הוראות התקנה - מעבדה 2:

שלב 1 - יצירת דיאגרמה ארכיטקטונית ותחשיב עלויות
שלב 2 - בדיקת המערכת הקיימת ע"י כניסה לאתר ה-Monolithic application
לפי כתובת הIP של ה-EC2
בדיקת תקינות שניתן להוסיף/לערוך/למחות רשומות של suppliers

שלב 2.3 - ניתוח אופן הרצת האפליקציה המונוליטית

1. התחברות לשרת

השתמש ב־EC2 Instance Connect כדי להתחבר לשרת בשם **`MonolithicAppServer`**.

2. בדיקת האפליקציה

הרץ את הפקודה הבאה כדי לבדוק איזו תוכנה מאזינה על פורט 80:

```
sudo lsof -i :80
```

שים לב אילו תהליכים מאזינים על פורט 80 ובאיזו פרוטוקול ופורט רץ תהליך ה־node.

3. בדיקת תהליך node

הרץ את הפקודה הבאה כדי לבדוק את התהליכים שקשורים ל־node:

```
ps -ef | head -1; ps -ef | grep node
```

בדוק איזה משתמש מריץ את תהליך ה־node והאם מזהה התהליך (PID) תואם למה שראית בפקודה הקודמת.

4. ניתוח קוד האפליקציה

עבור לתיקיית קוד האפליקציה:

```
cd ~/resources/codebase_partner

ls
```

בתיקייה זו קיים הקובץ `index.js` שמכיל את הלוגיקה הבסיסית של האפליקציה.

חיבור למסד הנתונים RDS

1. **מציאת כתובת ה־endpoint**
מצא את כתובת ה־endpoint של מסד הנתונים מסוג RDS בסביבת הלימוד.

2. **בדיקת גישה למסד הנתונים**

השתמש בפקודה הבאה כדי לבדוק אם הפורט הסטנדרטי של MySQL פתוח (3306):

```
nmap -Pn <endpoint>
```

3. **התחברות ל־MySQL**

התחבר למסד הנתונים באמצעות לקוח MySQL שמותקן כבר על השרת:

```
mysql -u admin -p
```

כאשר תתבקש, הזן את הסיסמה:

```
lab-password
```

4. **בדיקת נתונים במסד הנתונים**

מהרשומה `mysql>`, הרץ את הפקודות הבאות כדי לבדוק את התוכן:

```
SHOW DATABASES;

USE COFFEE;

SHOW TABLES;

SELECT * FROM suppliers;
```

5. **יציאה**

צא מה־MySQL:

```
EXIT;
```

סגור את חלונית EC2 Instance Connect ואת חלונית האפליקציה "coffee suppliers".

שלב 3.1 יצירת סביבת עבודה
צור סביבת עבודה ב-Cloud9 בשם `MicroservicesIDE`, השתמש בשרת EC2 חדש מסוג t3.small ולהריץ את מערכת ההפעלה Amazon Linux 2. השרת צריך לתמוך בחיבורים דרך SSH ולהיות מופעל בתוך רשת ה־LabVPC בתת-הרשת הציבורית מספר 1 (Public Subnet1).

שלב 3.2 העתקת קוד האפליקציה ל־IDE שלך

1. **הורדת קובץ ההתחברות (`labsuser.pem`)**

מהפאנל *AWS Details* בהוראות המעבדה, הורד את הקובץ `labsuser.pem` למחשב המקומי שלך.

2. **העלאת הקובץ ל־Cloud9**

העלה את הקובץ שהורדת אל סביבת AWS Cloud9 שלך.

3. **שינוי הרשאות על הקובץ**

השתמש בפקודה הבאה כדי להגדיר הרשאות מתאימות לקובץ:

```
chmod 400 ~/environment/labsuser.pem
```

4. **יצירת תיקיית זמנית**

צור תיקייה זמנית בנתיב הבא:

```
mkdir -p /home/ec2-user/environment/temp
```

5. **שליפת כתובת ה־IPv4 הפרטית של השרת**

היכנס לקונסול של Amazon EC2 ומצא את כתובת ה־Private IPv4 של השרת `MonolithicAppServer`.

6. **העתקת קוד המקור באמצעות scp**

השתמש בפקודה הבאה (התאם את כתובת ה־IP):

```
scp -r -i ~/environment/labsuser.pem
ubuntu@<כתובת-IP>:/home/ubuntu/resources/codebase_partner/*
~/environment/temp/
```

7. **אימות ההעתקה**

בדוק בדפדפן הקבצים של Cloud9 שהקבצים אכן הועתקו לתיקייה `temp`.

שלב 3.3 - יצירת תקיות עבור Microservices

רשום את הפקודות הבאות ליצירת תקיות

```
mkdir microservices

cd microservices

mkdir customer

mkdir employee
```

והפקודות הבאות להעתקת קבצי המקור לתקיות ה-Microservices ומחיקת התקייה הזמנית

```
cp -r ~/environment/temp/* ~environment/microservices/customer/

mv -f ~/environment/temp/* ~environment/microservices/employee/

rmdir ~/environment/temp/
```

שלב 3.4 יצירת Repository ב-CodeCommit בשם microservices והעלאת הקבצים

```
cd ~/environment/microservices

git init

git branch -m dev

git add .

git commit -m 'two unmodified copies of the application code'

git remote add origin
https://git-codecommit.us-east-1.amazonaws.com/v1/repos/microservice
s

git push -u origin dev
```

בדוק בCodeCommit שנוצר Repository חדש בשם microservices

שלב 4.1 התאמת הגדרות קבוצת האבטחה של ה־EC2 של Cloud9

בשלב זה תשתמש ב־AWS Cloud9 כסביבת בדיקות.

תפעיל קונטיינרים של Docker כדי לבדוק microservices.
כדי שתוכל לגשת אליהם דרך הדפדפן, עליך לפתוח פורטים מתאימים בגדרות security groups של Cloud9.

- פתח תעבורת TCP נכנסת על הפורטים:
  - 8080
  - 8081

הפעולה תתבצע דרך הגדרות קבוצת האבטחה של מופע ה־EC2 של Cloud9.

שלב 4.2 שינוי קוד המקור של שירות הלקוח (Customer Microservice)

בקוד השירות של הלקוח יש עדיין פונקציונליות של ניהול ספקים (add/edit/delete) שלא מתאימה ללקוחות.

הסר את הקוד כדי להשאיר רק פעולות קריאה (read-only) כמו בדוגמא.

שינוי בקובץ `supplier.controller.js`

נתיב: `customer/app/controller/supplier.controller.js`

```
const Supplier = require("../models/supplier.model.js");
const {body, validationResult} = require("express-validator");

exports.findAll = (req, res) => {
  Supplier.getAll((err, data) => {
    if (err)
      res.render("500", {message: "The was a problem retrieving the list of suppliers"});
    else res.render("supplier-list-all", {suppliers: data});
  });
};

exports.findOne = (req, res) => {
  Supplier.findById(req.params.id, (err, data) => {
    if (err) {
      if (err.kind === "not_found") {
        res.status(404).send({
          message: `Not found Supplier with id ${req.params.id}.`
        });
      } else {
        res.render("500", {message: `Error retrieving Supplier with id ${req.params.id}`});
      }
    } else res.render("supplier-update", {supplier: data});
  });
};
```

שינוי בקובץ supplier.model.js

נתיב: customer/app/models/supplier.model.js

- השאר רק את הפונקציות:
  - Supplier.getAll
  - Supplier.findById
- אל תשכח להשאיר את השורה האחרונה:

```
module.exports = Supplier;
```

שינוי בתפריט הניווט

נתיב: customer/views/nav.html

- שורה 3: שנה מ־Monolithic Coffee suppliers ל־Coffee suppliers
- שורה 7: שנה מ־Home ל־Customer home
- לאחר שורה 8, הוסף את השורה הבאה:

```
<a class="nav-link" href="/admin/suppliers">Administrator link</a>
```

ניקוי ממשק המשתמש

נתיב: customer/views/supplier-list-all.html

- מחק את שורה 32 (כפתור "Add a new supplier")
- מחק את שורות 26 ו־27 (כפתורי עריכה/עדכון של ספקים)

מחיקת קבצים מיותרים

מחק את הקבצים הבאים מתיקיית customer/views:

