

2,730 views Jun 8, 2022 #cdk #aws #awstutorial

How to create CRUD api tutorial link : [YouTube](#) • How to Create an API in AWS | API Gat...

In this tutorial, we will create AWS Cloud Infrastructure (Api Gateway, Lambda and DynamoDb Table) using JAVA language.

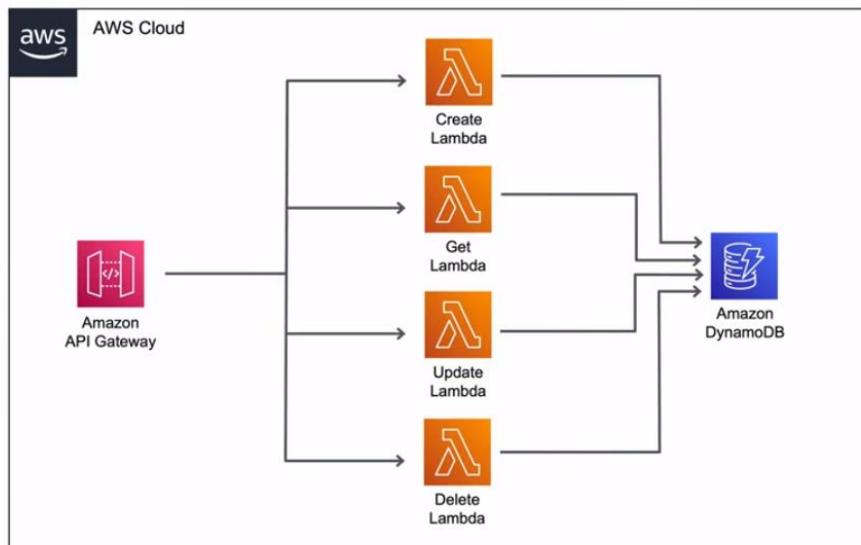
AWS CDK takes the code and compiles it down to CloudFormation template. CDK code is easy to understand and build reusable Cloud Components (Constructs).

aws cdk

In this tutorial, we will create AWS Cloud Infrastructure using JAVA language. We will create an Api gateway, 4 Lambda functions, a DynamoDB Table and connect them all together.

aws cdk

Diagram



aws cdk

PREREQUISITES

AWS ACCOUNT

AWS CDK INSTALLED

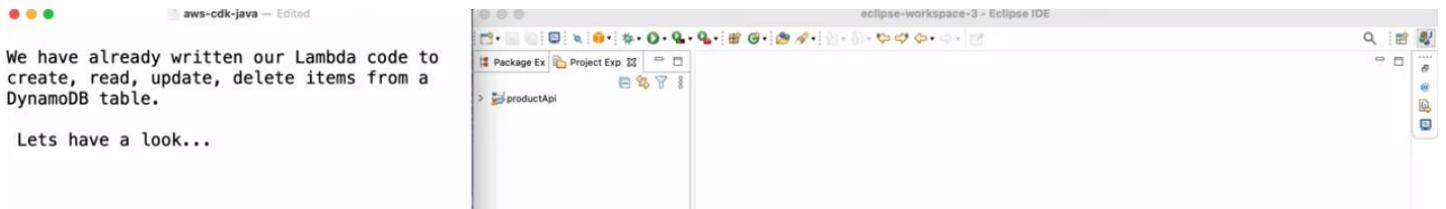
MAVEN INSTALLED

UNDERSTANDING OF BASIC JAVA

HAVE EXPERIENCE WITH POPULAR AWS SERVICES

aws cdk

LETS START ...



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.pwm.aws.product.api.lambda</groupId>
  <artifactId>productApi</artifactId>
  <version>0.1</version>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.6.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <encoding>UTF-8</encoding>
          <forceJavacCompilerUse>true</forceJavacCompilerUse>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.0.0</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-bom</artifactId>
        <version>1.11.1000</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-lambda-java-core -->
  </dependencies>

```

```
23<!--
24  <plugin>
25    <groupId>org.apache.maven.plugins</groupId>
26    <artifactId>maven-shade-plugin</artifactId>
27    <version>3.0.0</version>
28    <executions>
29      <execution>
30        <phase>package</phase>
31        <goals>
32          <goal>shade</goal>
33        </goals>
34      </execution>
35    </executions>
36  </plugin>
37</plugins>
38</build>
39<dependencyManagement>
40  <dependencies>
41    <dependency>
42      <groupId>com.amazonaws</groupId>
43      <artifactId>aws-java-sdk-bom</artifactId>
44      <version>1.11.1000</version>
45      <type>pom</type>
46      <scope>import</scope>
47    </dependency>
48  </dependencies>
49</dependencyManagement>
50<dependencies>
51  <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-lambda-java-core -->
52  <dependency>
53    <groupId>com.amazonaws</groupId>
54    <artifactId>aws-lambda-java-core</artifactId>
55    <version>1.2.1</version>
56  </dependency>
57  <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-dynamodb -->
58  <dependency>
59    <groupId>com.amazonaws</groupId>
60    <artifactId>aws-java-sdk-dynamodb</artifactId>
61  </dependency>
62  <dependency>
63    <groupId>com.googlecode.json-simple</groupId>
64    <artifactId>json-simple</artifactId>
65    <version>1.1.1</version>
66  </dependency>
67  <dependency>
68    <groupId>com.googlecode.gson</groupId>
69    <artifactId>gson</artifactId>
70    <version>2.8.5</version>
71  </dependency>
72</dependencies>
73</project>
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

```
productApi/pom.xml
1 package com.pwm.aws.crud.product.api.model;
2
3 import com.google.gson.Gson;
4
5 public class Product {
6     private int id;
7     private String name;
8     private double price;
9
10    public Product(int id, String name, double price) {
11        this.id = id;
12        this.name = name;
13        this.price = price;
14    }
15
16    public Product(String json) {
17        Gson gson = new Gson();
18        Product tempProduct = gson.fromJson(json, Product.class);
19        this.id = tempProduct.id;
20        this.name = tempProduct.name;
21        this.price = tempProduct.price;
22    }
23
24    public String toString() {
25        return new Gson().toJson(this);
26    }
27
28    public int getId() {
29        return id;
30    }
31
32    public void setId(int id) {
33        this.id = id;
34    }
35
36    public String getName() {
37        return name;
38    }
39
40    public void setName(String name) {
41        this.name = name;
42    }
43
44    public double getPrice() {
45        return price;
46    }
47
48    public void setPrice(double price) {
49        this.price = price;
50    }
51
52 }
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

```
productApi/pom.xml
1 package com.pwm.aws.crud.product.api.model;
2
3 import java.util.Collections;
4
5 public class ApiResponse {
6     private final String body;
7     private final Map<String, String> headers;
8     private final int statusCode;
9
10    public ApiResponse(final String body, final Map<String, String> headers, final int statusCode) {
11        this.body = body;
12        this.statusCode = statusCode;
13        this.headers = Collections.unmodifiableMap(new HashMap<String, String>(headers));
14    }
15
16    public String getBody() {
17        return body;
18    }
19
20    public Map<String, String> getHeaders() {
21        return headers;
22    }
23
24    public int getStatusCode() {
25        return statusCode;
26    }
27
28
29
30 }
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

```
productApi/pom.xml
1 package com.pwm.aws.crud.product.api.model;
2
3 import java.util.Collections;
4
5 class ApiResponse {
6     private final String body;
7     private final Map<String, String> headers;
8     private final int statusCode;
9
10    public ApiResponse(final String body, final Map<String, String> headers, final int statusCode) {
11        this.body = body;
12        this.statusCode = statusCode;
13        this.headers = Collections.unmodifiableMap(new HashMap<String, String>(headers));
14    }
15
16    public String getBody() {
17        return body;
18    }
19
20    public Map<String, String> getHeaders() {
21        return headers;
22    }
23
24    public int getStatusCode() {
25        return statusCode;
26    }
27
28
29
30 }
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

```

1 package com.pwm.aws.crud.product.api;
2
3 import java.util.HashMap;
4
5 public class CreateProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
6
7     @Override
8     public ApiResponse handleRequest(Map<String, Object> input, Context context) {
9
10        ApiResponse apiResponse = null;
11        Map<String, String> headers = new HashMap<>();
12        headers.put("Content-Type", "application/json");
13
14        try {
15            if (input.get("body") != null) {
16                Product product = new Product((String) input.get("body"));
17                apiResponse = new ApiResponse(createProduct(product), headers, 200);
18            }
19        } catch (Exception e) {
20            JSONObject errorObj = new JSONObject();
21            errorObj.put("error", e);
22            apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
23        }
24
25        return apiResponse;
26    }
27
28    @Override
29    @SuppressWarnings("unchecked")
30    private String createProduct(Product product) {
31        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();
32        DynamoDB dynamoDB = new DynamoDB(client);
33        String tableName = System.getenv("TABLE_NAME");
34        String primaryKey = System.getenv("PRIMARY_KEY");
35        Table table = dynamoDB.getTable(tableName);
36        Item item = new Item().withPrimaryKey(primaryKey, product.getId())
37            .withString("name", product.getName())
38            .withNumber("price", product.getPrice());
39
40        table.putItem(item);
41        JSONObject jsonObject = new JSONObject();
42        jsonObject.put("message", "Item created with ID : " + product.getId());
43        return jsonObject.toString();
44    }
45
46}

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

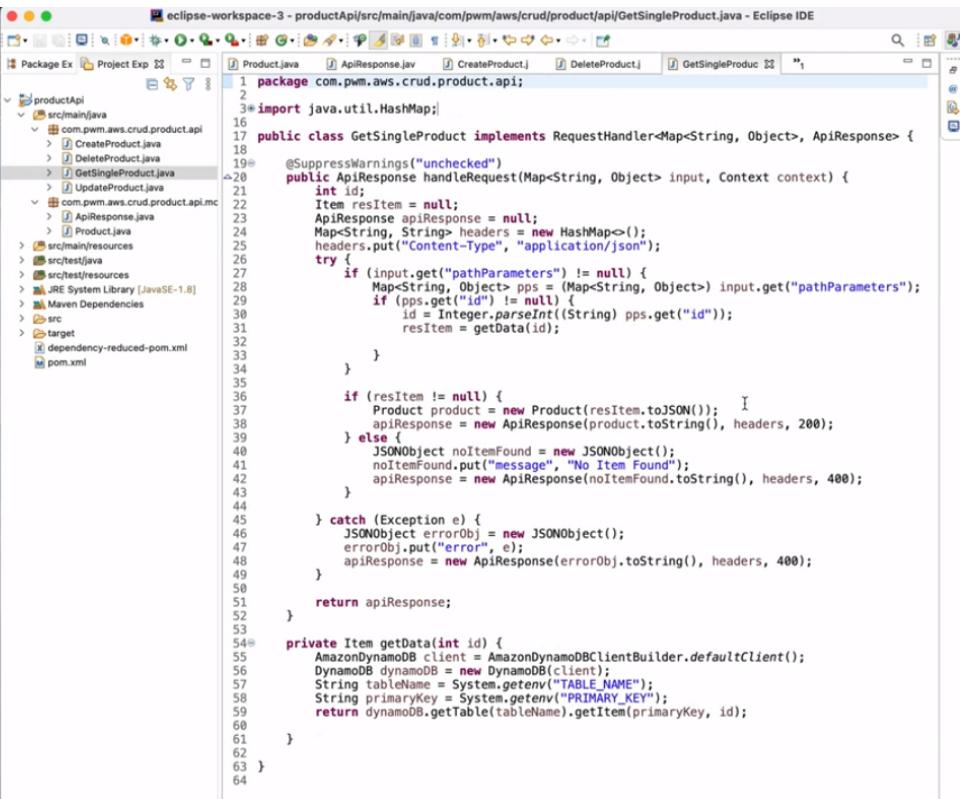
```

1 package com.pwm.aws.crud.product.api;
2
3 import java.util.HashMap;
4
5 public class DeleteProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
6
7     @Override
8     public ApiResponse handleRequest(Map<String, Object> input, Context context) {
9
10        int id;
11        ApiResponse apiResponse = null;
12        Map<String, String> headers = new HashMap<>();
13        headers.put("Content-Type", "application/json");
14
15        try {
16            if (input.get("pathParameters") != null) {
17                Map<String, Object> pps = (Map<String, Object>) input.get("pathParameters");
18                if (pps.get("id") != null) {
19                    id = Integer.parseInt(String.valueOf(pps.get("id")));
20                    apiResponse = new ApiResponse(deleteProduct(id), headers, 200);
21                }
22            }
23        } catch (Exception e) {
24            JSONObject errorObj = new JSONObject();
25            errorObj.put("error", e);
26            apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
27        }
28
29        return apiResponse;
30    }
31
32    @Override
33    @SuppressWarnings("unchecked")
34    private String deleteProduct(int id) {
35        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();
36        DynamoDB dynamoDB = new DynamoDB(client);
37        String tableName = System.getenv("TABLE_NAME");
38        String primaryKey = System.getenv("PRIMARY_KEY");
39        dynamoDB.getTable(tableName).deleteItem(primaryKey, id);
40
41        JSONObject jsonObject = new JSONObject();
42        jsonObject.put("message", "Item deleted with ID : " + id);
43        return jsonObject.toString();
44    }
45
46}

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...



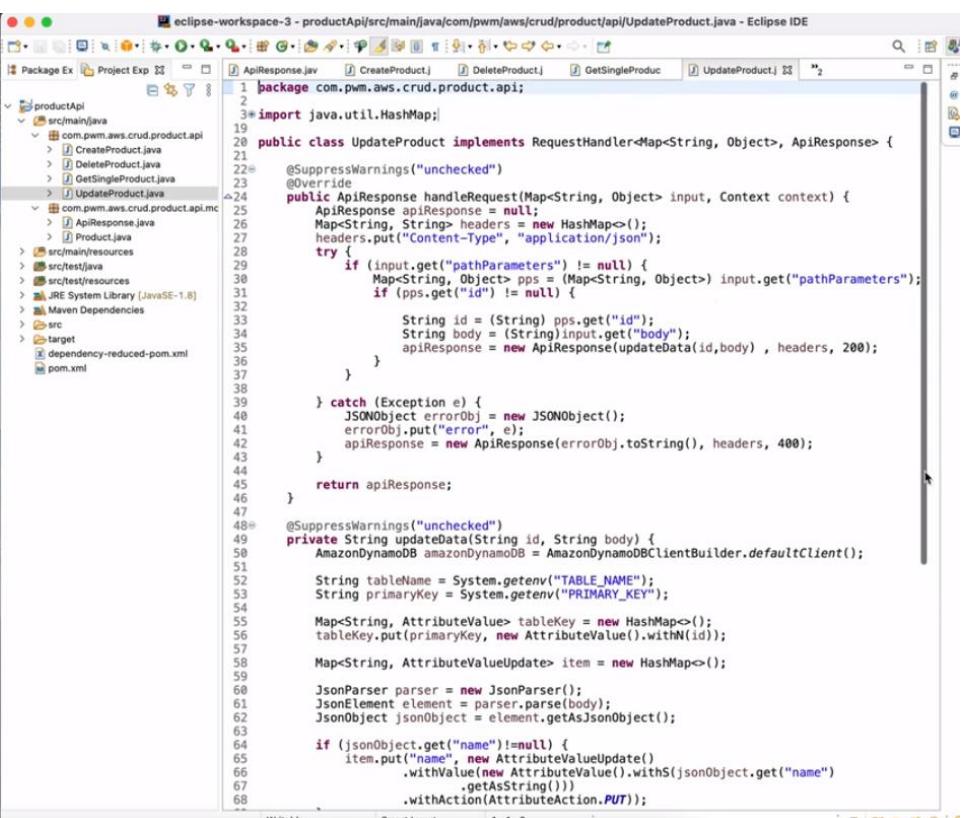
```

