Kurs: Skalbara molnapplikationer
Inlämning: 2 Del 1
Namn: Maria Schillström
Datum: 2025-11-01
Git: https://github.com/MariaSchillstrom/Docker-Swarm-.NET-MVC-application-.git

Innehållsförteckning

Översikt av lösningen

Arkitektur

Note: Denna tutorial följer inte ett production-ready arbetsflöde där applikationen normalt utvecklas och testas färdigt innan infrastrukturen sätts upp. I produktion skulle antingen (1) en färdig MVC-applikation finnas innan Swarm-konfigurationen, eller (2) frontend och backend separeras där frontend hostas på t.ex. AWS S3 för att möjliggöra snabba uppdateringar utan container-rebuilds. Det sistnämnda alternativet behandlas i en separat rapport.

För denna labb antar vi att en färdig applikation finns tillgänglig.

Implementerad lösning: Jag har skapat en skalbar containerbaserad värdmiljö för en .NET MVC-webbapplikation med följande komponenter:

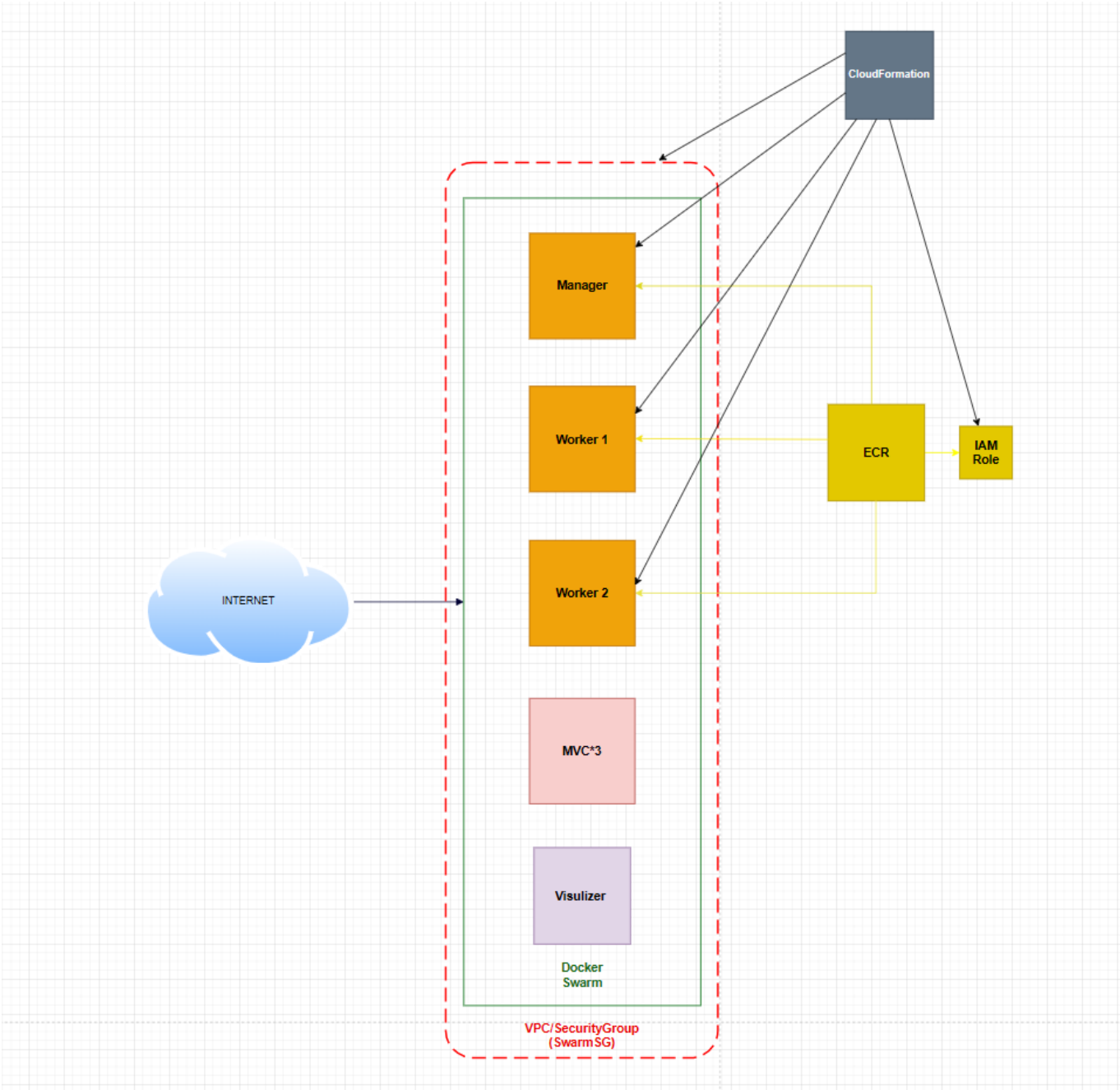
Infrastruktur:

- 3 EC2-instanser på AWS (1 Manager + 2 Workers)
- Docker Swarm orchestrerar containers över alla noder
- Security Group kontrollerar nätverkstrafik
- IAM Role ger säker åtkomst till container registry

Applikation:

- .NET MVC-app containeriserad med Docker
- Lagrad i privat ECR (Elastic Container Registry)
- Deployad med 3 replicas för redundans
- Visualizer för grafisk överblick av klustret (körs endast på Manager)

Hur det fungerar



Figur 1: Systemarkitektur för Docker Swarm på AWS. CloudFormation skapar infrastrukturen, IAM ger ECR-access, och Swarm orchestrerar containers över tre noder.

Flöde vid deployment:

1. Developer bygger Docker image lokalt
2. Image pushas till ECR
3. Deploy-skript körs på Manager
4. Manager hämtar image från ECR
5. Swarm distribuerar 3 replicas över noderna
6. Manager bestämmer placement baserat på constraints
7. Services exponeras via port 80

Fördelar med denna setup:

- **Resiliens:** Om en nod går ner fortsätter de andra
- **Skalbarhet:** `docker service scale myapp_web=5` ökar till 5 replicas

- **Loadbalancing:** Inbyggt i Swarm, ingen extern loadbalancer behövs
- **Orchestrering:** Manager koordinerar var containers ska köras
- **Rolling updates:** Uppdatera app utan downtime
- **Enkel hantering:** Ett deploy-kommando uppdaterar hela klustret

AWS-tjänster som används

EC2 (Elastic Compute Cloud)

- 3 t3.micro instanser för Swarm-klustret
- Kör Docker Engine och applikations-containers

ECR (Elastic Container Registry)

- Privat registry för Docker images
- Lagrar MVC-applikationens container image

VPC & Security Groups

- Nätverksisolering och brandväggsregler
- Kontrollerar in-/utgående trafik

IAM (Identity and Access Management)

- Hanterar behörigheter för EC2 att hämta images från ECR

CloudFormation

- Infrastructure as Code för automatiserad resurs-skapande

Komponenternas uppgift och syfte

Infrastrukturkomponenter

Security Group (SwarmSG)

- **Syfte:** Brandvägg som kontrollerar nätverkstrafik
- **Uppgift:** Tillåter SSH, HTTP, Swarm-kommunikation mellan noder

EC2 Manager-nod

- **Syfte:** Koordinerar Swarm-klustret
- **Uppgift:** Schemalägger containers, hanterar state, tar emot deploy-kommandon, bestämmer vilken worker som kör vilken container

EC2 Worker-noder (2st)

- **Syfte:** Kör applikations-containers
- **Uppgift:** Exekverar containers enligt Manager's instruktioner

ECR Repository

- **Syfte:** Lagrar Docker images privat
- **Uppgift:** Distribuerar MVC-app image till alla Swarm-noder

IAM Role (EC2-ECR-Access)

- **Syfte:** Säker åtkomst till ECR utan credentials
- **Uppgift:** Ger EC2-instanser read-only access till ECR

Applikationskomponenter

MVC Web Service (3 replicas)

- **Syfte:** Webbapplikation tillgänglig via HTTP
- **Uppgift:** Svarar på HTTP-förfrågningar, loadbalansas automatiskt av Swarm

Visualizer Service (1 replica)

- **Syfte:** Grafisk överblick av Swarm-klustret
- **Uppgift:** Visar noder, services och container-distribution
- **Placement:** Körs endast på Manager-noden (placement constraint)

Säkerhetshantering

Nätverkssäkerhet:

- Security Group begränsar SSH till min IP-adress
- Swarm-kommunikation (portar 2377, 7946, 4789) endast mellan noder
- HTTP öppen för publik access (port 80, 8080)