- supplier-add.html
- supplier-form-fields.html
- supplier-update.html

שינוי בקובץ index.js

נתיב: customer/index.js

- **שורות 27 עד 37 – הפוך אותן להערות ע"י הוספת // בתחילת כל שורה**
- **שורה 45 – שנה את מספר הפורט ל־8080**

```
app.listen(8080, () => {
    console.log("Server is running on port 8080.");
});
```

שלב 4.3 יצירת Docker עבור customer microservice

בנתיב `/customer`, צרו קובץ חדש בשם `Dockerfile` עם התוכן הבא:

```
FROM  node:11-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY . .
RUN npm install
EXPOSE 8080
CMD ["npm", "run", "start"]
```

בתוך הטרמינל של AWS Cloud9, עברו לתיקיית `customer`:

```
cd ~/environment/microservices/customer

docker build --tag customer .
```

לאחר מכן, בדקו שה-Image נוצר:

```
docker images
```

```
voclabs:~/environment/microservices/employee (dev) $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
employee     latest    614abdd0b631   13 seconds ago   82.7MB
customer     latest    1a55d56ea8c7   43 minutes ago   82.7MB
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cryyakcaqjr1.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/employee (dev) $ docker run -d --name employee_1 -p 8081:8081 -e APP_DB_HOST="$dbEndpoint" employee
e59aa479ea564ea3c5b6f459141aa2ee71e85c13aeea409875c25a9f980eccbc
voclabs:~/environment/microservices/employee (dev) $ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                                   NAMES
e59aa479ea56   employee   "docker-entrypoint.s…"   3 minutes ago    Up 3 minutes    8080/tcp, 0.0.0.0:8081->8081/tcp, :::8081->8081/tcp    employee_1
1f620b4a3e16   customer   "docker-entrypoint.s…"   38 minutes ago   Up 38 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp               customer_1
voclabs:~/environment/microservices/employee (dev) $
```

נגדיר משתנה סביבה עם כתובת ה-DB:

```
dbEndpoint=$(cat
~/environment/microservices/customer/app/config/config.js | grep
'APP_DB_HOST' | cut -d '"' -f2)
echo $dbEndpoint
```

ונריץ את ה-Container:

```
docker run -d --name customer_1 -p 8080:8080 -e APP_DB_HOST="$dbEndpoint"
customer
```

נבדוק אילו קונטיינרים פעילים:

```
docker ps
```

ונבדוק שה-microservice של customer עובד:
ניגש את כתובת הIP של Cloud9

```
http://<cloud-9-public-IPv4-address>:8080
```

נוודא ש:

- **ניתן להציג את רשימת הספקים (List of suppliers).**
- **אין כפתורי "Add a new supplier" או כפתורי עריכה.**
- **הקישור "Administrator link" עדיין לא פעיל (עד שייבנה employee microservice).**

נשמור את השינויים ב-CodeCommit
בצעו commit ו-push של השינויים:

```
git add .
git commit -m "Refactor customer microservice for Docker deployment"
git push
```

שלב 4.4 – שינוי קוד המקור של employee microservice:

שינויים בקוד:

1. `employee/app/controller/supplier.controller.js`

מצאו את כל השורות עם `redirect` והוסיפו `admin/` לנתיב.

להרצת פקודת חיפוש מהירה:

```
cd ~/environment/microservices/employee
grep -n 'redirect' app/controller/supplier.controller.js
```

2. `employee/index.js`

הוסיפו `admin/` לכל `app.get` ו-`app.post`.

שנו את פורט ההאזנה (ב־שורה 45) ל־8081 כדי למנוע התנגשויות עם השירות של הלקוחות.

מצאו את השורות עם:

```
grep -n 'app.get\|app.post' index.js
```

שימו לב:

ב־שורה 22 הנתיב צריך להסתיים ב `/admin`

3. `employee/views/supplier-add.html` ו- `supplier-update.html`

הוסיפו `/admin` בתחילת הכתובות בתוך `action=` של הטפסים.

```
*/grep -n 'action' views
```

4. `employee/views/supplier-list-all.html` ו- `home.html`

עדכנו את הקישורים (`href`) – הוסיפו `admin/` לכתובות.
חיפוש קישורים:

```
grep -n 'href' views/supplier-list-all.html views/home.html
```

5. `employee/views/header.html`

שנו את כותרת הדף ל־`Manage coffee suppliers`.

6. `employee/views/nav.html`

שורה 3: שנו מ־"Monolithic Coffee suppliers" ל־"Manage coffee suppliers".

שורה 7: החליפו את הקישור ל:

```
<a class="nav-link" href="/admin/suppliers">Administrator home</a>
```

שורה חדשה (אחרי שורה 8): הוסיפו קישור לעמוד הלקוחות:

```
<a class="nav-link" href="/">Customer home</a>
```

לאחר כל העריכות – ודאו ששמרתם את הקבצים.

שלב 4.5 - צור Dockerfile והרץ Container לבדיקת microservice

העתק את ה-`Dockerfile` משירות הלקוחות (customer) לתיקיית העובדים (`/employee`).

ערוך את שורת `EXPOSE` בקובץ החדש ושנה את הפורט ל-`8081`.

בנה את הImage:

```
cd ~/environment/microservices/employee
docker build -t employee .
```

הרץ את הקונטיינר:

```
docker run -d --name employee_1 -p 8081:8081 -e DBHOST=<your-db-endpoint> employee
```

פתחו דפדפן עם כתובת ואמתו ש-Employee microservice עובד:

```
http://<Cloud9-IP>:8081/admin/suppliers
```

כולל רשימת ספקים, כפתורים להוספה ועריכה!

בדקו את הפונקציונליות

- הוספת ספק חדש – ודא שהוא מופיע לאחר ההוספה.
- עריכת ספק קיים – ודא שהשינויים נשמרים.
- מחיקת ספק – ערוך ספק ובחר באופציית "Delete this supplier", ואמת שהוא הוסר מהרשימה.

שלב 4.6 - שינוי הפורט ל-8080 והכנת image לפריסה ב-ECS

ערוך את הקבצים הבאים:

- `employee/index.js`: שנה את הפורט מ-`8081` ל-`8080`.
- `employee/Dockerfile`: שנה את שורת `EXPOSE` ל-`8080`.

2. מחק את הקונטיינר הישן:

```
docker rm -f employee_1
```

3. בנה מחדש את התמונה:

```
docker build -t employee .
```

אין צורך להריץ קונטיינר חדש. זה הכנה לפריסה ב-ECS.

שלב 4.7 - שליחת השינויים ל-CodeCommit

סקור שינויים

- **לחץ על סמל ה־Source Control ב-AWS Cloud9.**
- **עיין ברשימת הקבצים ששונו (כמו `index.js`) והשווה לגרסאות קודמות.**

בצע commit ו-push ל-CodeCommit

באמצעות ממשק Git של Cloud9 או בטרמינל:

```
git add .
git commit -m "Updated employee microservice to support /admin
path and changed port to 8080"
git push
```

שלב 5: יצירת מאגרי ECR, ECS Cluster, הגדרות משימה וקבצי AppSpec

שלב 5.1: יצירת מאגרי ECR והעלאת תמונות Docker

התחברות ל-ECR דרך Docker:

```
account_id=$(aws sts get-caller-identity | grep Account | cut
-d '"' -f4)
aws ecr get-login-password --region us-east-1 | docker login
--username AWS --password-stdin
$account_id.dkr.ecr.us-east-1.amazonaws.com
```

יצירת מאגרי ECR נפרדים:

- customer
- employee

עדכון הרשאות לכל Cluster:

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "ecr:*"
    }
  ]
}
```

תיוג Images

```
docker tag customer:latest
$account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
docker tag employee:latest
$account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
```

דחיפת התמונות ל-ECR:

```
docker push
$account_id.dkr.ecr.us-east-1.amazonaws.com/customer:latest
```

```
docker push
$account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
```

שלב 5.2 - יצירת ECS

- צור ECS cluster בשם `microservices-serverlesscluster` מסוג Fargate.
- הגדר אותו לשימוש ב־LabVPC, תתי־רשת PublicSubnet1 ו־PublicSubnet2 בלבד.
- עקוב אחר תהליך ההקמה ב־CloudFormation עד שהסטטוס הוא `CREATE_COMPLETE`.

שלב 5.3 - יצירת מאגר CodeCommit

- צור מאגר בשם `deployment` שישמש לאחסון קבצי הגדרה.

- ב־Cloud9, צור תיקייה בשם `deployment` והפעל בה Git עם ענף `dev`.

שלב 5.4 - הגדרות משימה (task definitions)

customer:

צור קובץ `taskdef-customer.json` עם התוכן הבא:

```
{
  "containerDefinitions": [
    {
      "name": "customer",
      "image": "<IMAGE1_NAME>",
      "environment": [
        {
          "name": "APP_DB_HOST",
          "value": "<RDS-ENDPOINT>"
        }
      ],
      ...
```

```
    }
  ],
  "requiresCompatibilities": ["FARGATE"],
  ...
  "executionRoleArn":
"arn:aws:iam::<ACCOUNT-ID>:role/PipelineRole",
  "family": "customer-microservice"
}
```

- החלף `<ACCOUNT-ID>` במספר חשבון AWS שלך.
- החלף `<RDS-ENDPOINT>` בכתובת של RDS שלך.
- הערה: לאחר ההרצה הראשונה, מחליפים את `"image": "customer"` ל־`"<IMAGE1_NAME>"`.

employee:

- שכפל את הקובץ וערוך את שלוש הפעמים שבהן מופיע המילה `customer` ל־`employee`.
- הרשם כל אחד באמצעות הפקודה:

```
aws ecs register-task-definition --cli-input-json
file:///home/ec2-user/environment/deployment/taskdef-employee.
json
```

שלב 5.5 - קובצי AppSpec

customer:

צור קובץ `appspec-customer.yaml` עם התוכן:

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
```

```
ContainerName: "customer"
ContainerPort: 8080
```

employee:

- שכפל ושנה את `ContainerName` ל־`employee`.

שלב 5.6 - עדכון הקבצים ב-CodeCommit

ערוך את שורת ה־`image` בכל קובץ JSON ל־`"<IMAGE1_NAME>"`.

שמור את כל 4 הקבצים:

- `taskdef-customer.json`
- `taskdef-employee.json`
- `appspec-customer.yaml`
- `appspec-employee.yaml`

דחוף את הקבצים ל־CodeCommit:

```
git add .
git commit -m "Added ECS task defs and AppSpec files"
git push origin dev
```

שלב 6 - יצירת Target Groups ו-Application Load Balancer

שלב 6.1 - יצירת ארבע Target Groups

בגלל שהולכים להשתמש באסטרטגיית פריסה מסוג Blue/Green, נדרשות שתי קבוצות יעד לכל מיקרו-שירות.
המשמעות:

- Blue: הסביבה הנוכחית שפועלת בפרודקשן
- Green: הגרסה החדשה שעוברת בדיקות לפני החלפה

הגדרות שיש להחיל עבור כל Target Group:

קבוצת יעד ראשונה למיקרו-שירות `customer`:

- שם: `customer-tg-one`
  סוג יעד: IP addresses
- פרוטוקול: HTTP
- פורט: 8080
- VPC: LabVPC
- נתיב לבדיקה (Health Check): `/`
- לא לרשום מטרות (targets) בשלב זה

קבוצת יעד שנייה ל-`customer`:

- זהה להגדרות הקודמות, רק עם שם: `customer-tg-two`

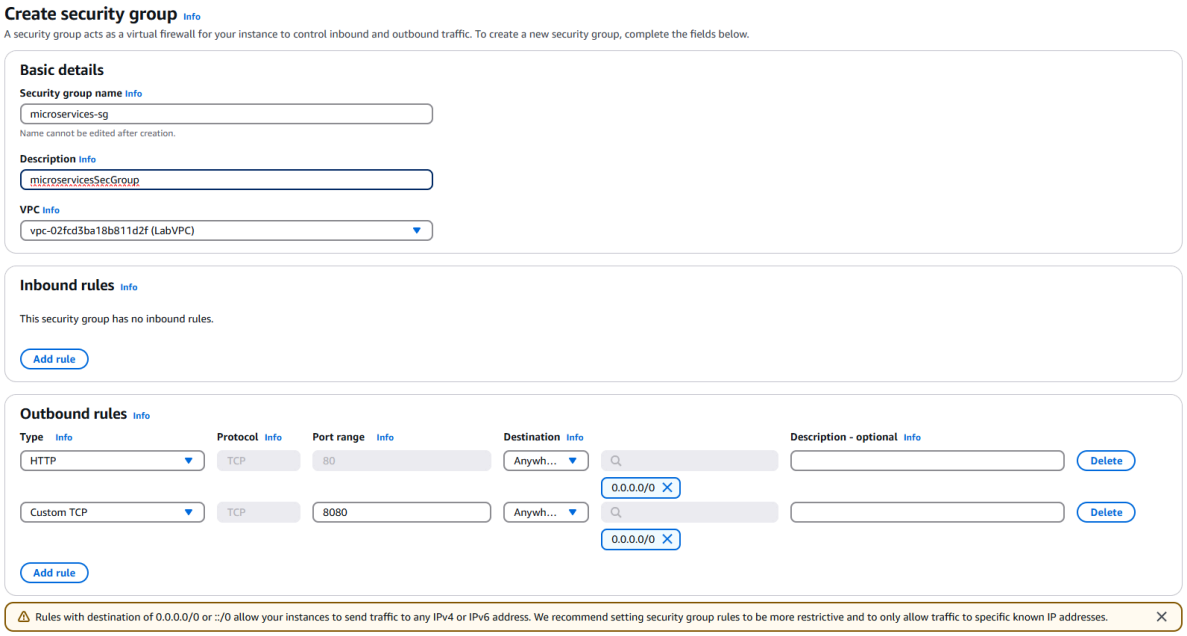
קבוצת יעד ראשונה ל-`employee`:

- שם: `employee-tg-one`
- נתיב לבדיקה (Health Check): `/admin/suppliers`
  שאר ההגדרות כמו בקבוצות הקודמות

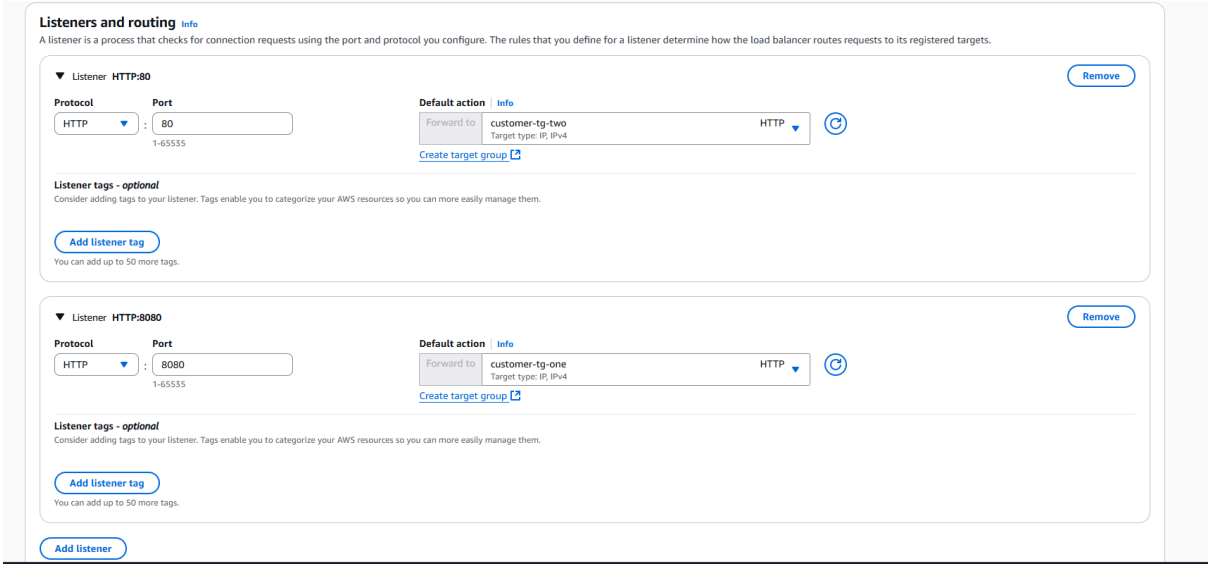
קבוצת יעד שנייה ל-`employee`:

- כמו הקודמת, עם שם: `employee-tg-two`

שלב 6.2 - יצירת security group עבור microservices



יצירת Application Load Balancer



שלב 7.1 יצירת ECS עבור Customer microservice

צור קובץ חדש `create-customer-microservice-tg-two.json` בסביבת ה-Cloud9
בתוך תיקיית deployment והעתק את התוכן הבא:

```
{
  "taskDefinition": "customer-microservice:REVISION-NUMBER",
  "cluster": "microservices-serverlesscluster",
  "loadBalancers": [
      {
          "targetGroupArn": "MICROSERVICE-TG-TWO-ARN",
          "containerName": "customer",
          "containerPort": 8080
      }
  ],
  "desiredCount": 1,
  "launchType": "FARGATE",
  "schedulingStrategy": "REPLICA",
  "deploymentController": {
      "type": "CODE_DEPLOY"
  },
  "networkConfiguration": {
      "awsvpcConfiguration": {
          "subnets": [
              "PUBLIC-SUBNET-1-ID",
              "PUBLIC-SUBNET-2-ID"
          ],
          "securityGroups": [
              "SECURITY-GROUP-ID"
          ],
          "assignPublicIp": "ENABLED"
      }
  }
}
```

משימה 7.2: יצירת שירות Amazon ECS עבור employee microservice

שלבי ביצוע:

1. שכפול קובץ JSON
   - העתק את הקובץ ששימש ליצירת שירות הלקוח (למשל: `create-customer-microservice-tg-two.json`)
   - שמור אותו בשם: `create-employee-microservice-tg-two.json` באותה תיקייה.

2. עריכת הקובץ המשוכפל:

- שורה 2: שנה את `"customer-microservice"` ל-`"employee-microservice"`
  כמו כן, עדכן את מספר הגרסה (revision number).

- שורה 6: הדבק את ARN של קבוצת היעד `employee-tg-two`
  ❗ אל תבצע רק החלפת טקסט אוטומטית – ה-ARN שונה לגמרי.

- שורה 7: שנה את `"customer"` ל-`"employee"`

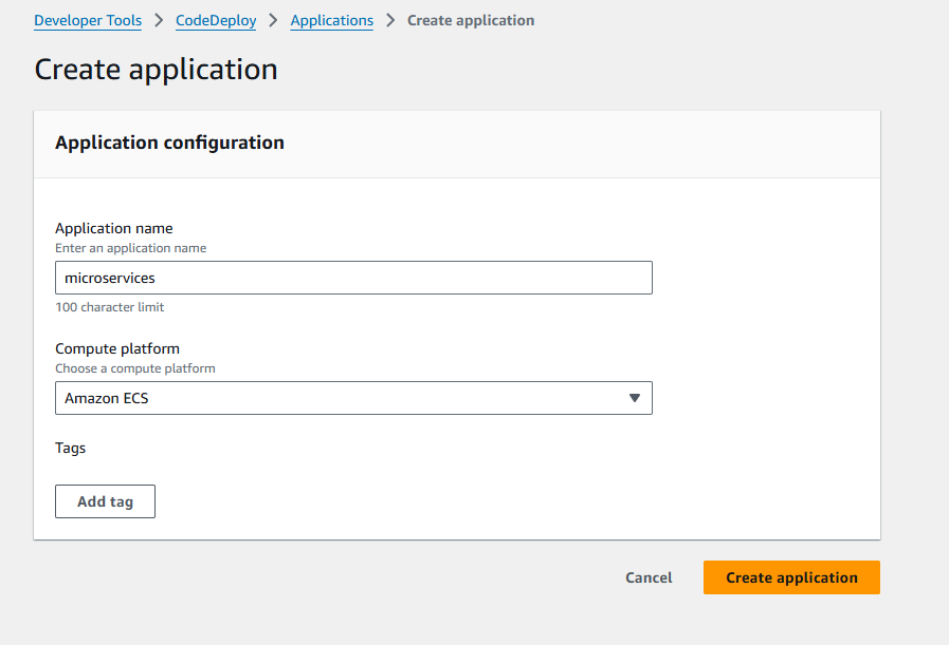
3. שמירת הקובץ:

- שמור את השינויים שביצעת בקובץ.

4. הרצת פקודת CLI מתאימה:

- הרץ את פקודת ה-AWS CLI ליצירת השירות ב-ECS תוך שימוש בקובץ ה-JSON שערכת.

שלב 8.1 יצירת CodeDeploy Application



יצירת Deployment group עבור customer

Developer Tools > CodeDeploy > Applications > microservices > Create deployment group

Create deployment group

Application

Application

microservices

Compute type

Amazon ECS

Deployment group name

Enter a deployment group name

microservices-customer

100 character limit

Service role

Enter a service role

Enter a service role with CodeDeploy permissions that grants AWS CodeDeploy access to your target instances.

arn:aws:iam::215251297864:role/DeployRole

Environment configuration

Choose an ECS cluster name

microservices-serverlesscluster

Choose an ECS service name

customer-microservice

Load balancers

Choose a load balancer

microservicesLB

Production listener port

HTTP: 80

Test listener port - *optional*

A test listener is required if you want to test your replacement version before traffic reroutes to it

HTTP: 8080

Target group 1 name

customer-tg-two

Target group 2 name

customer-tg-one

Deployment settings

Traffic rerouting

Choose whether traffic reroutes to the replacement environment immediately or waits for you to start the rerouting process

- (●) Reroute traffic immediately
- ( ) Specify when to reroute traffic

Deployment configuration

Choose from a list of default and custom deployment configurations. A deployment configuration is a set of rules that determines how fast an application is deployed and the success or failure conditions for a deployment.

CodeDeployDefault.ECSAllAtOnce or **Create deployment configuration**

Original revision termination

Specify how long CodeDeploy waits before it terminates the original task set. After termination starts, you cannot rollback manually or automatically

Days	Hours	Minutes
0	0	5

באופן דומה גם עבור Employee.

Application

Application
microservices
Compute type
Amazon ECS

Deployment group name

Enter a deployment group name
microservices-employee
100 character limit

Service role

Enter a service role
Enter a service role with CodeDeploy permissions that grants AWS CodeDeploy access to your target instances.
arn:aws:iam::215251297864:role/DeployRole

Environment configuration

Choose an ECS cluster name
microservices-serverlesscluster

Choose an ECS service name
employee-microservice

Load balancers

Choose a load balancer
microservicesLB

Production listener port
HTTP: 80

Test listener port - *optional*
A test listener is required if you want to test your replacement version before traffic reroutes to it
HTTP: 8080

Target group 1 name
employee-tg-two

Target group 2 name
employee-tg-one

Deployment settings

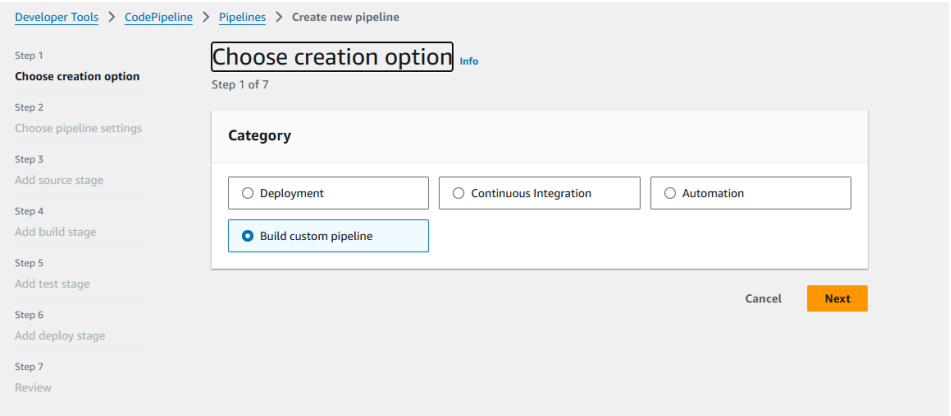
Traffic rerouting
Choose whether traffic reroutes to the replacement environment immediately or waits for you to start the rerouting process
- (●) Reroute traffic immediately
- ( ) Specify when to reroute traffic