1 package com.pwm.aws.crud.product.api;
2
3 import java.util.HashMap;
4
5 public class GetSingleProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
6
7     @Override
8     public ApiResponse handleRequest(Map<String, Object> input, Context context) {
9         int id;
10        Item resItem = null;
11        ApiResponse apiResponse = null;
12        Map<String, String> headers = new HashMap<>();
13        headers.put("Content-Type", "application/json");
14
15        try {
16            if (input.get("pathParameters") != null) {
17                Map<String, Object> pps = (Map<String, Object>) input.get("pathParameters");
18                if (pps.get("id") != null) {
19                    id = Integer.parseInt(String.valueOf(pps.get("id")));
20                    resItem = getDynamoDB().getItem(tableName, primaryKey, id);
21                }
22            }
23
24            if (resItem != null) {
25                Product product = new Product(resItem.toJSONString());
26                apiResponse = new ApiResponse(product.toString(), headers, 200);
27            } else {
28                JSONObject noItemFound = new JSONObject();
29                noItemFound.put("message", "No Item Found");
30                apiResponse = new ApiResponse(noItemFound.toString(), headers, 400);
31            }
32
33        } catch (Exception e) {
34            JSONObject errorObj = new JSONObject();
35            errorObj.put("error", e);
36            apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
37        }
38
39        return apiResponse;
40    }
41
42    private Item getData(int id) {
43        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();
44        DynamoDB dynamoDB = new DynamoDB(client);
45        String tableName = System.getenv("TABLE_NAME");
46        String primaryKey = System.getenv("PRIMARY_KEY");
47        return dynamoDB.getTable(tableName).getItem(primaryKey, id);
48    }
49
50 }

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...



```

1 package com.pwm.aws.crud.product.api;
2
3 import java.util.HashMap;
4
5 public class UpdateProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
6
7     @Override
8     public ApiResponse handleRequest(Map<String, Object> input, Context context) {
9         ApiResponse apiResponse = null;
10        Map<String, String> headers = new HashMap<>();
11        headers.put("Content-Type", "application/json");
12
13        try {
14            if (input.get("pathParameters") != null) {
15                Map<String, Object> pps = (Map<String, Object>) input.get("pathParameters");
16                if (pps.get("id") != null) {
17
18                    String id = (String) pps.get("id");
19                    String body = (String) input.get("body");
20                    apiResponse = new ApiResponse(updateData(id, body), headers, 200);
21                }
22            }
23
24        } catch (Exception e) {
25            JSONObject errorObj = new JSONObject();
26            errorObj.put("error", e);
27            apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
28        }
29
30        return apiResponse;
31    }
32
33    @SuppressWarnings("unchecked")
34    private String updateData(String id, String body) {
35        AmazonDynamoDB amazonDynamoDB = AmazonDynamoDBClientBuilder.defaultClient();
36
37        String tableName = System.getenv("TABLE_NAME");
38        String primaryKey = System.getenv("PRIMARY_KEY");
39
40        Map<String, AttributeValue> tableKey = new HashMap<>();
41        tableKey.put(primaryKey, new AttributeValue().withS(id));
42
43        Map<String, AttributeValueUpdate> item = new HashMap<>();
44
45        JsonParser parser = new JsonParser();
46        JsonElement element = parser.parse(body);
47        JsonObject jsonObject = element.getAsJsonObject();
48
49        if (jsonObject.get("name") != null) {
50            item.put("name", new AttributeValueUpdate()
51                .WithValue(new AttributeValue().withS(jsonObject.get("name")
52                    .getAsString())))
53                .withAction(AttributeAction.PUT));
54
55        }
56
57        return amazonDynamoDB.updateItem(tableName, tableKey, item);
58    }
59
60 }

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

```
43     }
44 
45     return apiResponse;
46 }
47 
48+ @SuppressWarnings("unchecked")
49 private String updateData(String id, String body) {
50     AmazonDynamoDB amazonDynamoDB = AmazonDynamoDBClientBuilder.defaultClient();
51 
52     String tableName = System.getenv("TABLE_NAME");
53     String primaryKey = System.getenv("PRIMARY_KEY");
54 
55     Map<String,AttributeValue> tableKey = new HashMap<>();
56     tableKey.put(primaryKey, new AttributeValue().withN(id));
57 
58     Map<String,AttributeValueUpdate> item = new HashMap<>();
59 
60     JsonParser parser = new JsonParser();
61     JsonElement element = parser.parse(body);
62     JsonObject jsonObject = element.getAsJsonObject();
63 
64     if (jsonObject.get("name")!=null) {
65         item.put("name", new AttributeValueUpdate()
66             .WithValue(new AttributeValue().withS(jsonObject.get("name")
67                 .getAsString())))
68             .withAction(AttributeAction.PUT));
69     }
70 
71     if (jsonObject.get("price")!=null) {
72         item.put("price", new AttributeValueUpdate()
73             .WithValue(new AttributeValue().withN(jsonObject.get("price")
74                 .getAsString())))
75             .withAction(AttributeAction.PUT));
76     }
77 
78     UpdateItemRequest updateItemRequest = new UpdateItemRequest();
79     /* Setting Table Name */
80     updateItemRequest.setTableName(tableName);
81     /* Setting Primary Key */
82     updateItemRequest.setKey(tableKey);
83     updateItemRequest.setReturnValues(ReturnValue.ALL_NEW);
84     updateItemRequest.setAttributeUpdates(item);
85     amazonDynamoDB.updateItem(updateItemRequest);
86 
87     JSONObject messageObject = new JSONObject();
88     messageObject.put("message", "Item Updated with ID : " + id);
89     return messageObject.toString();
90 }
91 
92 }
93 
94 }
95 }
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

```
43     }
44 
45     return apiResponse;
46 }
47 
48+ @SuppressWarnings("unchecked")
49 private String updateData(String id, String body) {
50     AmazonDynamoDB amazonDynamoDB = AmazonDynamoDBClientBuilder.defaultClient();
51 
52     String tableName = System.getenv("TABLE_NAME");
53     String primaryKey = System.getenv("PRIMARY_KEY");
54 
55     Map<String,AttributeValue> tableKey = new HashMap<>();
56     tableKey.put(primaryKey, new AttributeValue().withN(id));
57 
58     Map<String,AttributeValueUpdate> item = new HashMap<>();
59 
60     JsonParser parser = new JsonParser();
61     JsonElement element = parser.parse(body);
62     JsonObject jsonObject = element.getAsJsonObject();
63 
64     if (jsonObject.get("name")!=null) {
65         item.put("name", new AttributeValueUpdate()
66             .WithValue(new AttributeValue().withS(jsonObject.get("name")
67                 .getAsString())))
68             .withAction(AttributeAction.PUT));
69     }
70 
71     if (jsonObject.get("price")!=null) {
72         item.put("price", new AttributeValueUpdate()
73             .WithValue(new AttributeValue().withN(jsonObject.get("price")
74                 .getAsString())))
75             .withAction(AttributeAction.PUT));
76     }
77 
78     UpdateItemRequest updateItemRequest = new UpdateItemRequest();
79     /* Setting Table Name */
80     updateItemRequest.setTableName(tableName);
81     /* Setting Primary Key */
82     updateItemRequest.setKey(tableKey);
83     updateItemRequest.setReturnValues(ReturnValue.ALL_NEW);
84     updateItemRequest.setAttributeUpdates(item);
85     amazonDynamoDB.updateItem(updateItemRequest);
86 
87     JSONObject messageObject = new JSONObject();
88     messageObject.put("message", "Item Updated with ID : " + id);
89     return messageObject.toString();
90 }
91 
92 }
93 
94 }
95 }
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

```

ClientBuilder.defaultClient();

);

Map<String, String> map = new HashMap<>();
map.put("id", id);
map.put("name", name);
map.put("category", category);
map.put("price", price);
map.put("description", description);

Map<String, String> shMap = new HashMap<>();
shMap.put("id", id);
shMap.put("name", name);
shMap.put("category", category);
shMap.put("price", price);
shMap.put("description", description);

(
);

nS(jsonObject.get("name"))

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

```

ClientBuilder.defaultClient();

);

Map<String, String> map = new HashMap<>();
map.put("id", id);
map.put("name", name);
map.put("category", category);
map.put("price", price);
map.put("description", description);

Map<String, String> shMap = new HashMap<>();
shMap.put("id", id);
shMap.put("name", name);
shMap.put("category", category);
shMap.put("price", price);
shMap.put("description", description);

(
);

nS(jsonObject.get("name"))

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run `$ cdk init app --language java` in that folder

```

ClientBuilder.defaultClient();

);

Map<String, String> map = new HashMap<>();
map.put("id", id);
map.put("name", name);
map.put("category", category);
map.put("price", price);
map.put("description", description);

Map<String, String> shMap = new HashMap<>();
shMap.put("id", id);
shMap.put("name", name);
shMap.put("category", category);
shMap.put("price", price);
shMap.put("description", description);

(
);

nS(jsonObject.get("name"))

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run `$ cdk init app --language java` in that folder

```

Last login: Wed Jun  8 12:10:57 on ttys000
productApiCDK:~ mdomor.farque$ 

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MRMIOMP2030:productApiCDK mdomor.farque$ 

```

```

ClientBuilder.defaultClient();

);

Map<String, String> map = new HashMap<>();
map.put("id", id);
map.put("name", name);
map.put("category", category);
map.put("price", price);
map.put("description", description);

Map<String, String> shMap = new HashMap<>();
shMap.put("id", id);
shMap.put("name", name);
shMap.put("category", category);
shMap.put("price", price);
shMap.put("description", description);

(
);

nS(jsonObject.get("name"))

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

eclipse-workspace-3

```
productApiCDK — node /usr/local/bin/cdk init app --language java — 80x24
Last login: Wed Jun 8 12:10:57 on ttys000
The default interactive shell is now zsh.
To update your account to use zsh, please run 'chsh -s /bin/zsh'.
For more details, please visit https://support.apple.com/kb/HT208050.
MRM1OMP2030:productApiCDK mdomor.farouque$ cdk init app --language java
```

UpdateProduct.java - Eclipse IDE

```
ClientBuilder.defaultClient();
);
Map<String, String> idMap = Map.of("id", id);
Map<String, String> nameMap = Map.of("name", name);
Map<String, String> priceMap = Map.of("price", price);
Map<String, String> itemRequestMap = Map.of("itemRequest", itemRequest);

Map<String, String> idMap = Map.of("id", id);
Map<String, String> nameMap = Map.of("name", name);
Map<String, String> priceMap = Map.of("price", price);
Map<String, String> itemRequestMap = Map.of("itemRequest", itemRequest);
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

This will create an empty aws cdk app with language JAVA

We can now open the Project in an IDE

eclipse-workspace-3

```
This is a blank project for CDK development with Java.
The 'cdk.json' file tells the CDK Toolkit how to execute your app.
It is a [Maven](https://maven.apache.org/) based project, so you can open this project with any Maven compatible Java IDE to build and run tests.

## Useful commands
* `mvn package` compile and run tests
* `cdk ls` list all stacks in the app
* `cdk synth` emits the synthesized CloudFormation template
* `cdk deploy` deploy this stack to your default AWS account/region
* `cdk diff` compare deployed stack with current state
* `cdk docs` open CDK documentation

Enjoy!
Initializing a new git repository...
Executing 'mvn package'
All done!
```

UpdateProduct.java - Eclipse IDE

```
ClientBuilder.defaultClient();
);
Map<String, String> idMap = Map.of("id", id);
Map<String, String> nameMap = Map.of("name", name);
Map<String, String> priceMap = Map.of("price", price);
Map<String, String> itemRequestMap = Map.of("itemRequest", itemRequest);
```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

