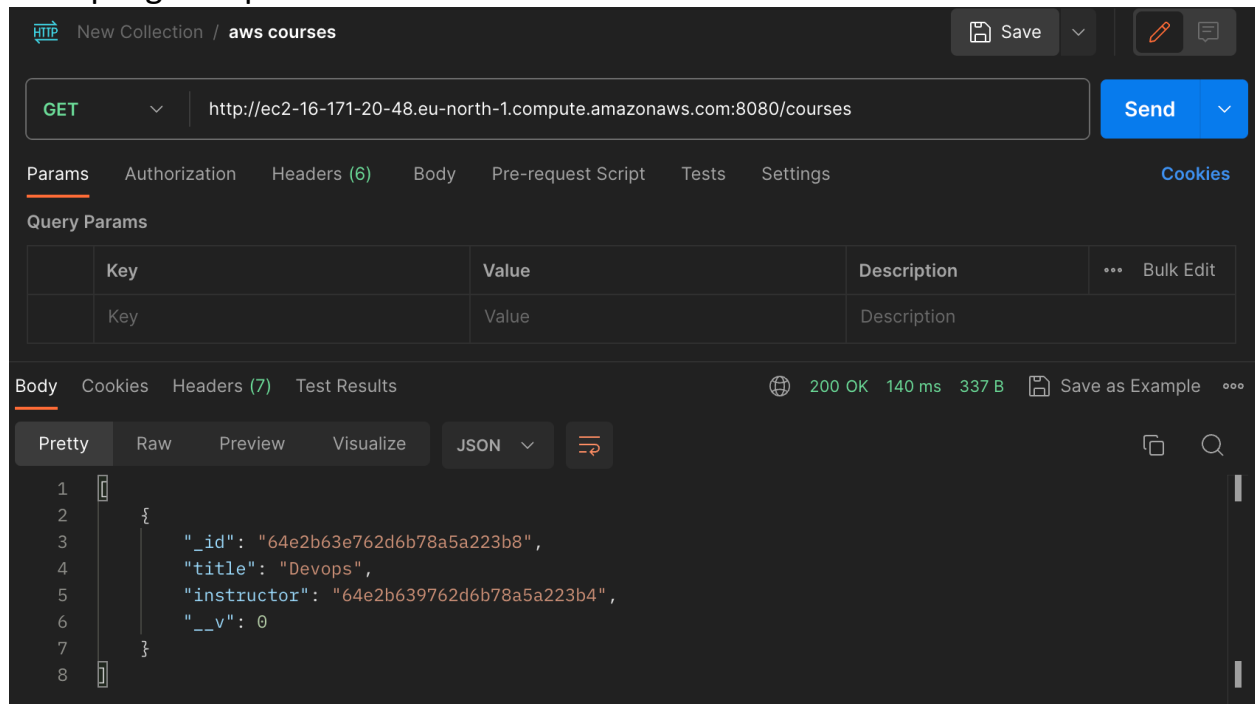
Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results 200 OK 137 ms 314 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "64e2b639762d6b78a5a223b4",
3   "name": "Inst",
4   "expertise": "Devops",
5   "__v": 0
6 }
```

Example get request for courses:



HTTP New Collection / aws courses

GET http://ec2-16-171-20-48.eu-north-1.compute.amazonaws.com:8080/courses Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results 200 OK 140 ms 337 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "64e2b63e762d6b78a5a223b8",
3   "title": "Devops",
4   "instructor": "64e2b639762d6b78a5a223b4",
5   "__v": 0
6 }
```

## Results:

As a result, all API requests work well with all tables. It runs well on EC2 machine.