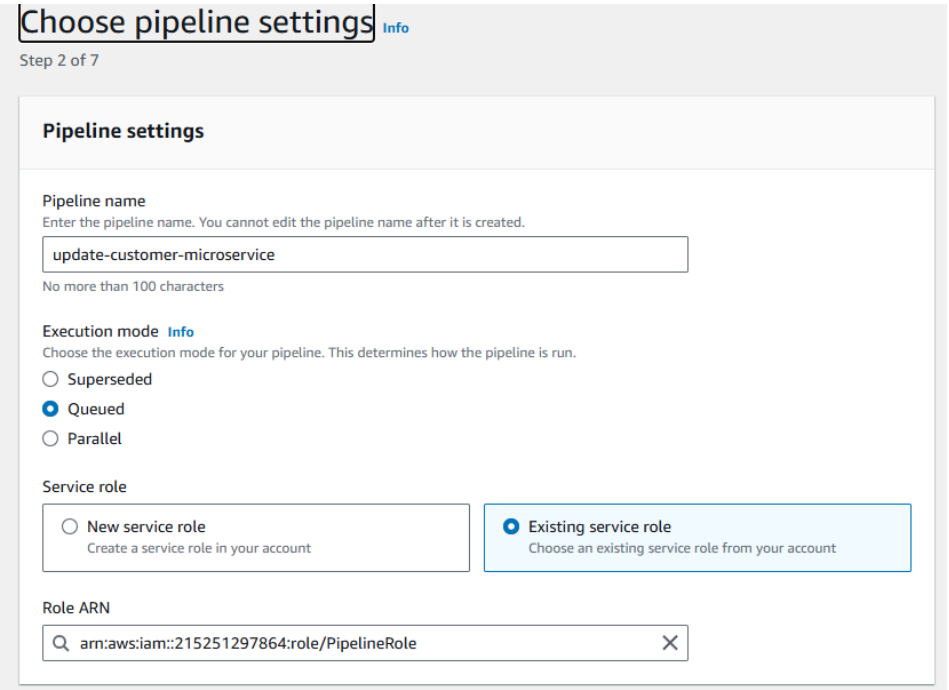
Deployment configuration
Choose from a list of default and custom deployment configurations. A deployment configuration is a set of rules that determines how fast an application is deployed and the success or failure conditions for a deployment.
CodeDeployDefault.ECSAllAtOnce or Create deployment configuration

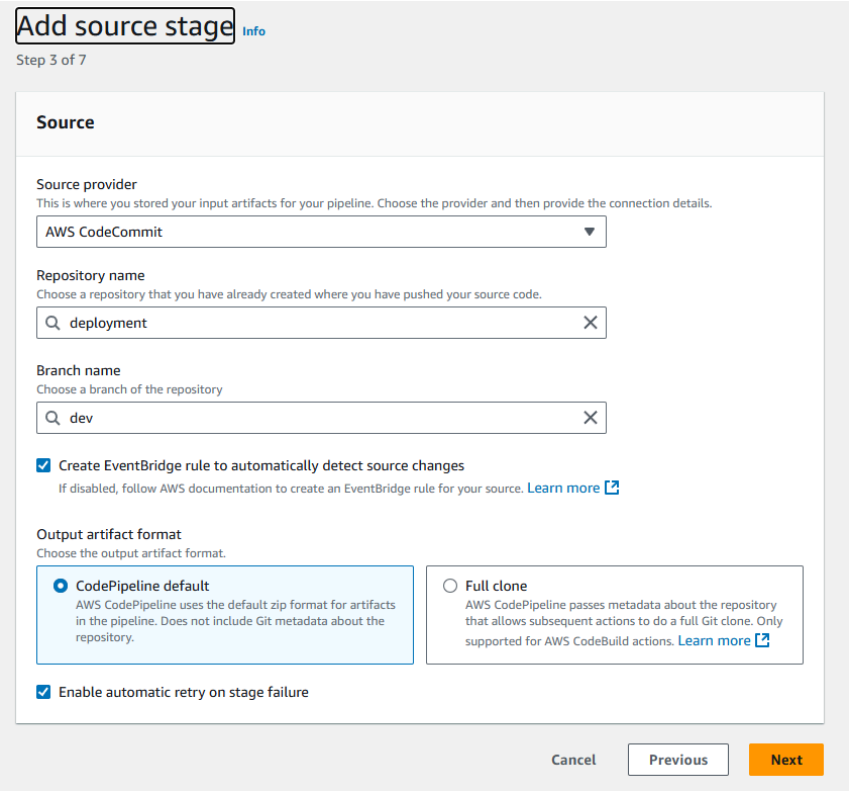
Original revision termination
Specify how long CodeDeploy waits before it terminates the original task set. After termination starts, you cannot rollback manually or automatically