This will create an empty aws cdk app with language JAVA

We can now open the Project in an IDE

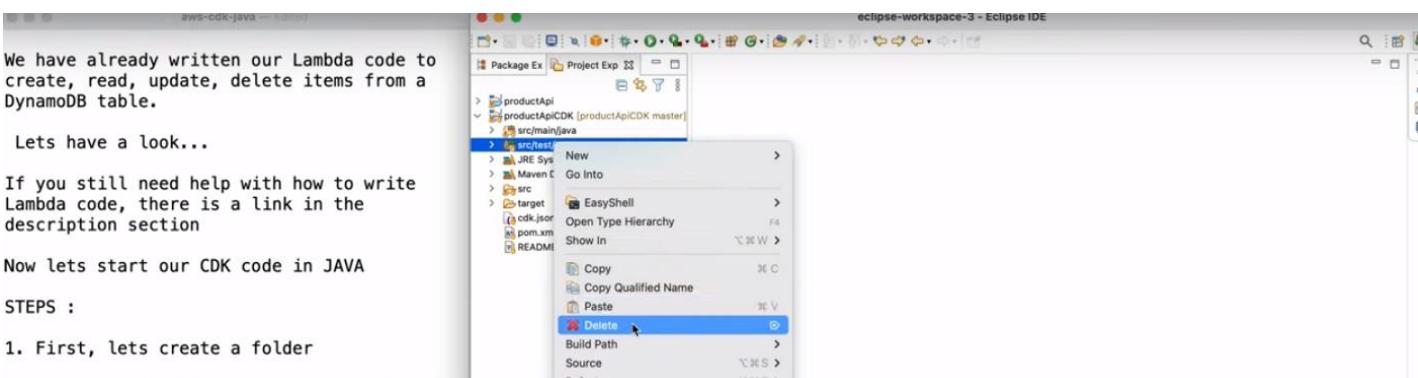
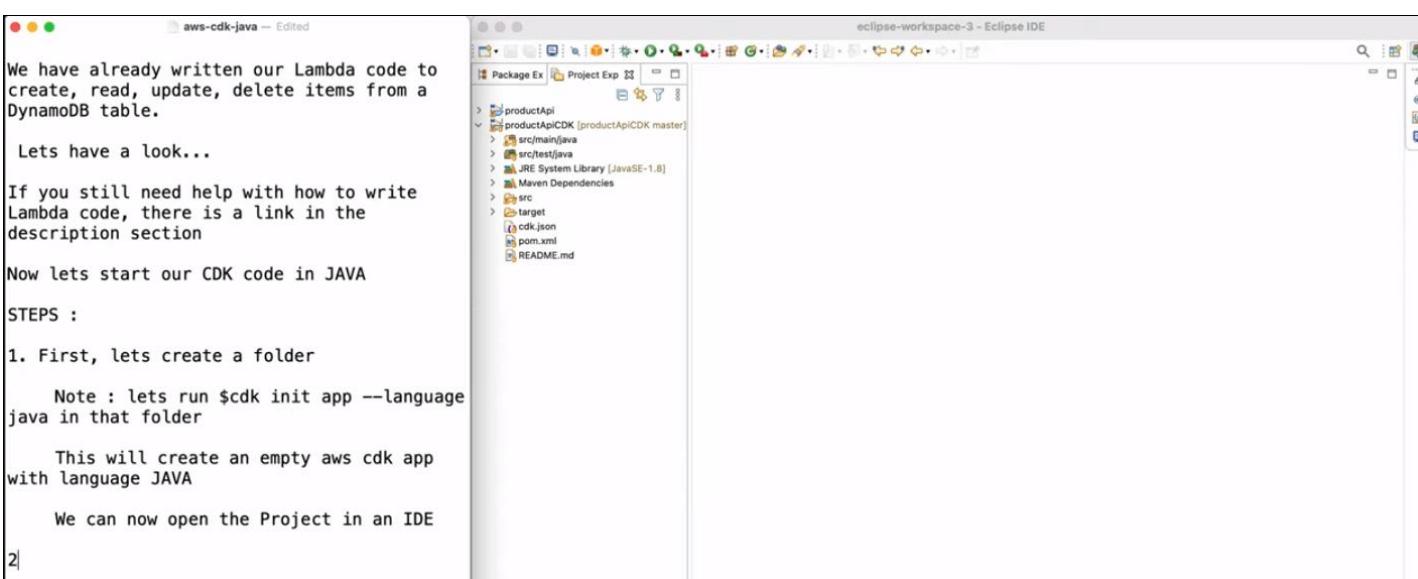
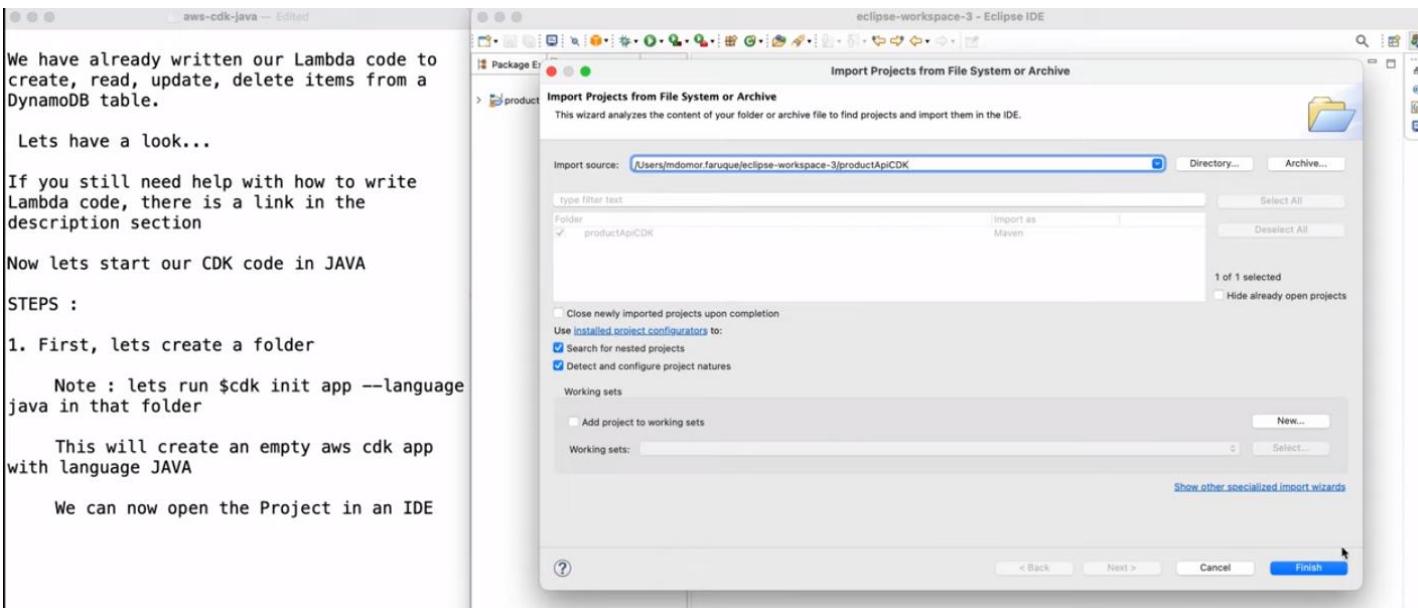
eclipse-workspace-3 - Eclipse IDE

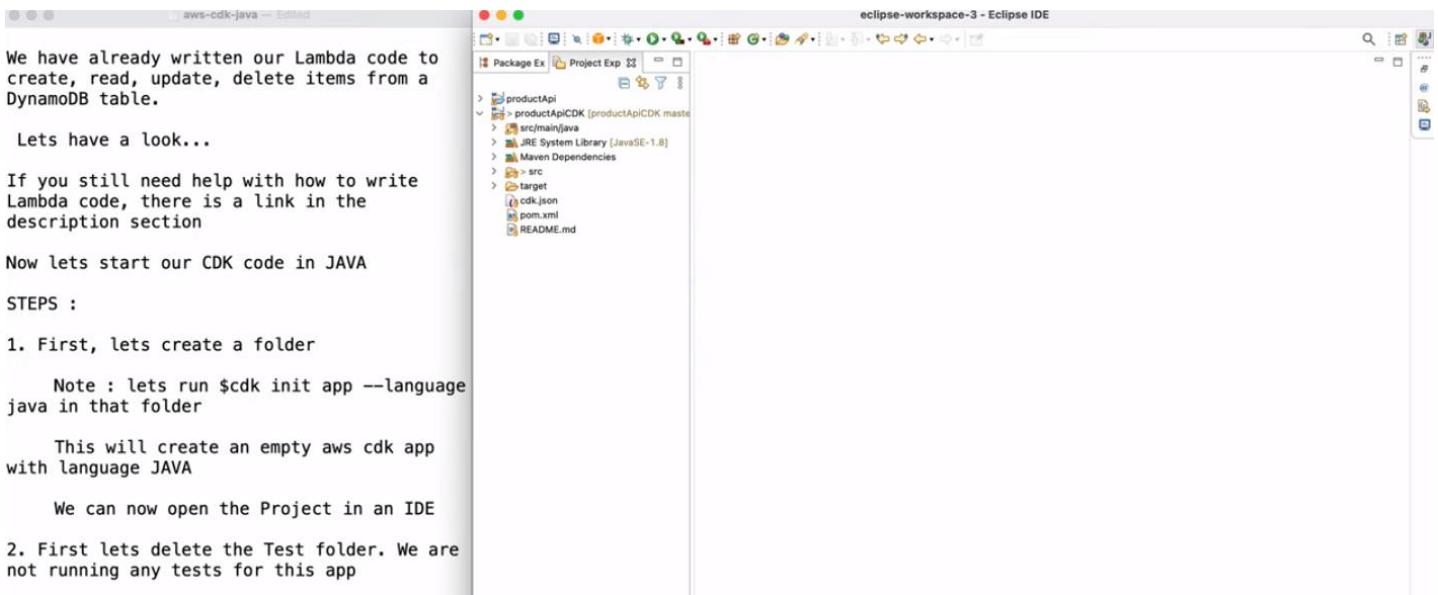
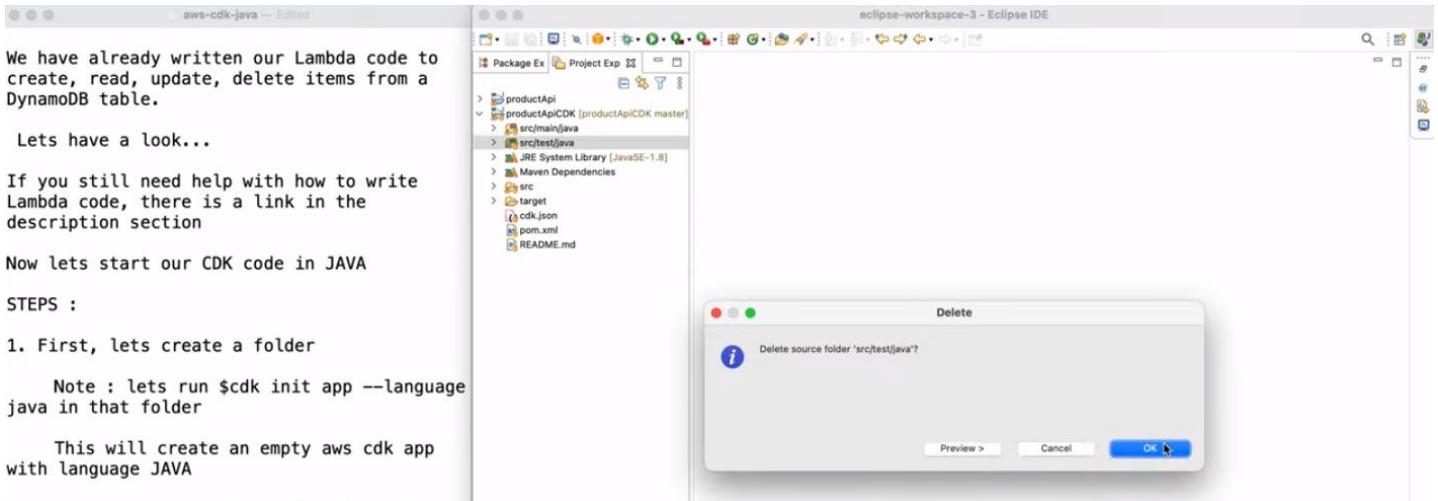
Import Projects from File System or Archive

Select the folder to find projects to import

Name	Size	Kind	Date Added
productApiCDK		Folder	Today at 12:34 PM
etc		Folder	Yesterday at 11:25 PM
productApi		Folder	Jun 3, 2022 at 4:51 PM

New Folder Cancel Open





We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

This will create an empty aws cdk app with language JAVA

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

```

<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.myorg</groupId>
  <artifactId>product_api_cdk</artifactId>
  <version>0.1</version>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <cdk.version>2.27.0</cdk.version>
    <constructs.version>10.0.0,11.0.0</constructs.version>
    <junit.version>5.7.1</junit.version>
  </properties>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>3.0.0</version>
        <configuration>
          <mainClass>com.myorg.ProductApiCdkApp</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <!-- AWS Cloud Development Kit -->
    <dependency>
      <groupId>software.amazon.awscdk</groupId>
      <artifactId>aws-cdk-lib</artifactId>
      <version>${cdk.version}</version>
    </dependency>
    <dependency>
      <groupId>software.constructs</groupId>
      <artifactId>constructs</artifactId>
      <version>${constructs.version}</version>
    </dependency>
  </dependencies>

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

This will create an empty aws cdk app with language JAVA

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

```

<?xml version="1.0" encoding="UTF-8"?>
<project build.sourceEncoding="UTF-8" cdk.version="2.27.0" constructs.version="10.0.0,11.0.0" junit.version="5.7.1">
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <cdk.version>2.27.0</cdk.version>
    <constructs.version>10.0.0,11.0.0</constructs.version>
    <junit.version>5.7.1</junit.version>
  </properties>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>3.0.0</version>
        <configuration>
          <mainClass>com.myorg.ProductApiCdkApp</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <!-- AWS Cloud Development Kit -->
    <dependency>
      <groupId>software.amazon.awscdk</groupId>
      <artifactId>aws-cdk-lib</artifactId>
      <version>${cdk.version}</version>
    </dependency>
    <dependency>
      <groupId>software.constructs</groupId>
      <artifactId>constructs</artifactId>
      <version>${constructs.version}</version>
    </dependency>
  </dependencies>

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

This will create an empty aws cdk app with language JAVA

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

```

1 package com.myorg;
2
3* import software.amazon.awscdk.App;
4
5 public class ProductApiCdkApp {
6     public static void main(final String[] args) {
7         App app = new App();
8
9         new ProductApiCdkStack(app, "ProductApiCdkStack", StackProps.builder())
10            // If you don't specify 'env', this stack will be environment-agnostic.
11            // Account/Region-dependent features and context lookups will not work,
12            // but a single synthesized template can be deployed anywhere.
13            .env(Environment.builder()
14                .account(System.getenv("CDK_DEFAULT_ACCOUNT"))
15                .region(System.getenv("CDK_DEFAULT_REGION")))
16            .build());
17
18            // Uncomment the next block to specialize this stack for the AWS Account
19            // and Region that are implied by the current CLI configuration.
20            /*
21            .env(Environment.builder()
22                .account("123456789012")
23                .region("us-east-1")
24                .build());
25
26            // Uncomment the next block if you know exactly what Account and Region you
27            // want to deploy the stack to.
28            /*
29            .env(Environment.builder()
30                .account("123456789012")
31                .region("us-east-1")
32                .build());
33
34
35            // For more information, see https://docs.aws.amazon.com/cdk/latest/guide/env
36            .build());
37
38        }
39    }
40
41 }
42
43

```

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

This will create an empty aws cdk app with language JAVA

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we

```

1 package com.myorg;
2
3* import software.amazon.awscdk.App;
4
5 public class ProductApiCdkApp {
6     public static void main(final String[] args) {
7         App app = new App();
8
9         new ProductApiCdkStack(app, "ProductApiCdkStack", StackProps.builder())
10            // If you don't specify 'env', this stack will be environment-agnostic.
11            // Account/Region-dependent features and context lookups will not work,
12            // but a single synthesized template can be deployed anywhere.
13            .env(Environment.builder()
14                .account(System.getenv("CDK_DEFAULT_ACCOUNT"))
15                .region(System.getenv("CDK_DEFAULT_REGION")))
16            .build());
17
18            // Uncomment the next block to specialize this stack for the AWS Account
19            // and Region that are implied by the current CLI configuration.
20            /*
21            .env(Environment.builder()
22                .account("123456789012")
23                .region("us-east-1")
24                .build());
25
26            // Uncomment the next block if you know exactly what Account and Region you
27            // want to deploy the stack to.
28            /*
29            .env(Environment.builder()
30                .account("123456789012")
31                .region("us-east-1")
32                .build());
33
34
35            // For more information, see https://docs.aws.amazon.com/cdk/latest/guide/env
36            .build());
37
38        }
39    }
40
41 }
42
43

```

This is where we will set our account info

We have already written our Lambda code to create, read, update, delete items from a DynamoDB table.

Lets have a look...