Åtkomstkontroll:

- IAM Role för EC2 istället för hårdkodade credentials
- Principle of least privilege - endast read-only ECR-access
- SSH-nycklar för säker server-access

Container-säkerhet:

- Privat ECR repository (inte publik Docker Hub)
- Multi-stage Dockerfile minimerar image-storlek
- .NET base images från Microsoft (verifierade)

Data-säkerhet:

- Ingen känslig data i containers eller images
- Secrets kan hanteras via Docker Secrets (ej implementerat i denna demo)

Infrastructure as Code och Automation

CloudFormation Templates

sg.yaml - Parametriserad Security Group

- Definierar alla nätverksregler
- Self-reference för Swarm-kommunikation
- Parametriserad för återanvändbarhet

ec2.yaml - EC2-instanser med IAM Role

- Skapar 3 instanser (1 Manager + 2 Workers)
- IAM Role för ECR-access inkluderad
- UserData installerar Docker automatiskt vid start

Fördelar med IaC:

- Reproducerbar infrastruktur
- Versionskontroll av infrastruktur-kod
- Enkel att skapa/radera hela miljön
- Dokumentation genom kod

Automation

Bash-skript (docker-stack.sh):

- Automatiserar deployment av Docker Stack
- Skapar docker-stack.yml dynamiskt
- Deployer och verifierar services automatiskt

EC2 UserData:

```
#!/bin/bash
dnf update -y
dnf install -y docker
systemctl enable --now docker
usermod -aG docker ec2-user
```

Installerar och konfigurerar Docker automatiskt vid instance-start.

Dockerfile: Multi-stage build för optimerad image-skapande och minimal runtime-image.

Webbapplikationen

Test-applikation: .NET MVC

Beskrivning: En minimal ASP.NET Core MVC-applikation med standard template.

Syfte: Verifiera att:

- Container-bygge fungerar
- ECR push/pull fungerar
- Swarm loadbalancing fungerar
- Services kan nås från internet

Funktionalitet:

- Standard MVC hem-sida
- Visar att applikationen körs
- Minimal för att fokusera på infrastruktur

Varför .NET MVC:

- Representerar en verklig webbapplikation
- Visar att Swarm kan hantera stateful applikationer
- Containeriseras enkelt med Dockerfile

Implementation - Steg för steg

1. Skapa infrastruktur med CloudFormation

Jag började med att skapa resurser manuellt för att förstå strukturen, och använde sedan IaC Generator för att generera CloudFormation templates. Dessa parametriserades för återanvändbarhet.

Skapade resurser:

- Security Group (se bilaga)
- 3 EC2-instanser med IAM Role (IAM lades till senare)

Kör skripten:

```
# Security Group
aws cloudformation create-stack \
  --stack-name swarm-sg \
  --template-body file://templates/sg.yaml

# EC2-instanser
aws cloudformation create-stack \
  --stack-name swarm-ec2 \
  --template-body file://templates/ec2.yaml \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameters ParameterKey=SubnetId,ParameterValue=subnet-058684dd8c601d55d
```

Verifiera:

```
aws cloudformation describe-stacks --stack-name swarm-sg --query
'Stacks[0].StackStatus'
aws cloudformation describe-stacks --stack-name swarm-ec2 --query
'Stacks[0].StackStatus'
```

2. Initiera Docker Swarm

2.1 Hämta IP-adresser

```
aws cloudformation describe-stacks --stack-name swarm-ec2 --query
'Stacks[0].Outputs[*].[OutputKey,OutputValue]' --output table
```

Spara Manager och Workers publika/privata IPs.

2.2 Anslut till Manager

```
ssh -i Keyswarm1029.pem ec2-user@<manager-public-ip>
```

```
ssh -i Keyswarm1029.pem ec2-user@54.154.62.190

A newer release of "Amazon Linux" is available.
  Version 2023.9.20251027:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-1-34 ~]$
```

2.3 Initiera Swarm

```
sudo docker swarm init --advertise-addr <manager-private-ip>
```

Kopiera join-token som genereras.

```
[ec2-user@ip-172-31-1-34 ~]$ sudo docker swarm init --advertise-addr 172.31.1.34
Swarm initialized: current node (q521kpu631yylqlmfbnrw11d4) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-303kpn7av16ntcn1r23e7s6nd9b4m99r5cbk08cpjxz60r16jq-6fjqnyfux22mdtjlzd7pnxft5 172.31.1.34:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

2.4 Joina Workers