Days	Hours	Minutes
0	0	5

שלב 8.2 יצירת CodePipeline

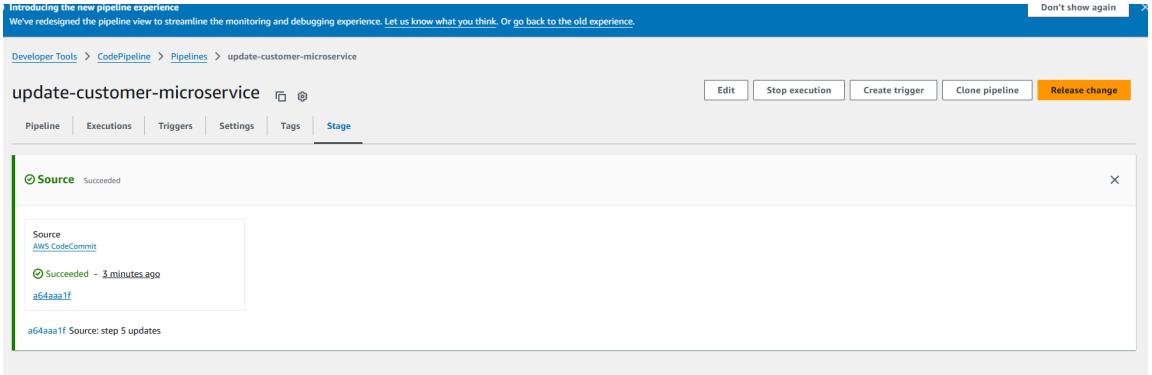




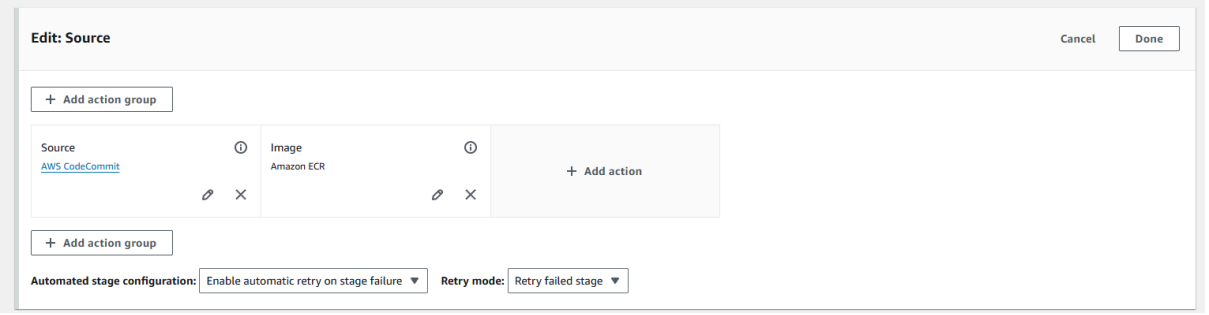


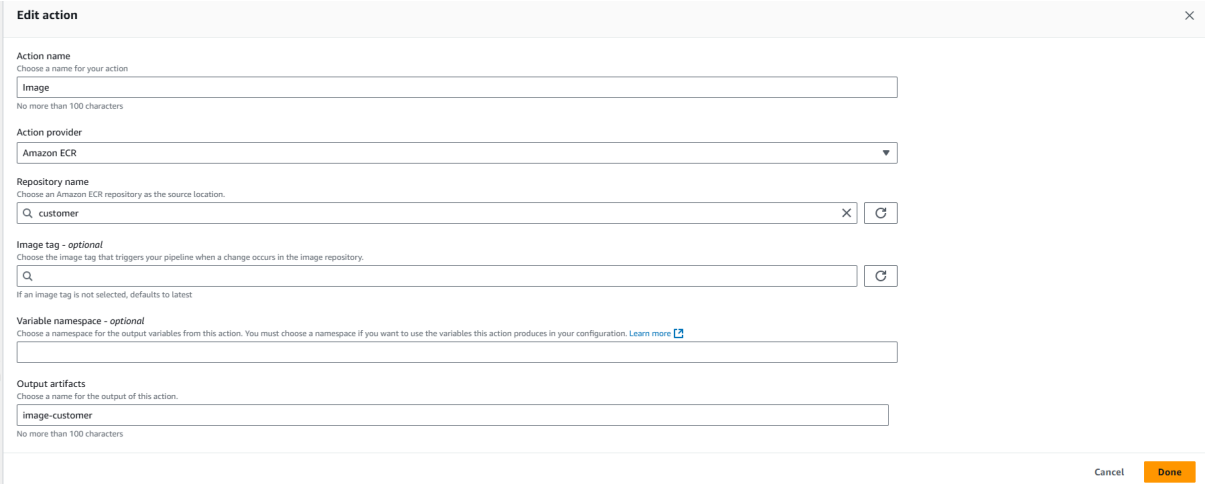
עושים skip ל-build ו-test

נכנסים ל-Pipeline שנוצר

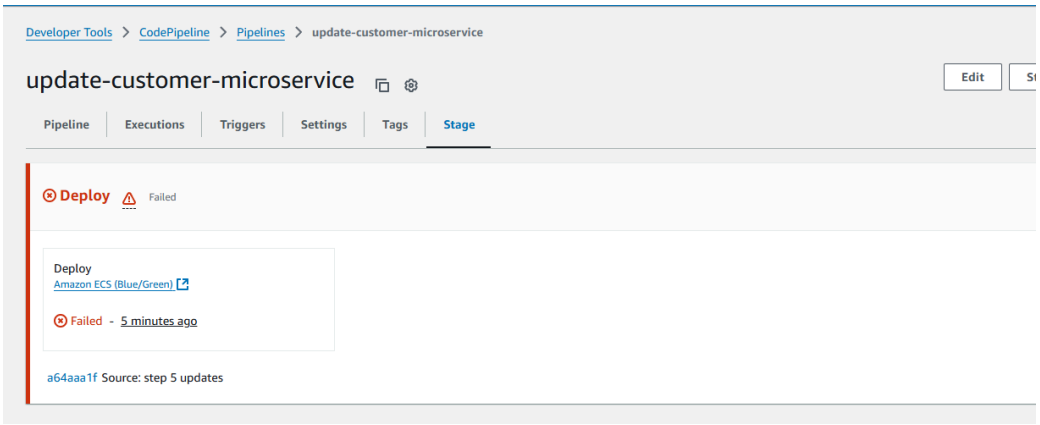


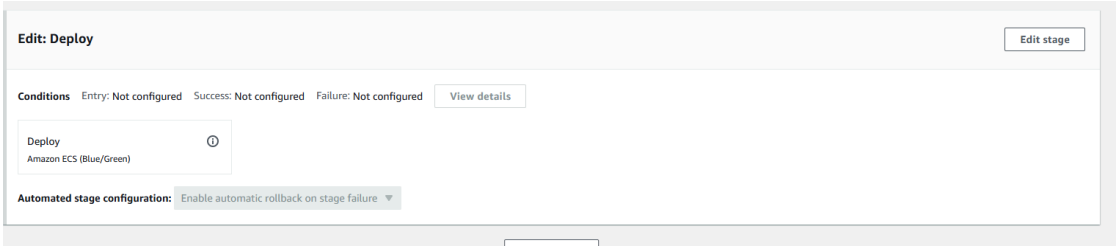
מוסיפים ל-Source