If you still need help with how to write Lambda code, there is a link in the description section

Now lets start our CDK code in JAVA

STEPS :

1. First, lets create a folder

Note : lets run \$cdk init app --language java in that folder

```

1 package com.myorg;
2
3* import software.constructs.Construct;
4 // import software.amazon.awscdk.Duration;
5 // import software.amazon.awscdk.services.sqs.Queue;
6
7 public class ProductApiCdkStack extends Stack {
8     public ProductApiCdkStack(final Construct scope, final String id) {
9         this(scope, id, null);
10    }
11
12
13    public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
14        super(scope, id, props);
15
16        // The code that defines your stack goes here
17
18        // example resource
19        // final Queue queue = Queue.Builder.create(this, "ProductApiCdkQueue")
20        //     .visibilityTimeout(Duration.seconds(300))
21        //     .build();
22    }
23
24 }
25

```

This is where we will write our CDK code

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

```

1 package com.myorg;
2
3 import software.amazon.awscdk.App;
4
5 public class ProductApiCdkApp {
6
7     static Environment makeEnvironment(String account, String region) {
8
9         return Environment.builder()
10            .account(account)
11            .region(region)
12            .build();
13    }
14
15    public static void main(final String[] args) {
16        App app = new App();
17
18        new ProductApiCdkStack(app, "ProductApiCdkStack", StackProps.builder()
19            // If you don't specify 'env', this stack will be environment-agnostic,
20            // Account/Region-dependent features and context lookups will not work,
21            // but a single synthesized template can be deployed anywhere.
22            // Uncomment the next block to specialize this stack for the AWS Account
23            // and Region that are implied by the current CLI configuration.
24            /*
25                .env(Environment.builder()
26                    .account(System.getenv("CDK_DEFAULT_ACCOUNT"))
27                    .region(System.getenv("CDK_DEFAULT_REGION"))
28                    .build())
29            */
30            // Uncomment the next block if you know exactly what Account and Region you
31            // want to deploy the stack to.
32            /*
33                .env(Environment.builder()
34                    .account("123456789012")
35                    .region("us-east-1")
36                    .build())
37            */
38
39        // For more information, see https://docs.aws.amazon.com/cdk/latest/guide/env
40        // .build());
41
42        app.synth();
43    }
44
45 }
46
47 
```

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

```

1 package com.myorg;
2
3 import software.amazon.awscdk.App;
4
5 public class ProductApiCdkApp {
6
7     static Environment makeEnvironment(String account, String region) {
8
9         account = (account==null) ? System.getenv("CDK_DEFAULT_ACCOUNT") : account;
10        region = (region==null) ? System.getenv("CDK_DEFAULT_REGION") : region;
11
12        return Environment.builder().account(account).region(region).build();
13    }
14
15    public static void main(final String[] args) {
16        App app = new App();
17        Environment env = makeEnvironment(null, null);
18
19        new ProductApiCdkStack(app, "ProductApiCdkStack",
20            StackProps.builder()
21                .env(env)
22                .build());
23
24        app.synth();
25    }
26
27 }
28
29 
```

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

```

1 package com.myorg;
2
3 import software.constructs.Construct;
4 // import software.amazon.awscdk.Duration;
5 // import software.amazon.awscdk.services.sqs.Queue;
6
7 public class ProductApiCdkStack extends Stack {
8
9     public ProductApiCdkStack(final Construct scope, final String id) {
10        super(scope, id, null);
11    }
12
13    public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
14        super(scope, id, props);
15
16        // The code that defines your stack goes here
17
18        // example resource
19        // final Queue queue = Queue.Builder.create(this, "ProductApiCdkQueue")
20        //     .visibilityTimeout(Duration.seconds(300))
21        //     .build();
22    }
23
24 }
25 
```

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

```

    package com.myorg;
    import software.constructs.Construct;
    import software.amazon.awscdk.Stack;
    import software.amazon.awscdk.StackProps;
    public class ProductApiCdkStack extends Stack {
        public ProductApiCdkStack(final Construct scope, final String id) {
            super(scope, id, null);
        }
        public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
            super(scope, id, props);
        }
    }

```

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

```

    package com.myorg;
    import software.constructs.Construct;
    import software.amazon.awscdk.RemovalPolicy;
    import software.amazon.awscdk.Stack;
    import software.amazon.awscdk.StackProps;
    import software.amazon.awscdk.services.dynamodb.Attribute;
    import software.amazon.awscdk.services.dynamodb.AttributeType;
    import software.amazon.awscdk.services.dynamodb.Table;
    import software.amazon.awscdk.services.dynamodb.TableProps;
    public class ProductApiCdkStack extends Stack {
        public ProductApiCdkStack(final Construct scope, final String id) {
            super(scope, id, null);
        }
        public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
            super(scope, id, props);
        }
        // Table code
        Attribute partitionKey = Attribute.builder()
            .name("id")
            .type(AttributeType.NUMBER)
            .build();
        TableProps tableProps = TableProps.builder()
            .tableName("products")
            .partitionKey(partitionKey)
            .removalPolicy(RemovalPolicy.DESTROY)
            // this will remove the table when we run cdk destroy
            .build();
        Table dynamoTable = new Table(this, "products", tableProps);
    }
}

```

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

```

19 }
20 public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
21     super(scope, id, props);
22
23     // Table code
24
25     Attribute partitionKey = Attribute.builder()
26         .name("id")
27         .type(AttributeType.NUMBER)
28         .build();
29
30     TableProps tableProps = TableProps.builder()
31         .tableName("products")
32         .partitionKey(partitionKey)
33         .removalPolicy(RemovalPolicy.DESTROY)
34         // this will remove the table when we run cdk destroy
35         .build();
36
37     Table dynamoTable = new Table(this, "products", tableProps);
38
39     // Lets create two environment variable that can be accessible from our lambda code
40
41     Map<String, String> lambdaEnvMap = new HashMap<>();
42     lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
43     lambdaEnvMap.put("PRIMARY_KEY", "id");
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

```

We can now open the Project in an IDE

2. First lets delete the Test folder. We are not running any tests for this app

Lets have a look at the code that we just created with the init command

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

```

1 package com.pwm.aws.crud.product.api;
2
3 import java.util.HashMap;
4
5 public class CreateProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
6
7     @SuppressWarnings("unchecked")
8     @Override
9     public ApiResponse handleRequest(Map<String, Object> input, Context context) {
10
11         ApiResponse apiResponse = null;
12         Map<String, String> headers = new HashMap<>();
13         headers.put("Content-Type", "application/json");
14
15         try {
16             if (input.get("body") != null) {
17                 Product product = new Product((String) input.get("body"));
18                 apiResponse = new ApiResponse(createProduct(product), headers, 200);
19             }
20         } catch (Exception e) {
21             JSONObject errorObj = new JSONObject();
22             errorObj.put("error", e);
23             apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
24         }
25
26         return apiResponse;
27     }
28
29     @SuppressWarnings("unchecked")
30     private String createProduct(Product product) {
31         AmazonDynamoDBClientBuilder.defaultClient();
32         AmazonDynamoDBClient client = AmazonDynamoDBClientBuilder.defaultClient();
33         DynamoDB dynamoDB = new DynamoDB(client);
34         String tableName = System.getenv("TABLE_NAME");
35         String primaryKey = System.getenv("PRIMARY_KEY");
36         Table table = dynamoDB.getTable(tableName);
37         Item item = new Item().withPrimaryKey(primaryKey, product.getId())
38             .withString("name", product.getName())
39             .withNumber("price", product.getPrice());
40
41         table.putItem(item);
42         JSONObject jsonObject = new JSONObject();
43         jsonObject.put("message", "Item created with ID : " + product.getId());
44         return jsonObject.toString();
45     }
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62

```

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

```

package com.pwm.aws.crud.product.api;
import java.util.HashMap;
public class GetSingleProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
    @SuppressWarnings("unchecked")
    public ApiResponse handleRequest(Map<String, Object> input, Context context) {
        int id = input.get("id");
        Item resItem = null;
        ApiResponse apiResponse = null;
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        try {
            if (input.get("pathParameters") != null) {
                Map<String, Object> pps = (Map<String, Object>) input.get("pathParameters");
                if (pps.get("id") != null) {
                    id = Integer.parseInt((String) pps.get("id"));
                    resItem = getData(id);
                }
            }
            if (resItem != null) {
                Product product = new Product(resItem.toJSONString());
                apiResponse = new ApiResponse(product.toString(), headers, 200);
            } else {
                JSONObject noItemFound = new JSONObject();
                noItemFound.put("message", "No Item Found");
                apiResponse = new ApiResponse(noItemFound.toString(), headers, 400);
            }
        } catch (Exception e) {
            JSONObject errorObj = new JSONObject();
            errorObj.put("error", e);
            apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
        }
        return apiResponse;
    }

    private Item getData(int id) {
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();
        DynamoDB dynamoDB = new DynamoDB(client);
        String tableName = System.getenv("TABLE_NAME");
        String primaryKey = System.getenv("PRIMARY_KEY");
        return dynamoDB.getTable(tableName).getItem(primaryKey, id);
    }
}

```

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

```

package com.pwm.aws.crud.product.api;
import java.util.HashMap;
public class CreateProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
    @SuppressWarnings("unchecked")
    @Override
    public ApiResponse handleRequest(Map<String, Object> input, Context context) {
        ApiResponse apiResponse = null;
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        try {
            if (input.get("body") != null) {
                Product product = new Product(input.get("body"));
                apiResponse = new ApiResponse(createProduct(product), headers, 200);
            }
        } catch (Exception e) {
            JSONObject errorObj = new JSONObject();
            errorObj.put("error", e);
            apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
        }
        return apiResponse;
    }

    @SuppressWarnings("unchecked")
    private String createProduct(Product product) {
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();
        DynamoDB dynamoDB = new DynamoDB(client);
        String tableName = System.getenv("TABLE_NAME");
        String primaryKey = System.getenv("PRIMARY_KEY");
        Table table = dynamoDB.getTable(tableName);
        Item item = new Item().withPrimaryKey(primaryKey, product.getId())
            .withString("name", product.getName())
            .withNumber("price", product.getPrice());
        table.putItem(item);
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("message", "Item created with ID : " + product.getId());
        return jsonObject.toString();
    }
}

```

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

```

package com.pwm.aws.crud.product.api;
import java.util.HashMap;
public class DeleteProduct implements RequestHandler<Map<String, Object>, ApiResponse> {
    @SuppressWarnings("unchecked")
    @Override
    public ApiResponse handleRequest(Map<String, Object> input, Context context) {
        int id;
        ApiResponse apiResponse = null;
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        try {
            if (input.get("pathParameters") != null) {
                Map<String, Object> pps = (Map<String, Object>) input.get("pathParameters");
                if (pps.get("id") != null) {
                    id = Integer.parseInt((String) pps.get("id"));
                    apiResponse = new ApiResponse(deleteProduct(id), headers, 200);
                }
            }
        } catch (Exception e) {
            JSONObject errorObj = new JSONObject();
            errorObj.put("error", e);
            apiResponse = new ApiResponse(errorObj.toString(), headers, 400);
        }
        return apiResponse;
    }
    @SuppressWarnings("unchecked")
    private String deleteProduct(int id) {
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();
        DynamoDB dynamoDB = new DynamoDB(client);
        String tableName = System.getenv("TABLE_NAME");
        String primaryKey = System.getenv("PRIMARY_KEY");
        dynamoDB.getTable(tableName).deleteItem(primaryKey, id);
        JSONObject jsonObject = new JSONObject();
        jsonObject.put("message", "Item deleted with ID : " + id);
        return jsonObject.toString();
    }
}

```

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

```

public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
    super(scope, id, props);
    // Table code
    Attribute partitionKey = Attribute.builder()
        .name("id")
        .type(AttributeType.NUMBER)
        .build();
    TableProps tableProps = TableProps.builder()
        .tableName("products")
        .partitionKey(partitionKey)
        .removalPolicy(RemovalPolicy.DESTROY)
        // this will remove the table when we run cdk destroy
        .build();
    Table dynamoTable = new Table(this, "products", tableProps);
    // Lets create two environment variable that can be accessible from our lambda code
    Map<String, String> lambdaEnvMap = new HashMap<>();
    LambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
    lambdaEnvMap.put("PRIMARY_KEY", "id");
    // Lambda code
    String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
    String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
    String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
    String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
}

```

This is where we will set our account info

This is where we will write our CDK code

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

- > So we have a Dynamo Table, Four Lambda functions and Api gateway
- > Lets start with Dynamo Table
- > Lets have a look at our Lambda code where we access this variables
- > Lets write all four of our Lambda functions handler name
- > Now create Lambda Function objects with their properties
- > Lets create a helper method

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE

```

19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
    this(scope, id, null);
}

public ProductApiCdkStack(final Construct scope, final String id, final StackProps props)
super(scope, id, props);

// Table code

Attribute partitionKey = Attribute.builder()
.name("id")
.type(AttributeType.NUMBER)
.build();

TableProps tableProps = TableProps.builder()
.tableName("products")
.partitionKey(partitionKey)
.removePolicy(RemovalPolicy.DESTROY)
// this will remove the table when we run cdk destroy
.build();

Table dynamoTable = new Table(this,"products", tableProps);

// Lets create two environment variable that can be accessible from our lambda code

Map<String, String> lambdaEnvMap = new HashMap<>();
lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
lambdaEnvMap.put("PRIMARY_KEY", "id");

// Lambda code

String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";

Function createFunction = new Function(this, "createProductFunction", null);

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

- > So we have a Dynamo Table, Four Lambda functions and Api gateway
- > Lets start with Dynamo Table
- > Lets have a look at our Lambda code where we access this variables
- > Lets write all four of our Lambda functions handler name
- > Now create Lambda Function objects with their properties
- > Lets create a helper method
- > Lets get the absolute path for our jar file

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE

Right-click context menu for the 'productApi' package:

- Open
- Copy
- Copy Qualified Name
- Paste
- Delete
- Build Path
- Move...
- Rename...
- Import...
- Export...
- Refresh
- Coverage As
- Run As
- Debug As
- Profile As
- Team
- Compare With
- Replace With

```

19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
70
    this(scope, id, null);
}

public ProductApiCdkStack(final Construct scope, final String id, final StackProps props)
super(scope, id, props);

// Table code

Attribute partitionKey = Attribute.builder()
.name("id")
.type(AttributeType.NUMBER)
.build();

TableProps tableProps = TableProps.builder()
.tableName("products")
.partitionKey(partitionKey)
.removePolicy(RemovalPolicy.DESTROY)
// this will remove the table when we run cdk destroy
.build();

Table dynamoTable = new Table(this,"products", tableProps);

Open an Editor on the Selected Element environment variable that can be accessible from our lambda code

Map<String, String> lambdaEnvMap = new HashMap<>();
lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
lambdaEnvMap.put("PRIMARY_KEY", "id");

// Lambda code

String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";

Function createFunction = new Function(this, "createProductFunction", null);

vate FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String han
    return FunctionProps.builder()
        .code(Code.fromAsset(handler))
        .build();
}

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

- > So we have a Dynamo Table, Four Lambda functions and Api gateway
- > Lets start with Dynamo Table
- > Lets have a look at our Lambda code where we access this variables
- > Lets write all four of our Lambda functions handler name
- > Now create Lambda Function objects with their properties
- > Lets create a helper method
- > Lets get the absolute path for our jar file

```

FUNCTION createFunction = new Function(this, "CreateProductFunction", null);
    this(scope, id, null);

private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handlerName) {
    return FunctionProps.builder()
        .code(Code.fromAsset(handlerName))
    }
}

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

- > So we have a Dynamo Table, Four Lambda functions and Api gateway
- > Lets start with Dynamo Table
- > Lets have a look at our Lambda code where we access this variables
- > Lets write all four of our Lambda functions handler name
- > Now create Lambda Function objects with their properties
- > Lets create a helper method
- > Lets get the absolute path for our jar file