```
# Worker 1
ssh -i Keyswarm1029.pem ec2-user@<worker1-public-ip>
sudo docker swarm join --token <TOKEN> <manager-private-ip>:2377
exit

# Worker 2
ssh -i Keyswarm1029.pem ec2-user@<worker2-public-ip>
sudo docker swarm join --token <TOKEN> <manager-private-ip>:2377
exit
```

```
$ ssh -i keyswarm1029.pem ec2-user@34.245.181.76
The authenticity of host '34.245.181.76 (34.245.181.76)' can't be established.
ED25519 key fingerprint is SHA256:wqx+gnhyoBMmifYaadNZo9m6lafHbXpjOaS7qtGeq9o.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '34.245.181.76' (ED25519) to the list of known hosts.

A newer release of "Amazon Linux" is available.
  Version 2023.9.20251027:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-7-46 ~]$ SWMTKN-1-303kpn7av16ntcn1r23e7s6nd9b4m99r5cbk08cpjxz60r16jq-6fjqnyfux22mdtjlzd7pnxft5 172.31.1.34:2377
-bash: SWMTKN-1-303kpn7av16ntcn1r23e7s6nd9b4m99r5cbk08cpjxz60r16jq-6fjqnyfux22mdtjlzd7pnxft5: command not found
[ec2-user@ip-172-31-7-46 ~]$ sudo docker swarm join --token SWMTKN-1-303kpn7av16ntcn1r23e7s6nd9b4m99r5cbk08cpjxz60r16jq-6fjqnyfux22mdtjlzd7p
nxft5 172.31.1.34:2377
This node joined a swarm as a worker.
[ec2-user@ip-172-31-7-46 ~]$
```

2.5 Verifiera kluster

```
sudo docker node ls
```

```
[ec2-user@ip-172-31-1-34 ~]$ sudo docker node ls
ID                            HOSTNAME                                   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
q521kpu631yylqlmfbnrw11d4 *   ip-172-31-1-34.eu-west-1.compute.internal  Ready     Active         Leader           25.0.13
vbn9u2t338mbgsmnuy250irws     ip-172-31-7-46.eu-west-1.compute.internal  Ready     Active                          25.0.13
d56aulvpogoq01jt4xma214p5     ip-172-31-15-37.eu-west-1.compute.internal Ready     Active                          25.0.13
[ec2-user@ip-172-31-1-34 ~]$
```

3. Deploya test-stack (nginx)

3.1 Skapa deployment-skript

Skapade bash-skript som automatiserar deployment. Skriptet skapar docker-stack.yml och deployer automatiskt.

Komplett skript: `templates/docker-stack.sh`

3.2 Kör deployment