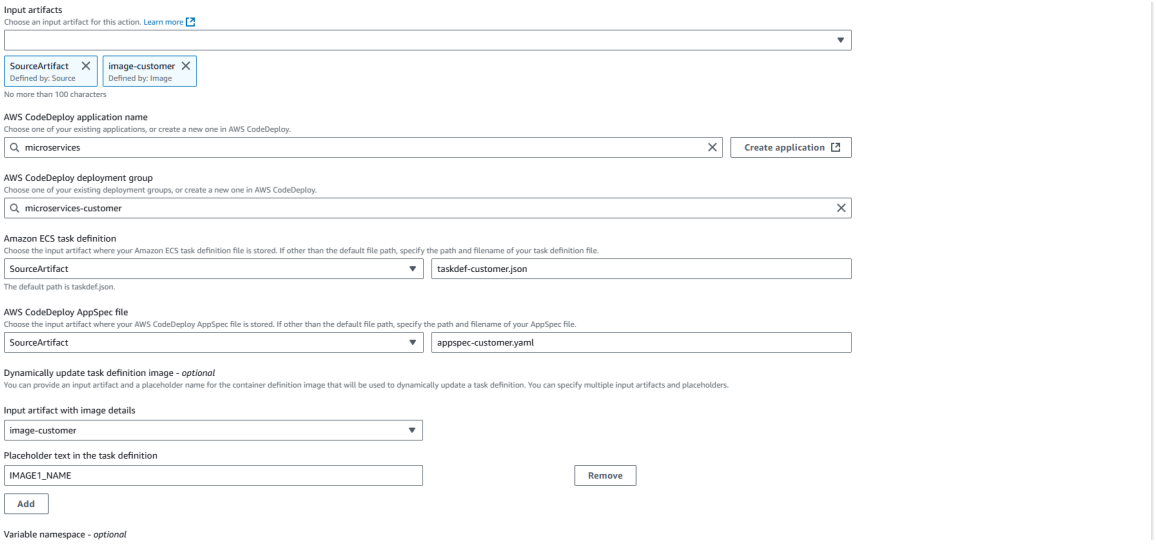




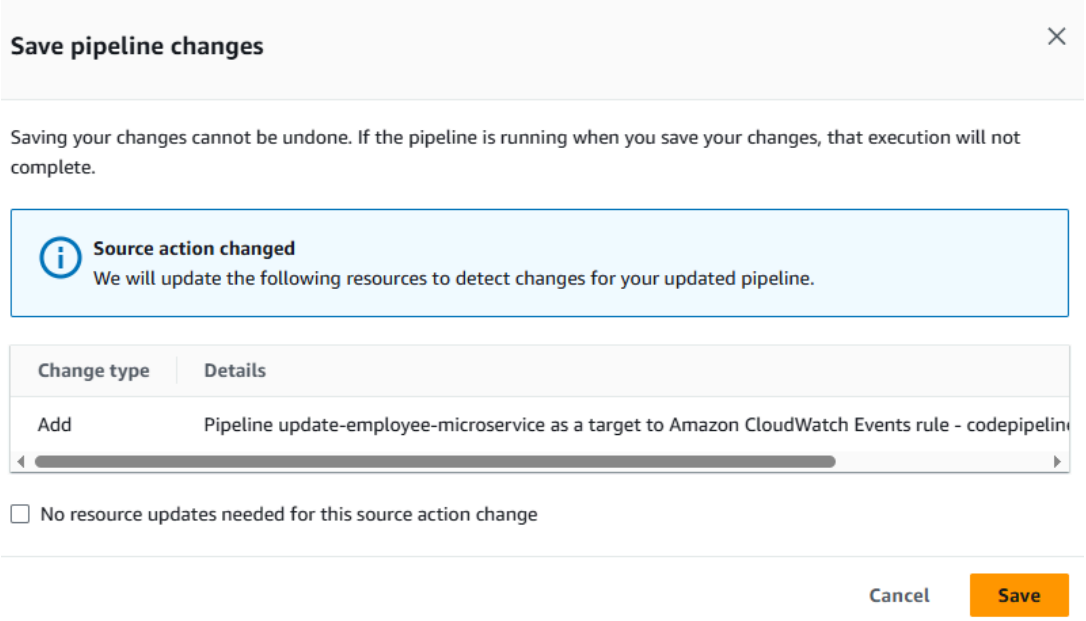
עורכים את ה-Deploy שנכשל





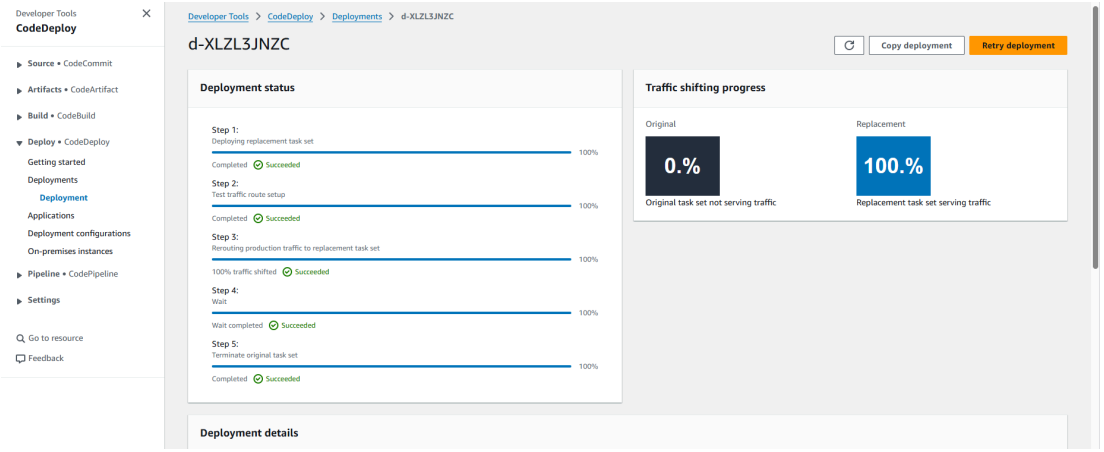


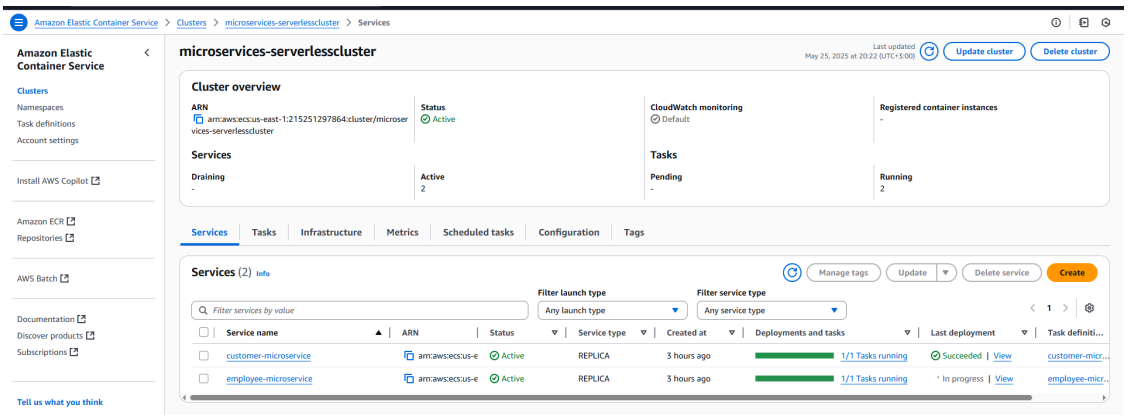
חשוב לשמור את השינויים

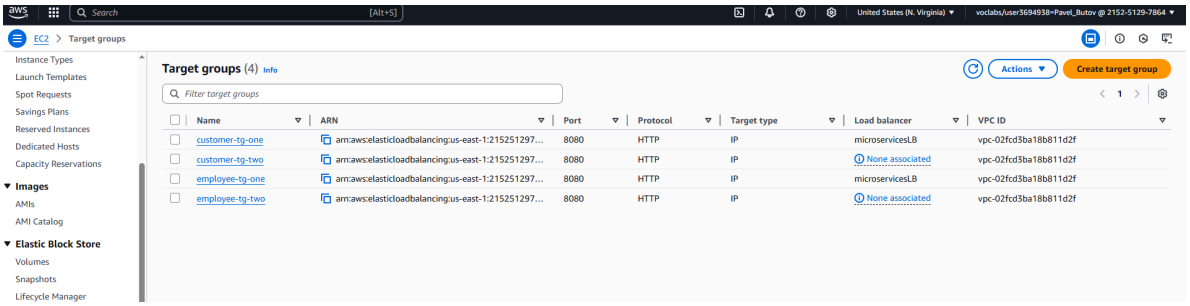


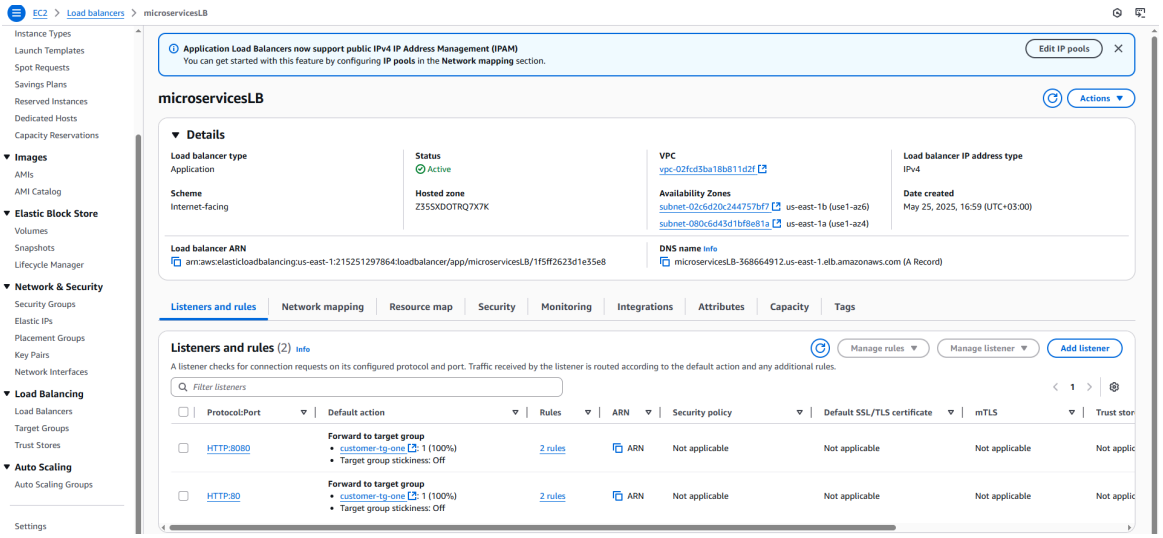
שלב 8.4 חזור על שלב 8.3 וצור Pipeline עבור employee