```

FUNCTION createFunction = new Function(this, "CreateProductFunction", null);
    this(scope, id, null);

private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handlerName) {
    return FunctionProps.builder()
        .code(Code.fromAsset(handlerName))
    }
}

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE

```

21    this(scope, id, null);
22  }
23
24  public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
25    super(scope, id, props);
26
27    // Table code
28
29    Attribute partitionKey = Attribute.builder()
30      .name("id")
31      .type(AttributeType.NUMBER)
32      .build();
33
34    TableProps tableProps = TableProps.builder()
35      .tableName("products")
36      .partitionKey(partitionKey)
37      .removalPolicy(RemovalPolicy.DESTROY)
38      // this will remove the table when we run cdk destroy
39      .build();
40
41    Table dynamoTable = new Table(this, "products", tableProps);
42
43    // Lets create two environment variable that can be accessible from our lambda code
44
45    Map<String, String> lambdaEnvMap = new HashMap<>();
46    lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
47    lambdaEnvMap.put("PRIMARY_KEY", "id");
48
49    // Lambda code
50
51    String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
52    String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
53    String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
54    String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
55
56    Function createFunction = new Function(this, "createProductFunction", null,
57
58    {
59      ...
60    }
61
62    private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
63      return FunctionProps.builder()
64        .code(code.fromAsset(handler))
65        ...
66    }
67
68  }
69
70
71
72
73

```

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE

```

21  );
22
23
24  @Construct final Construct scope, final String id, final StackProps props) {
25  ops);
26
27
28  Key = Attribute.builder()
29    .type(AttributeType.NUMBER)
30    .build();
31
32  ps = TableProps.builder()
33    .tableName("products")
34    .partitionKey(partitionKey)
35    .removalPolicy(RemovalPolicy.DESTROY)
36    .build();
37
38    // this will remove the table when we run cdk destroy
39
40    new Table(this, "products", tableProps);
41
42    environment variable that can be accessible from our lambda code
43
44    lambdaEnvMap = new HashMap<>();
45    lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
46    lambdaEnvMap.put("PRIMARY_KEY", "id");
47
48
49    onHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
50    andlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
51    onHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
52    onHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
53
54    tion = new Function(this, "createProductFunction", null,
55
56
57    etLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
58      s.builder()
59        .fromAsset("file:///Users/mdomor.faruque/eclipse-workspace-3/productApi/target/productApi-0.1.jar")
60
61
62
63
64
65
66
67
68
69
70
71
72
73

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

- > So we have a Dynamo Table, Four Lambda functions and Api gateway
- > Lets start with Dynamo Table
- > Lets have a look at our Lambda code where we access this variables
- > Lets write all four of our Lambda functions handler name
- > Now create Lambda Function objects with their properties
- > Lets create a helper method
- > Lets get the absolute path for our jar file

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

- > So we have a Dynamo Table, Four Lambda functions and Api gateway
- > Lets start with Dynamo Table
- > Lets have a look at our Lambda code where we access this variables
- > Lets write all four of our Lambda functions handler name
- > Now create Lambda Function objects with their properties
- > Lets create a helper method
- > Lets get the absolute path for our jar file

```

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE
29     Attribute partitionKey = Attribute.builder()
30         .name("id")
31         .type(AttributeType.NUMBER)
32         .build();
33
34     TableProps tableProps = TableProps.builder()
35         .tableName("products")
36         .partitionKey(partitionKey)
37         .removalPolicy(RemovalPolicy.DESTROY)
38         // this will remove the table when we run cdk destroy
39         .build();
40
41     Table dynamoTable = new Table(this,"products", tableProps);
42
43     // Lets create two environment variable that can be accessible from our lambda code
44
45     Map<String, String> lambdaEnvMap = new HashMap<>();
46     lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
47     lambdaEnvMap.put("PRIMARY_KEY", "id");
48
49     // Lambda code
50
51     String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
52     String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
53     String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
54     String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
55
56     Function createFunction = new Function(this, "createProductFunction", null,
57
58         .code(Code.fromAsset("/Users/mdomor.farouque/eclipse-workspace-3/productApi/target/productApi-0.1.jar"))
59         .handler(handler)
60         .runtime(Runtime.JAVA_8)
61         .environment(lambdaEnvMap));
62
63     private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
64         return FunctionProps.builder()
65             .code(Code.fromAsset("/Users/mdomor.farouque/eclipse-workspace-3/productApi/target/productApi-0.1.jar"))
66             .handler(handler)
67             .runtime(Runtime.JAVA_8)
68             .environment(lambdaEnvMap)
69             .timeout(Duration.seconds(15))
70             .memorySize(512)
71             .build();
72
73     }
74
75     I
76
77     I
78
79     I
80

```

```

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE
28     // Table code
29
30     Attribute partitionKey = Attribute.builder()
31         .name("id")
32         .type(AttributeType.NUMBER)
33         .build();
34
35     TableProps tableProps = TableProps.builder()
36         .tableName("products")
37         .partitionKey(partitionKey)
38         .removalPolicy(RemovalPolicy.DESTROY)
39         // this will remove the table when we run cdk destroy
40         .build();
41
42     Table dynamoTable = new Table(this,"products", tableProps);
43
44     // Lets create two environment variable that can be accessible from our lambda code
45
46     Map<String, String> lambdaEnvMap = new HashMap<>();
47     lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
48     lambdaEnvMap.put("PRIMARY_KEY", "id");
49
50     // Lambda code
51
52     String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
53     String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
54     String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
55     String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
56
57     Function createFunction = new Function(this, "createProductFunction",
58
59         .getLambdaFunctionProps(lambdaEnvMap, createFunctionHandlerName));
60
61     I
62
63     I
64
65     I
66
67     private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
68         return FunctionProps.builder()
69             .code(Code.fromAsset("/Users/mdomor.farouque/eclipse-workspace-3/productApi/target/productApi-0.1.jar"))
70             .handler(handler)
71             .runtime(Runtime.JAVA_8)
72             .environment(lambdaEnvMap)
73             .timeout(Duration.seconds(15))
74             .memorySize(512)
75             .build();
76
77     }
78
79     I
80

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

```

3. Lets set our account and region. If a
specific account and region is given, we
will set our app to that given
environment(account/region) or else we will
use the default account and region

CDK_DEFAULT_ACCOUNT and
CDK_DEFAULT_REGION environment variables in
our CDK code resolve to the default account
and region of our AWS CLI profile

4. Next, lets write our Infrastructure code
to create AWS resources

> So we have a Dynamo Table, Four Lambda
functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code
where we access this variables

> Lets write all four of our Lambda
functions handler name

> Now create Lambda Function objects
with their properties

> Lets create a helper method

> Lets get the absolute path for our jar
file

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE
1 package com.myorg;
2
3 import software.constructs.Construct;
4
5 import java.util.HashMap;
6 import java.util.Map;
7
8 import software.amazon.awscdk.Duration;
9 import software.amazon.awscdk.RemovalPolicy;
10 import software.amazon.awscdk.Stack;
11 import software.amazon.awscdk.StackProps;
12 import software.amazon.awscdk.services.cloudfront.Function;
13 import software.amazon.awscdk.services.dynamodb.Attribute;
14 import software.amazon.awscdk.services.dynamodb.AttributeType;
15 import software.amazon.awscdk.services.dynamodb.Table;
16 import software.amazon.awscdk.services.dynamodb.TableProps;
17 import software.amazon.awscdk.services.lambda.Code;
18 import software.amazon.awscdk.services.lambda.FunctionProps;
19 import software.amazon.awscdk.services.lambda.Runtime;
20
21 public class ProductApiCdkStack extends Stack {
22     public ProductApiCdkStack(final Construct scope, final String id) {
23         this(scope, id, null);
24     }
25
26     public ProductApiCdkStack(final Construct scope, final String id, final StackProps props) {
27         super(scope, id, props);
28
29         // Table code
30
31         Attribute partitionKey = Attribute.builder()
32             .name("id")
33             .type(AttributeType.NUMBER)
34             .build();
35
36         TableProps tableProps = TableProps.builder()
37             .tableName("products")
38             .partitionKey(partitionKey)
39             .removalPolicy(RemovalPolicy.DESTROY)
40             // this will remove the table when we run cdk destroy
41             .build();
42
43         Table dynamoTable = new Table(this, "products", tableProps);
44
45         // Lets create two environment variable that can be accessible from our lambda code
46
47         Map<String, String> lambdaEnvMap = new HashMap<>();
48         lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
49         lambdaEnvMap.put("PRIMARY_KEY", "id");
50
51         // Lambda code
52
53         String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
54
55         Function createFunction = new Function(this, "createProductFunction",
56             getLambdaFunctionProps(lambdaEnvMap, createFunctionHandlerName));
57
58         String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
59         Function getFunction = new Function(this, "getSingleProductFunction",
60             getLambdaFunctionProps(lambdaEnvMap, getFunctionHandlerName));
61
62         String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
63         Function updateFunction = new Function(this, "updateProductFunction",
64             getLambdaFunctionProps(lambdaEnvMap, updateFunctionHandlerName));
65
66         String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
67         Function deleteFunction = new Function(this, "deleteProductFunction",
68             getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
69
70         FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
71             return FunctionProps.builder()
72                 .code(Code.fromAsset("/Users/mdomor.farouque/eclipse-workspace-3/productApi/target/productApi-0.1.j
73                 .handler(handler)
74                 .environment(lambdaEnvMap)
75                 .timeout(Duration.seconds(15))
76                 .memorySize(512)
77                 .build();
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

```

3. Lets set our account and region. If a
specific account and region is given, we
will set our app to that given
environment(account/region) or else we will
use the default account and region

CDK_DEFAULT_ACCOUNT and
CDK_DEFAULT_REGION environment variables in
our CDK code resolve to the default account
and region of our AWS CLI profile

4. Next, lets write our Infrastructure code
to create AWS resources

> So we have a Dynamo Table, Four Lambda
functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code
where we access this variables

> Lets write all four of our Lambda
functions handler name

> Now create Lambda Function objects
with their properties

> Lets create a helper method

> Lets get the absolute path for our jar
file

eclipse-workspace-3 - productApiCDK/src/main/java/com/myorg/ProductApiCdkStack.java - Eclipse IDE
1 package com.myorg;
2
3 import software.constructs.Construct;
4
5 import java.util.HashMap;
6 import java.util.Map;
7
8 import software.amazon.awscdk.Duration;
9 import software.amazon.awscdk.RemovalPolicy;
10 import software.amazon.awscdk.Stack;
11 import software.amazon.awscdk.StackProps;
12 import software.amazon.awscdk.services.cloudfront.Function;
13 import software.amazon.awscdk.services.dynamodb.Attribute;
14 import software.amazon.awscdk.services.dynamodb.AttributeType;
15 import software.amazon.awscdk.services.dynamodb.Table;
16 import software.amazon.awscdk.services.dynamodb.TableProps;
17 import software.amazon.awscdk.services.lambda.Code;
18 import software.amazon.awscdk.services.lambda.FunctionProps;
19 import software.amazon.awscdk.services.lambda.Runtime;
20
21 public class ProductApiCdkStack extends Stack {
22     public ProductApiCdkStack(final Construct scope, final String id) {
23         super(scope, id, null);
24
25         // Table code
26
27         Attribute partitionKey = Attribute.builder()
28             .name("id")
29             .type(AttributeType.NUMBER)
30             .build();
31
32         TableProps tableProps = TableProps.builder()
33             .tableName("products")
34             .partitionKey(partitionKey)
35             .removalPolicy(RemovalPolicy.DESTROY)
36             // this will remove the table when we run cdk destroy
37             .build();
38
39         Table dynamoTable = new Table(this, "products", tableProps);
40
41         // Lets create two environment variable that can be accessible from our lambda code
42
43         Map<String, String> lambdaEnvMap = new HashMap<>();
44         lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
45         lambdaEnvMap.put("PRIMARY_KEY", "id");
46
47         // Lambda code
48
49         String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
50
51         Function createFunction = new Function(this, "createProductFunction",
52             getLambdaFunctionProps(lambdaEnvMap, createFunctionHandlerName));
53
54         String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
55         Function getFunction = new Function(this, "getSingleProductFunction",
56             getLambdaFunctionProps(lambdaEnvMap, getFunctionHandlerName));
57
58         String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
59         Function updateFunction = new Function(this, "updateProductFunction",
60             getLambdaFunctionProps(lambdaEnvMap, updateFunctionHandlerName));
61
62         String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
63         Function deleteFunction = new Function(this, "deleteProductFunction",
64             getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
65
66         FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
67             return FunctionProps.builder()
68                 .code(Code.fromAsset("/Users/mdomor.farouque/eclipse-workspace-3/productApi/target/productApi-0.1.j
69                 .handler(handler)
70                 .environment(lambdaEnvMap)
71                 .timeout(Duration.seconds(15))
72                 .memorySize(512)
73                 .build();
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

> Now lets grant this function read-write permission to dynamoDB table

```

35     TableProps tableProps = TableProps.builder()
36         .tableName("products")
37         .partitionKey(partitionKey)
38         .removalPolicy(RemovalPolicy.DESTROY)
39         // this will remove the table when we run cdk destroy
40         .build();
41
42     Table dynamoTable = new Table(this,"products", tableProps);
43
44     // Lets create two environment variable that can be accessible from our lambda code
45     Map<String, String> lambdaEnvMap = new HashMap<>();
46     lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
47     lambdaEnvMap.put("PRIMARY_KEY", "id");
48
49     // Lambda code
50
51     String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
52     String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
53     String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
54     String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
55
56     Function createFunction = new Function(this, "createProductFunction",
57         getLambdaFunctionProps(lambdaEnvMap, createFunctionHandlerName));
58
59     Function getFunction = new Function(this, "getProductFunction",
60         getLambdaFunctionProps(lambdaEnvMap, getFunctionHandlerName));
61
62     Function updateFunction = new Function(this, "updateProductFunction",
63         getLambdaFunctionProps(lambdaEnvMap, updateFunctionHandlerName));
64
65     Function deleteFunction = new Function(this, "deleteProductFunction",
66         getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
67
68     }
69
70     private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
71         return FunctionProps.builder()
72             .code(Code.fromAsset("/Users/mdmor.faruque/eclipse-workspace-3/productApi/target/productApi-0.1-SNAPSHOT.jar"))
73             .handler(handler)
74             .runtime(Runtime.JAVA_8)
75             .environment(lambdaEnvMap)
76             .timeout(Duration.seconds(15))
77             .memorySize(512)
78             .build();
79     }
80
81     Map<String, String> lambdaEnvMap = new HashMap<>();
82     lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
83     lambdaEnvMap.put("PRIMARY_KEY", "id");
84
85     // Lambda code
86
87     String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
88     String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
89     String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
90     String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
91
92     Function createFunction = new Function(this, "createProductFunction",
93         getLambdaFunctionProps(lambdaEnvMap, createFunctionHandlerName));
94
95     Function getFunction = new Function(this, "getProductFunction",
96         getLambdaFunctionProps(lambdaEnvMap, getFunctionHandlerName));
97
98     Function updateFunction = new Function(this, "updateProductFunction",
99         getLambdaFunctionProps(lambdaEnvMap, updateFunctionHandlerName));
100
101    Function deleteFunction = new Function(this, "deleteProductFunction",
102        getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
103
104    dynamoTable.grantReadWriteData(createFunction);
105    dynamoTable.grantReadWriteData(getFunction);
106    dynamoTable.grantReadWriteData(updateFunction);
107    dynamoTable.grantReadWriteData(deleteFunction);
108
109    }
110
111    private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
112        return FunctionProps.builder()
113            .code(Code.fromAsset("/Users/mdmor.faruque/eclipse-workspace-3/productApi/target/productApi-0.1-SNAPSHOT.jar"))
114            .handler(handler)
115            .runtime(Runtime.JAVA_8)
116            .environment(lambdaEnvMap)
117            .timeout(Duration.seconds(15))
118            .memorySize(512)
119            .build();
120    }