```
chmod +x deploy-swarm.sh
./deploy-swarm.sh
```

3.3 Resultat

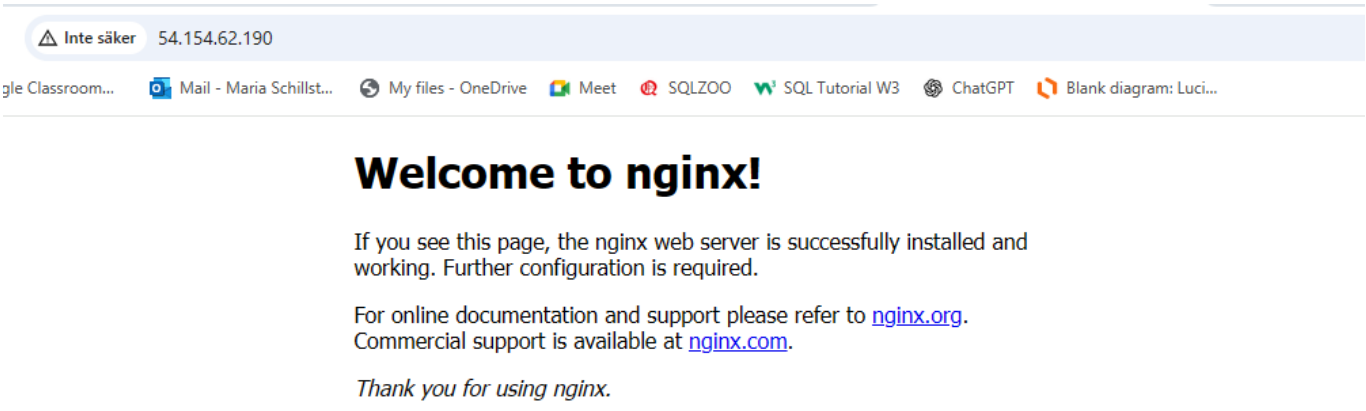
Stacken deployades med:

- 3 nginx replicas (fördelade av Manager över noderna)
- 1 visualizer replica (placement constraint: endast Manager)

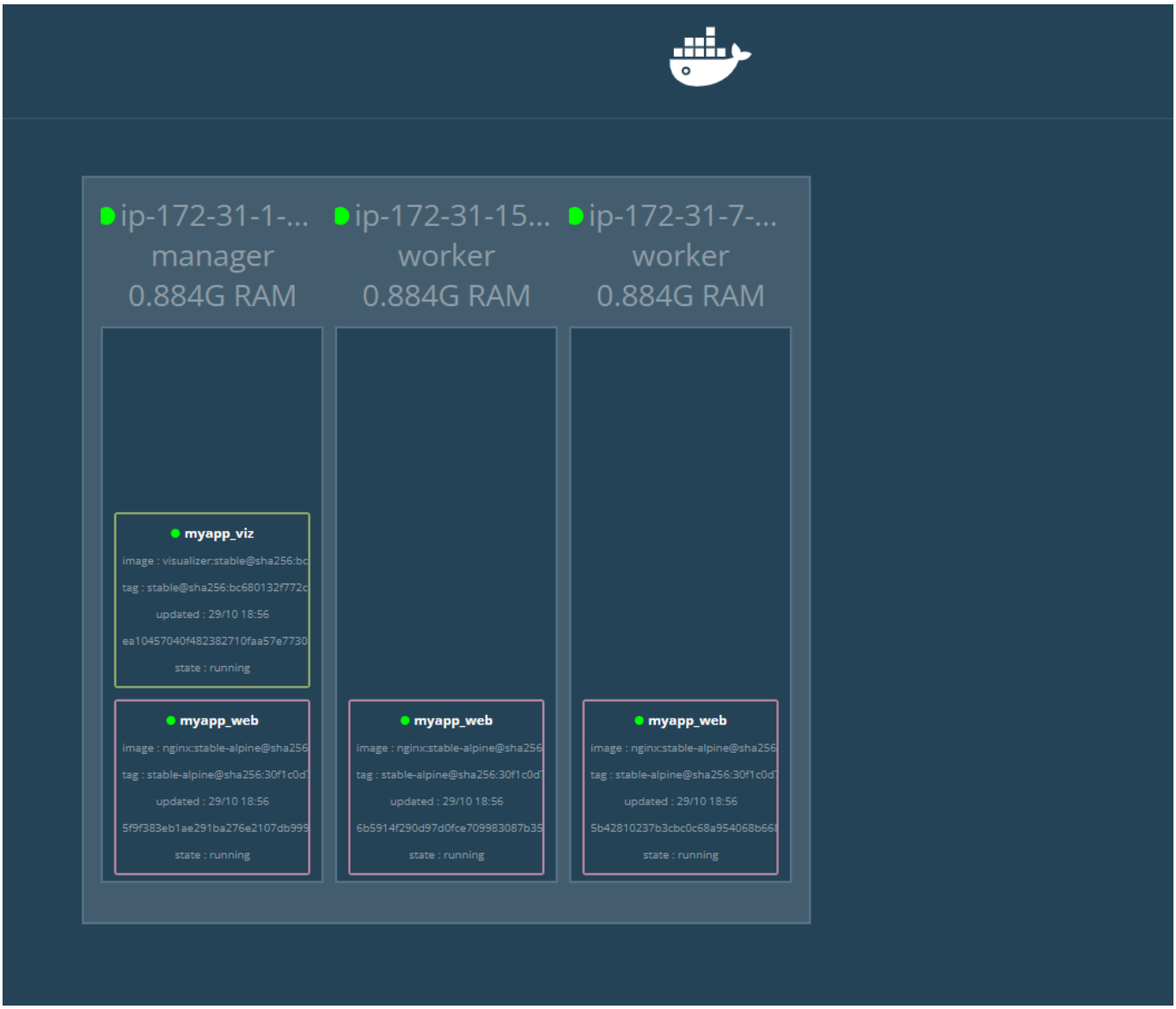
4. Testa och skala

4.1 Testa i webbläsare

Nginx: http:///



Visualizer: http://:8080/



4.2 Skala services

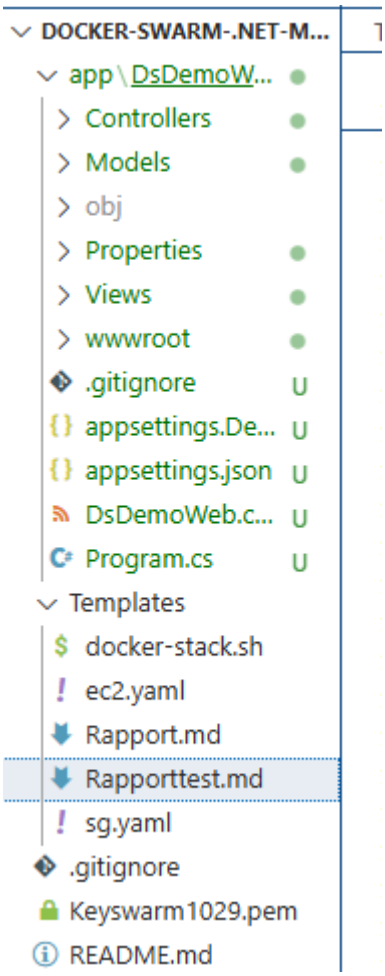
```
# Skala upp
sudo docker service scale myapp_web=5