שלב 8.5 אחרי שמירת השינויים בדוק את המערכת



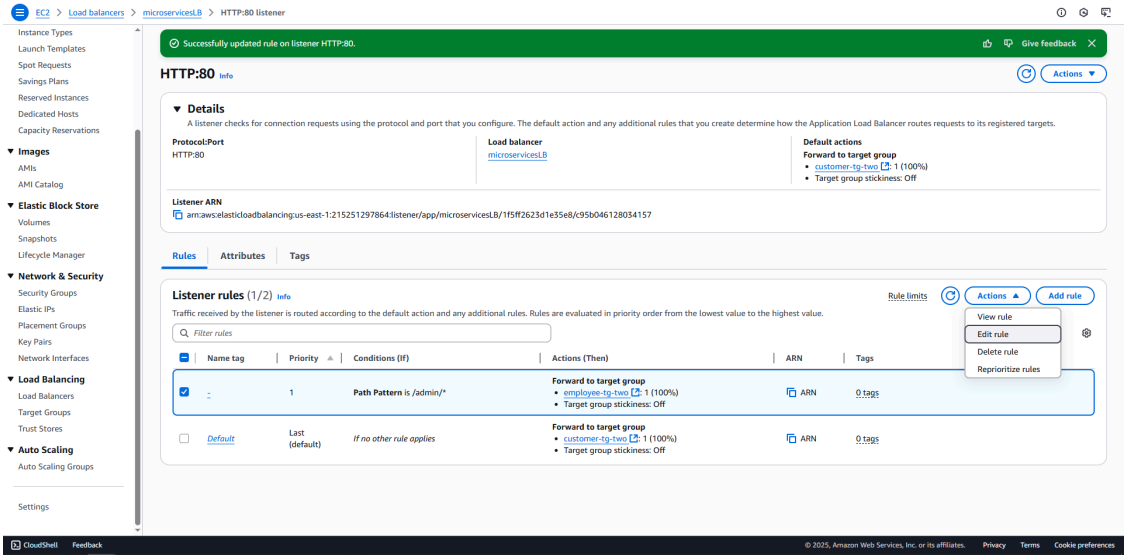


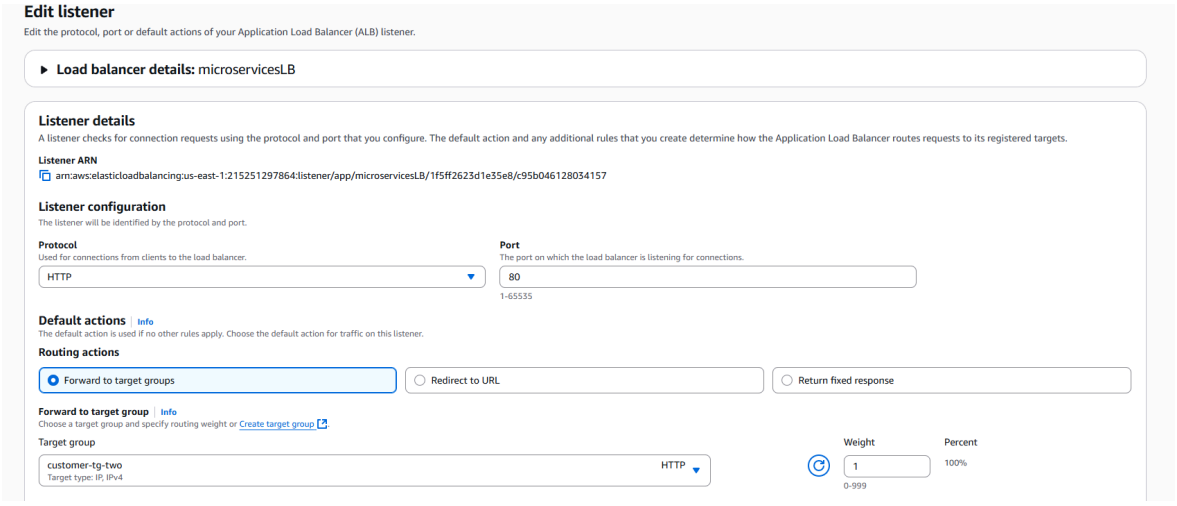




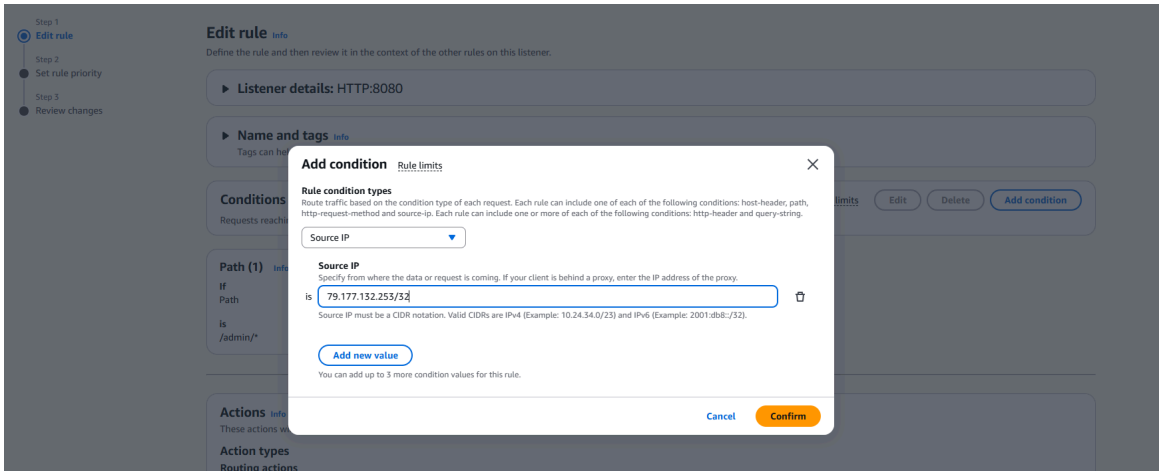
שני החוקים מצביעים ל-customer-tg-one

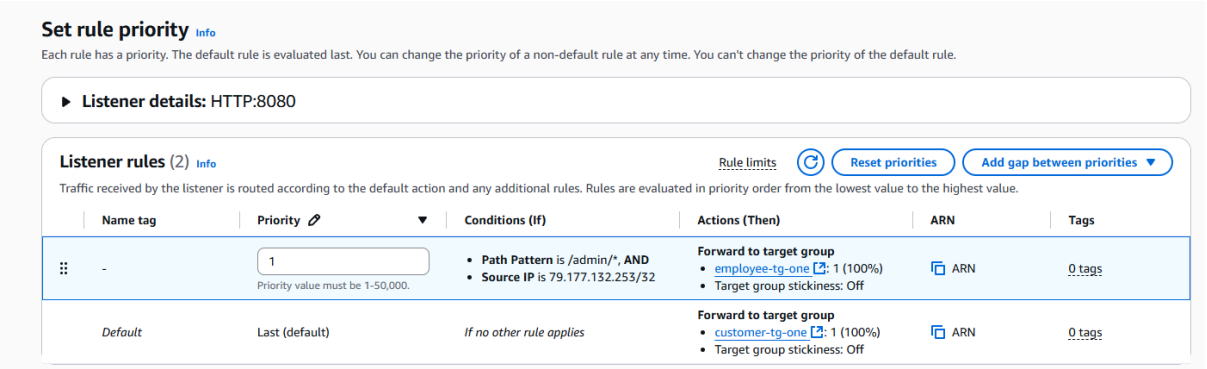
שלב 9.1 עורכים את ההגדרות של HTTP:80





להצביע חזרה ל- employee-tg-two ו-customer-tg-two
ובנוסף נוסיף עוד תנאי ל-employee המאפשר גישה רק למשתמש המחובר דרך כתובת IP
ספציפית עבור שני הפורטים HTTP:80 & 8080





מבצעים את השינו בקובץ microservices\employee\view\nav.html
בשורה 1 `navbar-dark bg-dark` to `navbar-light bg-light`

שלב 9.2 - מעדכנים את ה-Docker Image

```
 => exporting to image
 => => exporting layers
 => => writing image sha256:699e911fbef248e944440a25160336599dbba0f51a1cf3975f80975bf60ac183
 => => naming to docker.io/library/employee
voclabs:~/environment/microservices/employee (dev) $ dbEndpoint=$(cat ~/environment/microservices/employee/app/config/config.js | grep 'APP_DB_HOST' | cut -d '"' -f2)
voclabs:~/environment/microservices/employee (dev) $ echo $dbEndpoint
supplierdb.cryyakcaqjr1.us-east-1.rds.amazonaws.com
voclabs:~/environment/microservices/employee (dev) $ account_id=$(aws sts get-caller-identity |grep Account|cut -d '"' -f4)
voclabs:~/environment/microservices/employee (dev) $ echo $account_id
215251297864
voclabs:~/environment/microservices/employee (dev) $ docker tag employee:latest $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
voclabs:~/environment/microservices/employee (dev) $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $account_id.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
voclabs:~/environment/microservices/employee (dev) $ docker push $account_id.dkr.ecr.us-east-1.amazonaws.com/employee:latest
The push refers to repository [215251297864.dkr.ecr.us-east-1.amazonaws.com/employee]
78c6b56f7b27: Pushed
949374de4ba6: Pushed
5f70bf18a086: Layer already exists
6e59513952c1: Layer already exists
d81d715330b7: Layer already exists
1dc7f3bb09a4: Layer already exists
dcaceb729824: Layer already exists
f1b5933fe4b5: Layer already exists
tag invalid: The image tag 'latest' already exists in the 'employee' repository and cannot be overwritten because the tag is immutable.
```

אם יש שגיאה של tag invalid אז צריך לשנות את ה-Repository tag ל- Mutable דרך ECR

שלב 9.3 - בודקים שהמערכת התעדכנה דרך CodeDeploy אחרי הוספת השינויים

שלב 9.4 - בדיקה עם מכשיר נוסף מרשת שונה שניתן לגשת רק למערכת של Customer

שלב 9.5 - הרחבת ה-Customers למכונות נוספות לעמידה בעומס בעזרת הפקודה

```
aws ecs update-service --cluster microservices-serverlesscluster --service customer-microservice --desired-count 3
```

ב-Cloud9

סיימנו!