```

3. Lets set our account and region. If a specific account and region is given, we will set our app to that given environment(account/region) or else we will use the default account and region

CDK_DEFAULT_ACCOUNT and CDK_DEFAULT_REGION environment variables in our CDK code resolve to the default account and region of our AWS CLI profile

4. Next, lets write our Infrastructure code to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

> Now lets grant this function read-write permission to dynamoDB table

```

47     Map<String, String> lambdaEnvMap = new HashMap<>();
48     lambdaEnvMap.put("TABLE_NAME", dynamoTable.getTableName());
49     lambdaEnvMap.put("PRIMARY_KEY", "id");
50
51     // Lambda code
52
53     String createFunctionHandlerName = "com.pwm.aws.crud.product.api.CreateProduct";
54     String getFunctionHandlerName = "com.pwm.aws.crud.product.api.GetSingleProduct";
55     String updateFunctionHandlerName = "com.pwm.aws.crud.product.api.UpdateProduct";
56     String deleteFunctionHandlerName = "com.pwm.aws.crud.product.api.DeleteProduct";
57
58     Function createFunction = new Function(this, "createProductFunction",
59         getLambdaFunctionProps(lambdaEnvMap, createFunctionHandlerName));
60
61     Function getFunction = new Function(this, "getProductFunction",
62         getLambdaFunctionProps(lambdaEnvMap, getFunctionHandlerName));
63
64     Function updateFunction = new Function(this, "updateProductFunction",
65         getLambdaFunctionProps(lambdaEnvMap, updateFunctionHandlerName));
66
67     Function deleteFunction = new Function(this, "deleteProductFunction",
68         getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
69
70     dynamoTable.grantReadWriteData(createFunction);
71     dynamoTable.grantReadWriteData(getFunction);
72     dynamoTable.grantReadWriteData(updateFunction);
73     dynamoTable.grantReadWriteData(deleteFunction);
74
75     }
76
77     // REST API code
78
79
80     }
81
82     private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
83         return FunctionProps.builder()
84             .code(Code.fromAsset("/Users/mdmor.faruque/eclipse-workspace-3/productApi/target/productApi-0.1-SNAPSHOT.jar"))
85             .handler(handler)
86             .runtime(Runtime.JAVA_8)
87             .environment(lambdaEnvMap)
88             .timeout(Duration.seconds(15))
89             .memorySize(512)
90             .build();
91     }
92
93     }
94
95     }
96
97     }
98
99     }

```

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

> Now lets grant this function read-write permission to dynamoDB table

```

    71  getLambdaFunctionProps(lambdaEnvMap, updateFunctionHandlerName);
    72
    73  Function deleteFunction = new Function(this, "deleteProductFunction",
    74      getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
    75
    76  dynamoTable.grantReadWriteData(createFunction);
    77  dynamoTable.grantReadWriteData(getFunction);
    78  dynamoTable.grantReadWriteData(updateFunction);
    79  dynamoTable.grantReadWriteData(deleteFunction);
    80
    81
    82
    83
    84
    85
    86
    87
    88
    89
    90
    91
    92
    93
    94
    95
    96
    97
    98
    99
    100
    101
    102
    103
    104
    105
    106
    107
    108
    109
    110
    111
    112  private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
    113      return FunctionProps.builder()
    114          .code(Code.fromAsset("/Users/mdomor.faruque/eclipse-workspace-3/productApi/target/productApi-"
    115              ".jar"))
    116          .handler(handler)
    117          .runtime(Runtime.JAVA_8)
    118          .environment(lambdaEnvMap)
    119          .timeout(Duration.seconds(15))
    120          .memorySize(512)
    121          .build();
    122
    123
  
```

to create AWS resources

> So we have a Dynamo Table, Four Lambda functions and Api gateway

> Lets start with Dynamo Table

> Lets have a look at our Lambda code where we access this variables

> Lets write all four of our Lambda functions handler name

> Now create Lambda Function objects with their properties

> Lets create a helper method

> Lets get the absolute path for our jar file

> Now lets grant this function read-write permission to dynamoDB table

> We are done with our infrastructure code

5. Now RUN > \$ cdk synth

```

    71  getLambdaFunctionProps(lambdaEnvMap, updateFunctionHandlerName);
    72
    73  Function deleteFunction = new Function(this, "deleteProductFunction",
    74      getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
    75
    76  dynamoTable.grantReadWriteData(createFunction);
    77  dynamoTable.grantReadWriteData(getFunction);
    78  dynamoTable.grantReadWriteData(updateFunction);
    79  dynamoTable.grantReadWriteData(deleteFunction);
    80
    81
    82
    83
    84
    85
    86
    87
    88
    89
    90
    91
    92
    93
    94
    95
    96
    97
    98
    99
    100
    101
    102
    103
    104
    105
    106
    107
    108
    109
    110
    111
    112
    113
    114
    115
    116
    117
    118  private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String handler) {
    119      return FunctionProps.builder()
    120          .code(Code.fromAsset("/Users/mdomor.faruque/eclipse-workspace-3/productApi/target/productApi-"
    121              ".jar"))
    122          .handler(handler)
    123          .runtime(Runtime.JAVA_8)
    124          .environment(lambdaEnvMap)
  
```

aws-cdk-java — Edited

to create AWS resources

- > So we have a Dynamo Table, Four Lambda functions and Api gateway
- > Lets start with Dynamo Table
- > Lets have a look at our Lambda code where we access this variables
- > Lets write all four of our Lambda functions handler name
- > Now create Lambda Function objects with their properties
- > Lets create a helper method
- > Lets get the absolute path for our jar file
- > Now lets grant this function read-write permission to dynamoDB table
- > We are done with our infrastructure code

5. Now RUN > \$ cdk synth

```

    ...
    private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String functionName) {
        Function deleteFunction = new Function(this, "deleteProductFunction", getLambdaFunctionProps(lambdaEnvMap, deleteFunctionHandlerName));
        ...
    }

    dynamoTable.grantReadWriteData(createFunction);
    dynamoTable.grantReadWriteData(getFunction);
    dynamoTable.grantReadWriteData(updateFunction);
    dynamoTable.grantReadWriteData(deleteFunction);

    // REST API code
    RestApiProps restApiProps = RestApiProps.builder()
        .restApiName("Product Service")
        .build();

    RestApi api = new RestApi(this, "productsApi", restApiProps);
    // lets add resource path to our api
    IResource productsResource = api.getRoot().addResource("products");
    // lambda integration for this path
    Integration createIntegration = new LambdaIntegration(createFunction);
    productsResource.addMethod("POST", createIntegration);

    // Lets add {id} resource under the "products" resource
    IResource singleProductResource = productsResource.addResource("{id}");

    Integration getIntegration = new LambdaIntegration(getFunction);
    singleProductResource.addMethod("GET", getIntegration);

    Integration updateIntegration = new LambdaIntegration(updateFunction);
    singleProductResource.addMethod("PATCH", updateIntegration);

    Integration deleteIntegration = new LambdaIntegration(deleteFunction);
    singleProductResource.addMethod("DELETE", deleteIntegration);

}

private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String functionName) {
    return FunctionProps.builder()
        .code(Code.fromAsset("/Users/mdmor.farouque/eclipse-workspace-3/productApiCDK"))
        .handler(handler)
        .runtime(Runtime.JAVA_8)
        .environment(lambdaEnvMap)
}

```

> Lets write all four of our Lambda functions handler name

- > Now create Lambda Function objects with their properties
- > Lets create a helper method
- > Lets get the absolute path for our jar file
- > Now lets grant this function read-write permission to dynamoDB table
- > We are done with our infrastructure code

5. Now RUN > \$ cdk synth

> We are done with our infrastructure code

5. Now RUN > \$ cdk synth

Note : this command will print the CloudFormation template for our stack

6. Now RUN > cdk bootstrap

Note : This command will create resources required by the cdk to perform deployments in a given environment(account/region)

We only need to use the bootstrap command once for every environment (account/region). If we use the command more than once, the cdk will check if our CDKToolkit stack has to be updated. if required the stack will be updated. If not, running the bootstrap command does nothing.

before we run the command lets check our s3 bucket and CloudFormation Stacks

```

ps = RestApiProps.builder()
    .restApiName("Product Service")

    pi(this, "productsApi", restApiProps);
    th to our api
    rce = api.getRoot().addResource("products");
    or this path
    ration = new LambdaIntegration(createFunction);
    hod("POST", createIntegration);

    ce under the "products" resource
    Resource = productsResource.addResource("{id}");

    ion = new LambdaIntegration(getFunction);
    ddMethod("GET", getIntegration);

    Integration updateIntegration = new LambdaIntegration(updateFunction);
    singleProductResource.addMethod("PATCH", updateIntegration);

    Integration deleteIntegration = new LambdaIntegration(deleteFunction);
    singleProductResource.addMethod("DELETE", deleteIntegration);

}

private FunctionProps getLambdaFunctionProps(Map<String, String> lambdaEnvMap, String functionName) {
    return FunctionProps.builder()
        .code(Code.fromAsset("/Users/mdmor.farouque/eclipse-workspace-3/productApiCDK"))
        .handler(handler)
        .runtime(Runtime.JAVA_8)
        .environment(lambdaEnvMap)
}

```

```

Parameters:
  BootstrapVersion:
    Type: AWS::SSM::Parameter::Value[String]
    Default: /cdk-bootstrap/hnb659fds/version
    Description: Version of the CDK Bootstrap resources in this environment, automatically retrieved from SSM Parameter Store. [cdk:skip]
  Rules:
    CheckBootstrapVersion:
      Assertions:
        - Assert:
          Fn::Not:
            - Fn::Contains:
              - "1"
              - "2"
              - "3"
              - "4"
              - "5"
            - Ref: BootstrapVersion
      AssertDescription: CDK bootstrap stack version 6 required. Please run 'cdk bootstrap' with a recent version of the CDK CLI.

MRMIOMP2830:productApiCDK mdmor.farouque$ 

```

```

> Now create Lambda Function objects
with their properties

> Lets create a helper method

> Lets get the absolute path for our jar
file

> Now lets grant this function read-
write permission to dynamoDB table

> We are done with our infrastructure
code

5. Now RUN > $ cdk synth

Note : this command will print the
CloudFormation template for our stack

6. Now RUN > cdk bootstrap

Note : This command will create
resources required by the cdk to perform
deployments in a given environment(account/
region)

We only need to use the bootstrap

```

Console Home [Info](#)

Introducing the new widget Latest announcements. Find it at the bottom of your Console Home.

Recently visited [Info](#)

- CloudWatch
- Lambda
- API Gateway
- CloudFormation
- S3
- DynamoDB
- Simple Notification Service
- IAM

```

aws-cdk-java — Edited
write permission to dynamoDB table

> We are done with our infrastructure
code

5. Now RUN > $ cdk synth

Note : this command will print the
CloudFormation template for our stack

6. Now RUN > cdk bootstrap

Note : This command will create
resources required by the cdk to perform
deployments in a given environment(account/
region)

We only need to use the bootstrap
command once for every environment (account/
region). If we use the command more than
once, the cdk will check if our CDKToolkit
stack has to be updated. if required the
stack will be updated. If not, running the
bootstrap command does nothing.

before we run the command lets check
our s3 bucket and CloudFormation Stacks

You can see we have 3 buckets before
running the bootstrap command

Now lets check cloudformation stack

```

Amazon S3 [X](#)

Are you missing easy ways to reduce storage costs and enhance data protection? [Find out with S3 Storage Lens](#)

Buckets

- Access Points
- Object Lambda Access Points
- Multi-Region Access Points
- Batch Operations
- Access analyzer for S3

Block Public Access settings for this account

Storage Lens

- Dashboards
- AWS Organizations settings

Feature spotlight [3](#)

AWS Marketplace for S3

Buckets (3) [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

Name	AWS Region	Access	Creation date
com.omor.my-lambda-functions	Canada (Central) ca-central-1	Bucket and objects not public	April 10, 2022, 01:01:05 (UTC-04:00)
elasticbeanstalk-us-east- ... blur	US East (N. Virginia) us-east-1	Objects can be public	February 1, 2019, 11:11:16 (UTC-05:00)
elasticbeanstalk-us-east- ... blur	US East (Ohio) us-east-2	Objects can be public	July 18, 2018, 22:16:28 (UTC-04:00)

```
aws-cdk-java — Edited
write permission to dynamoDB table
```

```
> We are done with our infrastructure
code
```

5. Now RUN > \$ cdk synth

Note : this command will print the CloudFormation template for our stack

6. Now RUN > cdk bootstrap

Note : This command will create resources required by the cdk to perform deployments in a given environment(account/region)

We only need to use the bootstrap command once for every environment (account/region). If we use the command more than once, the cdk will check if our CDKToolkit stack has to be updated. if required the stack will be updated. If not, running the bootstrap command does nothing.

before we run the command lets check our s3 bucket and CloudFormation Stacks

You can see we have 3 buckets before running the bootstrap command

Now lets check cloudformation stack

```
aws-cdk-java — Edited
command once for every environment (account/
region). If we use the command more than
once, the cdk will check if our CDKToolkit
stack has to be updated. if required the
stack will be updated. If not, running the
bootstrap command does nothing.
```

before we run the command lets check our s3 bucket and CloudFormation Stacks

You can see we have 3 buckets before running the bootstrap command

Now lets check cloudformation stack

No stack before the command

Now lets run the command

I am running the command .. because of the security region I am not recording that

command is running

The screenshot shows the AWS CloudFormation Management Console. On the left, a sidebar menu includes 'Stacks' (which is selected), 'StackSets', 'Exports', 'Designer', 'Registry' (with sub-options 'Public extensions', 'Activated extensions', 'Publisher'), and 'Feedback'. The main content area features a large 'AWS CloudFormation' logo with the tagline 'Model and provision all your cloud infrastructure'. Below the logo, a text box states: 'AWS CloudFormation provides a common language to describe and provision all the infrastructure resources in your environment in a safe, repeatable way.' To the right, a 'Create a CloudFormation stack' button is visible. A 'Getting started' sidebar on the right lists links: 'What is AWS CloudFormation', 'Getting started with CloudFormation', 'Learn template basics', and 'Quick start'.

The screenshot shows the 'CloudFormation > Stacks' page. The sidebar on the left is identical to the previous screenshot. The main content area displays a table titled 'Stacks (0)' with columns for 'Stack name', 'Status', and 'Created time'. A message at the bottom left says 'No stacks' and 'No stacks to display'. At the bottom right, there is a 'Create stack' button and a 'View getting started guide' link.

```
aws-cdk-java -- Edited
command once for every environment (account/
region). If we use the command more than
once, the cdk will check if our CDKToolkit
stack has to be updated. if required the
stack will be updated. If not, running the
bootstrap command does nothing.
```

before we run the command lets check
our s3 bucket and CloudFormation Stacks

You can see we have 3 buckets before
running the bootstrap command

Now lets check cloudformation stack

No stack before the command

Now lets run the command

I am running the command .. because of
the security region I am not recording that

command is running

command completed

As you can see a new stack was created
by the bootstrap command |

The screenshot shows the AWS CloudFormation console with the 'Stacks' tab selected. A single stack named 'CDKToolkit' is listed in the table. The status column shows 'CREATE_IN_PROGRESS' and the created time is '2022-06-08 13:18:08 UTC-07:00'. The table has columns for 'Stack name', 'Status', and 'Created time'.

Stack name	Status	Created time
CDKToolkit	CREATE_IN_PROGRESS	2022-06-08 13:18:08 UTC-07:00

```
aws-cdk-java -- Edited
command once for every environment (account/
region). If we use the command more than
once, the cdk will check if our CDKToolkit
stack has to be updated. if required the
stack will be updated. If not, running the
bootstrap command does nothing.
```

before we run the command lets check
our s3 bucket and CloudFormation Stacks

You can see we have 3 buckets before
running the bootstrap command

Now lets check cloudformation stack

No stack before the command

Now lets run the command

I am running the command .. because of
the security region I am not recording that

command is running

command completed

As you can see a new stack was created
by the bootstrap command

lets check our s3 bucket

The screenshot shows the AWS CloudFormation console with the 'Stacks' tab selected. The same stack 'CDKToolkit' is listed in the table, but its status has changed to 'CREATE_COMPLETE'. The created time remains the same as in the previous screenshot.

Stack name	Status	Created time
CDKToolkit	CREATE_COMPLETE	2022-06-08 13:18:08 UTC-07:00

command once for every environment (account/region). If we use the command more than once, the cdk will check if our CDKToolkit stack has to be updated. If required the stack will be updated. If not, running the bootstrap command does nothing.

before we run the command lets check our s3 bucket and CloudFormation Stacks

You can see we have 3 buckets before running the bootstrap command

Now lets check cloudformation stack

No stack before the command

Now lets run the command

I am running the command .. because of the security region I am not recording that

command is running

command completed

As you can see a new stack was created by the bootstrap command

lets check our s3 bucket

You can see a new bucket was created by that bootstrap command.

This are important for our cdk to successfully create and deploy our cloud infrastructure. |

Name	AWS Region	Access	Creation date
cdk-hnb659fds-assets-123456789-ca-central-1	Canada (Central) ca-central-1	Bucket and objects not public	June 8, 2022, 13:18:20 (UTC-04:00)
com.omor.my-lambda-functions	Canada (Central) ca-central-1	Bucket and objects not public	April 10, 2022, 01:01:05 (UTC-04:00)
Elasticbeanstalk-us-east-1-123456789	US East (N. Virginia) us-east-1	Objects can be public	February 1, 2019, 11:11:16 (UTC-05:00)
Elasticbeanstalk-us-east-2-123456789	US East (Ohio) us-east-2	Objects can be public	July 18, 2018, 22:16:28 (UTC-04:00)

aws-cdk-java — Edited
command completed

As you can see a new stack was created by the bootstrap command

lets check our s3 bucket

You can see a new bucket was created by that bootstrap command.

This are important for our cdk to successfully create and deploy our cloud infrastructure.

7. Now run > \$ mvn package

> it should say BUILD SUCCESS

8. Now RUN > cdk deploy

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

> No api gateway

Streamline API development
Amazon API Gateway lets you simultaneously run multiple versions and release stages of the same API, allowing you to quickly iterate, test, and release new versions.

Performance at scale
Amazon API Gateway helps you improve performance by managing traffic to your existing back-end systems, throttling API call spikes, and enabling result caching.

SDK generation
Amazon API Gateway can generate client SDKs for JavaScript, iOS, and Android, which you can use to quickly test new APIs from your applications and distribute SDKs to third-party developers.

Streamline API development
Amazon API Gateway lets you simultaneously run multiple versions and release stages of the same API, allowing you to quickly iterate, test, and release new versions.

Performance at scale
Amazon API Gateway helps you improve performance by managing traffic to your existing back-end systems, throttling API call spikes, and enabling result caching.

SDK generation
Amazon API Gateway can generate client SDKs for JavaScript, iOS, and Android, which you can use to quickly test new APIs from your applications and distribute SDKs to third-party developers.

```

aws-cdk-java — Edited
command completed

As you can see a new stack was created
by the bootstrap command

lets check our s3 bucket

You can see a new bucket was created by
that bootstrap command.

This are important for our cdk to
successfully create and deploy our cloud
infrastructure.

7. Now run > $ mvn package
> it should say BUILD SUCCESS

8. Now RUN > cdk deploy

This command will create all the
functions, dynamo table and api gateway

Lets check before running the command

> No api gateway
> No lambda function

```

The screenshot shows the AWS Lambda Functions console. The left sidebar has 'AWS Lambda' selected under 'Services'. The main area is titled 'Functions (0)' and displays the message 'There is no data to display.'

```

This command will create all the
functions, dynamo table and api gateway

Lets check before running the command

> No api gateway
> No lambda function
> No Dyanmo Table
before running

now lets run the command.

[$cdk deploy] command running...

it will ask Do you wish to deploy these
changes (y/n)?
Type Y and enter

```

The screenshot shows the AWS DynamoDB Tables console. The left sidebar has 'DynamoDB' selected under 'Services'. The main area is titled 'Tables (0) Info' and displays the message 'No tables found' and 'We cannot find a match.'

```

This command will create all the
functions, dynamo table and api gateway

Lets check before running the command

> No api gateway
> No lambda function
> No Dyanmo Table
before running

now lets run the command.

[$cdk deploy] command running...

it will ask Do you wish to deploy these
changes (y/n)?
Type Y and enter

In progress.....
Done
Lets check

```

The screenshot shows the AWS DynamoDB Tables console. The left sidebar has 'DynamoDB' selected under 'Services'. The main area is titled 'Tables (1) Info' and shows a table named 'products' with the following details:

Name	Status	Partition key	Sort key	Indexes	Read capacity mode
products	Active	id (N)	-	0	Provisioned (5)

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

```
> No api gateway
> No lambda function
> No Dyanmo Table
```

before running

now lets run the command.

[\$cdk deploy] command running...

it will ask Do you wish to deploy these changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Function name	Description	Package type	Runtime	Last modified
ProductApiCdkStack-createProductFunctionB839AA42-QzOwBelYTt80	-	Zip	Java 8 on Amazon Linux 1	38 sec ago
ProductApiCdkStack-updateProductFunction38707609-rLZ4usf7ODkE	-	Zip	Java 8 on Amazon Linux 1	38 sec ago
ProductApiCdkStack-getProductFunction18486D68-nor9mlpgw30l	-	Zip	Java 8 on Amazon Linux 1	38 sec ago
ProductApiCdkStack-deleteProductFunction3F2D3938-P6PRzBcgWGCA	-	Zip	Java 8 on Amazon Linux 1	38 sec ago

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

```
> No api gateway
> No lambda function
> No Dyanmo Table
```

before running

now lets run the command.

[\$cdk deploy] command running...

it will ask Do you wish to deploy these changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Name	Description	ID	Protocol	Endpoint type	Created
Product Service	juksww0qlid	REST	Edge	2022-06-01	

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

```
> No api gateway
> No lambda function
> No Dyanmo Table
```

before running

now lets run the command.

[\$cdk deploy] command running...

it will ask Do you wish to deploy these changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

```

aws-cdk-java — Edited

This command will create all the
functions, dynamo table and api gateway

Lets check before running the command

> No api gateway
> No lambda function
> No Dynamo Table
before running

now lets run the command.

[$cdk deploy] command running...

it will ask Do you wish to deploy these
changes (y/n)?
Type Y and enter

In progress.....
Done
Lets check
Lets test our api from postman

```

The screenshot shows the AWS Lambda console with the URL 'ca-central-1.console.aws.amazon.com'. The main navigation bar includes 'S3 Management Console', 'API Gateway', 'Functions - Lambda', 'CloudFormation', 'DynamoDB', and 'CloudWatch'. Below the navigation, it says 'APIs > Product Service (juksww0qjd) > Stages'. A sidebar on the left lists 'APIs', 'Custom Domain Names', and 'VPC Links'. Under 'API: Product Service', there are links for 'Resources', 'Stages' (which is bolded), 'Authorizers', 'Gateway Responses', 'Models', 'Resource Policy', and 'Documentation'. A 'Create' button is at the top right of the stages section. To the right, a message says 'Select a stage' with a dropdown menu showing 'prod'.

```

aws-cdk-java — Edited

This command will create all the
functions, dynamo table and api gateway

Lets check before running the command

> No api gateway
> No lambda function
> No Dynamo Table
before running

now lets run the command.

[$cdk deploy] command running...

it will ask Do you wish to deploy these
changes (y/n)?
Type Y and enter

In progress.....
Done
Lets check
Lets test our api from postman

```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Scratch Pad', 'Collections', 'APIs' (which is bolded), 'Environments', 'Mock Servers', 'Monitors', and 'History'. A red callout box points to the 'APIs' icon with the text 'Switch to a workspace to use this feature'. Another red callout box points to a 'Switch to Workspaces' button. The main area shows a 'Untitled Request' with a 'GET' method selected. Below it are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. A 'Send' button is at the top right. The URL field is empty.

```

aws-cdk-java — Edited

This command will create all the
functions, dynamo table and api gateway

Lets check before running the command

> No api gateway
> No lambda function
> No Dynamo Table
before running

now lets run the command.

[$cdk deploy] command running...

it will ask Do you wish to deploy these
changes (y/n)?
Type Y and enter

In progress.....
Done
Lets check
Lets test our api from postman

```

The screenshot shows the AWS Lambda console with the URL 'ca-central-1.console.aws.amazon.com'. The main navigation bar and sidebar are identical to the previous screenshot. Under 'API: Product Service', the 'Stages' link is bolded. The 'prod' stage is expanded, showing a tree structure for the '/' endpoint. Underneath '/' are 'POST', 'PUT', 'DELETE', 'GET', and 'PATCH' methods. The 'PATCH' method is highlighted with a blue background. To the right, a message says 'Select a stage' with a dropdown menu showing 'prod'.

This command will create all the functions, dynamo table and api gateway before running

Lets check before running the command.

```
> No api gateway
> No lambda function
> No Dynamo Table
```

now lets run the command.

```
[$cdk deploy] command running...
```

it will ask Do you wish to deploy changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

The screenshot shows the AWS CloudFormation console with the Product Service stack selected. Under the Stages section, the prod stage is selected. The POST method for the /products endpoint is shown with its details: id, name, and price. An Invoke URL is provided: https://juksww0qid.execute-api.ca-central-1.amazonaws.com/prod/products.

This command will create all the functions, dynamo table and api gateway before running

Lets check before running the command.

```
> No api gateway
> No lambda function
> No Dynamo Table
```

now lets run the command.

```
[$cdk deploy] command running...
```

it will ask Do you wish to deploy changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

The screenshot shows a Postman collection named "aws-cdk-jav...". A POST request is being prepared to the URL https://juksww0qid.execute-api.ca-central-1.amazonaws.com/prod/products. The Body tab is selected, showing a JSON payload:

```
1 {
2   "id":1,
3   "name":"Random product 1",
4   "price":12.34
5 }
```

This command will create all the functions, dynamo table and api gateway before running

Lets check before running the command.

```
> No api gateway
> No lambda function
> No Dynamo Table
```

now lets run the command.

```
[$cdk deploy] command running...
```

it will ask Do you wish to deploy changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

The screenshot shows the AWS DynamoDB console with a single table named "products". The table has one item with the primary key "id" and value "1". The item also contains "name" and "price" fields.

Name	Status	Partition key	Sort key	Indexes	Read capacity mode
products	Active	id (N)	-	0	Provisioned (5)

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

- > No api gateway
- > No lambda function
- > No Dyanmo Table

before running

now lets run the command.

[\${cdk deploy} command running..

it will ask Do you wish to deploy changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

The screenshot shows the AWS CloudFormation console with the 'Products' table selected in the 'Tables' list. The table details are visible, including partition key 'id (Number)', sort key '-', capacity mode 'Provisioned', and table status 'Active'. The 'Items summary' section indicates 0 items returned.

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

- > No api gateway
- > No lambda function
- > No Dyanmo Table

before running

now lets run the command.

[\${cdk deploy} command running...

it will ask Do you wish to deploy these changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

You can see no Item. |

The screenshot shows the AWS CloudFormation console with the 'Products' table selected in the 'Tables' list. The table details are visible, including partition key 'id (Number)', sort key '-', capacity mode 'Provisioned', and table status 'Active'. The 'Items summary' section indicates 0 items returned.

This command will create all the functions, dynamo table and api gateway before running

Lets check before running the command.

- > No api gateway
- > No lambda function
- > No Dyanmo Table

now lets run the command.

[`$cdk deploy`] command running..

it will ask Do you wish to deploy changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

You can see no Item.

The screenshot shows the AWS CDK Java CLI terminal on the left and the Postman application on the right. The terminal output matches the text above. The Postman interface shows a POST request to 'https://juksww0qld.execute-api.ca-central-1.amazonaws.com/prod/products' with a JSON body containing an item with id 1, name 'Random product 1', and price 12.34. The 'Body' tab is selected, and the 'JSON' radio button is chosen. Below the body, the 'Responses' section shows a status of 'Sending request...' with a cartoon character icon.

This command will create all the functions, dynamo table and api gateway before running

Lets check before running the command.

- > No api gateway
- > No lambda function
- > No Dyanmo Table

now lets run the command.

[`$cdk deploy`] command running..

it will ask Do you wish to deploy changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

You can see no Item.

The screenshot shows the AWS CDK Java CLI terminal on the left and the Postman application on the right. The terminal output matches the text above. The Postman interface shows the same POST request to 'https://juksww0qld.execute-api.ca-central-1.amazonaws.com/prod/products'. The 'Body' tab is selected, and the 'JSON' radio button is chosen. Below the body, the 'Responses' section shows a successful response with status 200 OK, time 13.41 s, and size 526 B. The response body is a JSON object with a single key 'message' and the value 'Item created with ID : 1'.

This command will create all the functions, dynamo table and api gateway before running

Lets check before running the command.

- > No api gateway
- > No lambda function
- > No Dyanmo Table

now lets run the command.

[`$cdk deploy`] command running..

it will ask Do you wish to deploy changes (y/n)?
Type Y and enter

In progress.

Done

Lets check

Lets test our api from postman

You can see no Item.

The screenshot shows the AWS CDK Java CLI terminal on the left and the AWS Lambda and DynamoDB management console on the right. The terminal output matches the text above. The AWS interface shows the Lambda and DynamoDB services. In the DynamoDB section, a 'products' table is selected. The 'Tables (1)' list shows one table named 'products'. The 'Items returned (1)' table shows one item with id 1 and name 'Random pr...'. The 'Scan/Query items' section is visible below.

This command functions, dynamo

Lets check b

- > No api gat
- > No lambda
- > No Dyanmo

before running

now lets run

[`$cdk deploy`

it will ask changes (y/n)?

Type Y and e

In progress.

Done

Lets check

Lets test ou

You can see no It

This command functions, dynamo

Lets check b

- > No api gat
- > No lambda
- > No Dyanmo

before running

now lets run

[`$cdk deploy`

it will ask changes (y/n)?

Type Y and e

In progress.

Done

Lets check

Lets test ou

You can see no It

Lets create

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

- > No api gateway
- > No lambda function
- > No Dyanmo Table

before running

now lets run the command.

[`$cdk deploy`] command running...

it will ask Do you wish to deploy these changes (y/n)?

Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

You can see no Item.

This command functions, dynamo

Lets check b

- > No api gat
- > No lambda
- > No Dyanmo

before running

now lets run

[\${cdk deploy}

it will ask changes (y/n)?
Type Y and e

In progress.

Done

Lets check

Lets test ou

You can see no It

Lets create

Lets test th

Switch to a workspace to use this feature
This is an online feature and is only available in workspaces.
Switch to Workspaces

This command functions, dynamo

Lets check b

- > No api gat
- > No lambda
- > No Dyanmo

before running

now lets run

[\${cdk deploy}

it will ask changes (y/n)?
Type Y and e

In progress.

Done

Lets check

Lets test ou

You can see no It

Switch to a workspace to use this feature
This is an online feature and is only available in workspaces.
Switch to Workspaces

This command will create all the functions, dynamo table and api gateway

Lets check before running the command

- > No api gateway
- > No lambda function
- > No Dyanmo Table

before running

now lets run the command.

[\${cdk deploy}] command running...

it will ask Do you wish to deploy these changes (y/n)?
Type Y and enter

In progress.....

Done

Lets check

Lets test our api from postman

This command functions, dynamo before running

Lets check b

- > No api gat
- > No lambda
- > No Dyanno

now lets run

[\${cdk deploy

it will ask changes (y/n)? Type Y and e

In progress.

Done

Lets check

Lets test ou

You can see no It

Lets create

Switch to a workspace to use this feature

Postman

PATCH https://juksww0qld.execute-api.ca-central-1.amazonaws.com/prod/products/2

Params Authorization Headers (9) Body Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 {
2 "name": "Random product updated v2"
3 }

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON

1 {
2 "message": "Item Updated with ID : 2"
3 }

200 OK 12.71 s 526 B Save Response

Central mofarouqe1

name Random pr... 5 Random pr... 1

This command functions, dynamo before running

Lets check b

- > No api gat
- > No lambda
- > No Dyanno

now lets run

[\${cdk deploy

it will ask changes (y/n)? Type Y and e

In progress.

Done

Lets check

Lets test ou

You can see no It

Lets create

Switch to a workspace to use this feature

Home Workspaces Reports Explore

Scratch Pad New Import

Collections APIs Environments Mock Servers Monitors History

Postman

S3 Management Console API Gateway Functions - Lambda

DynamoDB

DynamoDB > Items > products

Tables (1)

- Any table tag
- Find tables by table name
- products

Autopreview Actions Create item Update table settings

Scan/Query items

products

Items returned (2)

	id	name
1	Random pr...	5
2	Random pr...	1

This command functions, dynamo before running

Lets check b

- > No api gat
- > No lambda
- > No Dyanno

now lets run

[\${cdk deploy

it will ask changes (y/n)? Type Y and e

In progress.

Done

Lets check

Lets test ou

Switch to a workspace to use this feature

Home Workspaces Reports Explore

Scratch Pad New Import

Collections APIs Environments Mock Servers Monitors History

Postman

S3 Management Console API Gateway Functions - Lambda

DynamoDB > Items: products > Item editor

Attributes View DynamoDB JSON

```
1 {  
2 "id": 2,  
3 "name": "Random product updated v2",  
4 "price": $2.34  
5 }
```

This command functions, dynamo before running

Lets check b

- > No api gat
- > No lambda
- > No Dyanmo

now lets run

[\${cdk deploy}

it will ask changes (y/n)?
Type Y and e

In progress.

Done

Lets check

Lets test ou

The screenshot shows the Postman interface. A request is being made to `https://juksww0qid.execute-api.ca-central-1.amazonaws.com/prod/products/2` using the PATCH method. The body of the request contains JSON data: `{ "name": "Random product updated v3", "price": 49.8 }`. The response status is 200 OK with a message: "Item Updated with ID : 2".

This command functions, dynamo before running

Lets check b

- > No api gat
- > No lambda
- > No Dyanmo

now lets run

[\${cdk deploy}

it will ask changes (y/n)?
Type Y and e

In progress.

Done

Lets check

Lets test ou

You can see no It

Lets create

The screenshot shows the AWS DynamoDB Management Console. The products table has two items:

id	name	price
2	Random pr...	5
1	Random pr...	1

This command functions, dynamo before running

Lets check b

- > No api gat
- > No lambda
- > No Dyanmo

now lets run

[\${cdk deploy}

it will ask changes (y/n)?
Type Y and e

In proress.

The screenshot shows the AWS DynamoDB Item editor for the products table. The item has been updated with the following attributes:

Attribute	Value
id	2
name	"Random product updated v3"
price	49.8

```

aws-cdk-java — Edited

This command will create all the
functions, dynamo table and api gateway

Lets check before running the command

> No api gateway
> No lambda function
> No Dynamo Table
before running

now lets run the command.

[$cdk deploy] command running...

it will ask Do you wish to deploy these
changes (y/n)?
Type Y and enter

In progress.....
Done
Lets check
Lets test our api from postman

```

This screenshot shows the Postman interface. A PATCH request is made to <https://juksww0qjd.execute-api.ca-central-1.amazonaws.com/prod/products/2>. The Body tab contains the following JSON:

```

1 {
2   "name": "Random product updated v3",
3   "price": 49.88
4 }

```

The response status is 200 OK, with a message: "Item Updated with ID : 2".

This screenshot shows the Postman interface. A DELETE request is made to <https://juksww0qjd.execute-api.ca-central-1.amazonaws.com/prod/products/2>. The response status is 200 OK, with a message: "Item deleted with ID : 2".

This screenshot shows the AWS DynamoDB Management Console. The products table is selected. The table has one item with the following data:

	id	name
1	Random pr...	Random pr...

Lets test the update api

Lets test the delete api

Everything looks good

9. now if we keep this in aws, it will cost us money. So if you dont need this resources, we can run cdk destroy to delete all this resources.

Lets run cdk destroy and check if everything got deleted or not

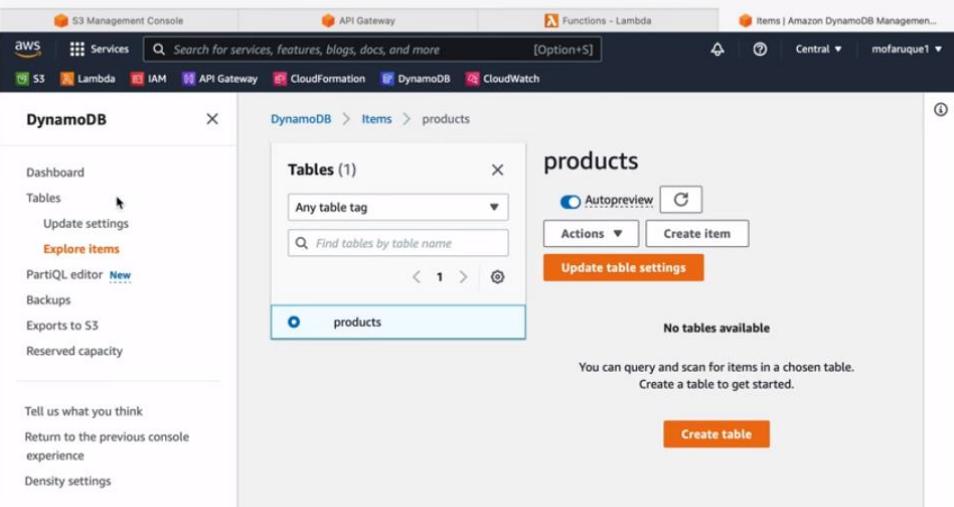
```
$ cdk destroy running .....
```

It will ask : Are you sure you want to delete: ProductApiCdkStack (y/n)?

Type y and Enter

Command running

Done, lets check



Lets test the update api

Lets test the delete api

Everything looks good

9. now if we keep this in aws, it will cost us money. So if you dont need this resources, we can run cdk destroy to delete all this resources.

Lets run cdk destroy and check if everything got deleted or not

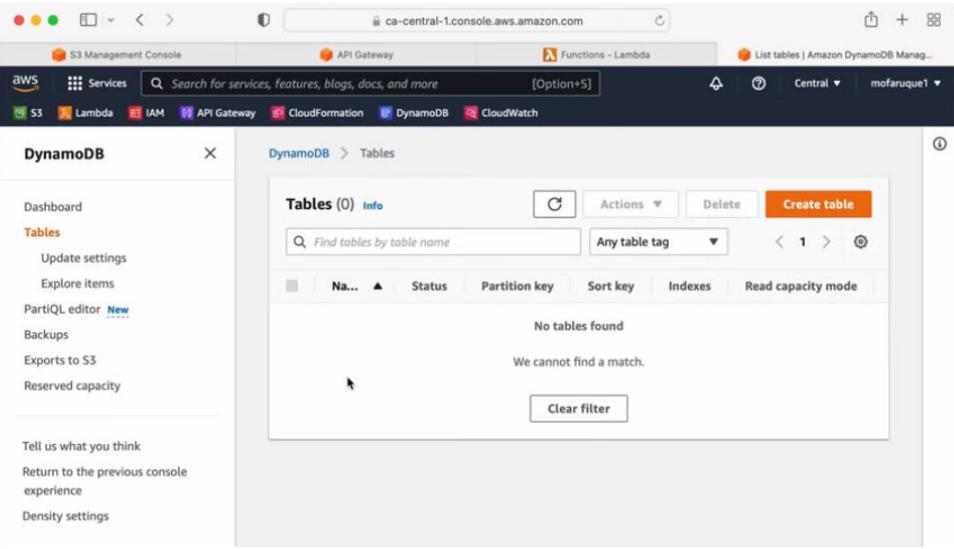
```
$ cdk destroy running .....
```

It will ask : Are you sure you want to delete: ProductApiCdkStack (y/n)?

Type y and Enter

Command running

Done, lets check



Lets test the update api

Lets test the delete api

Everything looks good

9. now if we keep this in aws, it will cost us money. So if you dont need this resources, we can run cdk destroy to delete all this resources.

Lets run cdk destroy and check if everything got deleted or not

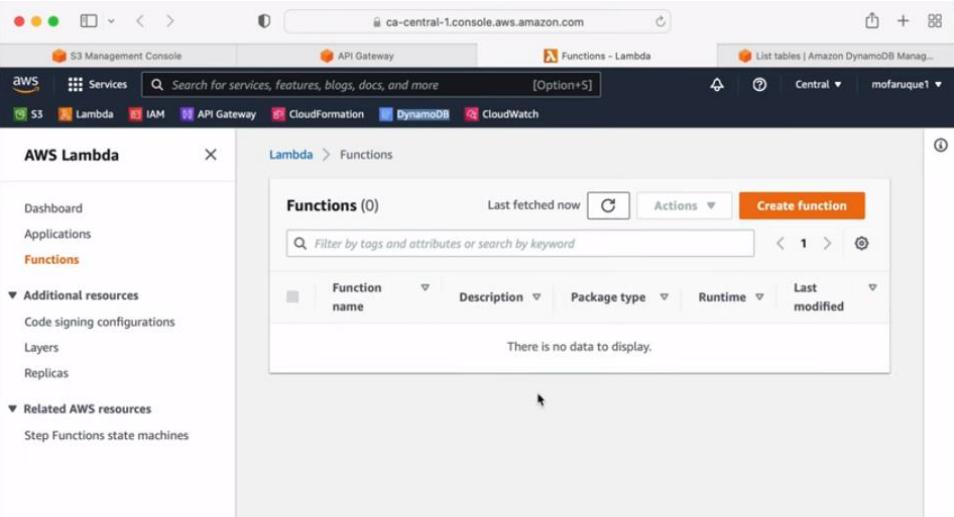
```
$ cdk destroy running .....
```

It will ask : Are you sure you want to delete: ProductApiCdkStack (y/n)?

Type y and Enter

Command running

Done, lets check



```

aws-cdk-java — Edited
Lets test the update api
Lets test the delete api
Everything looks good
9. now if we keep this in aws, it will cost us money. So if you dont need this resources, we can run cdk destroy to delete all this resources.

Lets run cdk destroy and check if everything got deleted or not

$ cdk destroy running .....

It will ask : Are you sure you want to delete: ProductApiCdkStack (y/n)?

```

Type y and Enter

Command running

Done, lets check

Amazon API Gateway
create, maintain, and secure APIs at any scale

Amazon API Gateway helps developers to create and manage APIs to back-end systems running on Amazon EC2, AWS Lambda, or any publicly addressable web service. With Amazon API Gateway, you can generate custom client SDKs for your APIs, to connect your back-end systems to mobile, web, and server applications or services.

Choose an API type

HTTP API

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:
Lambda, HTTP backends

Import **Build**

WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

```

aws-cdk-java — Edited
this resources, we can run cdk destroy to delete all this resources.

Lets run cdk destroy and check if everything got deleted or not

$ cdk destroy running .....

It will ask : Are you sure you want to delete: ProductApiCdkStack (y/n)?

```

Type y and Enter

Command running

Done, lets check

Some useful command

```

## Useful commands

* `mvn package`      compile and run tests
* `cdk ls`           list all stacks in the app
* `cdk synth`         emits the synthesized CloudFormation template
* `cdk deploy`        deploy this stack to your default AWS account/region
* `cdk diff`          compare deployed stack with current state
* `cdk docs`          open CDK documentation

```

THANK YOU FOR WATCHING

API Gateway **Functions - Lambda** **DynamoDB** **CloudWatch**

Choose an API type

HTTP API

Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:
Lambda, HTTP backends

Import **Build**

WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Import **Build**