# Skala ner
sudo docker service scale myapp_web=3
```


ip-172-31-1-...
manager
0.884G RAM
ip-172-31-15...
worker
0.884G RAM
ip-172-31-7-...
worker
0.884G RAM
myapp_viz
image : visualizer:stable@sha256:bc
tag : stable@sha256:bc680132f772c
updated : 29/10 18:56
ea10457040f482382710faa57e7730
state : running
myapp_web
image : nginx:stable-alpine@sha256
tag : stable-alpine@sha256:30f1c0d7
updated : 29/10 18:56
5f9f383eb1ae291ba276e2107db999
state : running
myapp_web
image : nginx:stable-alpine@sha256
tag : stable-alpine@sha256:30f1c0d7
updated : 29/10 18:56
6b5914f290d97d0fce709983087b35
state : running
myapp_web
image : nginx:stable-alpine@sha256
tag : stable-alpine@sha256:30f1c0d7
updated : 29/10 18:56
5b42810237b3cbc0c68a954068b66
state : running

5. Skapa och containerisera MVC-app

5.1 Skapa MVC-projekt

```
mkdir -p app && cd app
dotnet new mvc -n DsDemoWeb -o DsDemoWeb
cd DsDemoWeb
dotnet new gitignore
```



Kommentera bort HTTPS-redirection i `Program.cs`.

5.2 Skapa Dockerfile

```
# Build & publish
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
WORKDIR /src
COPY *.csproj ./
RUN dotnet restore
COPY . .
RUN dotnet publish -c Release -o /app/publish

# Runtime
```

```
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS production
WORKDIR /app
EXPOSE 80
ENV ASPNETCORE_URLS=http://+:80
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "DsDemoWeb.dll"]
```

5.3 Skapa ECR repository

```
AWS_REGION=eu-west-1
REPO=ds-demo-web

aws ecr create-repository --repository-name $REPO --region $AWS_REGION

ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REPO_URI=${ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/${REPO}
```

5.4 Logga in till ECR

```
aws ecr get-login-password --region eu-west-1 | docker login --username AWS --
password-stdin ${ACCOUNT_ID}.dkr.ecr.eu-west-1.amazonaws.com
```

```
maria@ADMINIS-S7QQ8L0 MINGW64 ~/source/repos/Docker-Swarm-.NET-MVC-application-/app/DsDemoWeb (Dev)
$ aws ecr get-login-password --region $AWS_REGION \
> AWS_REGION=eu-west-1
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
aws ecr get-login-password --region $AWS_REGION \
  | docker login --username AWS --password-stdin ${ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com

Unknown options: AWS_REGION=eu-west-1
Login Succeeded
```

6. Bygg och deploya MVC-appen

6.1 Bygg och pusha image

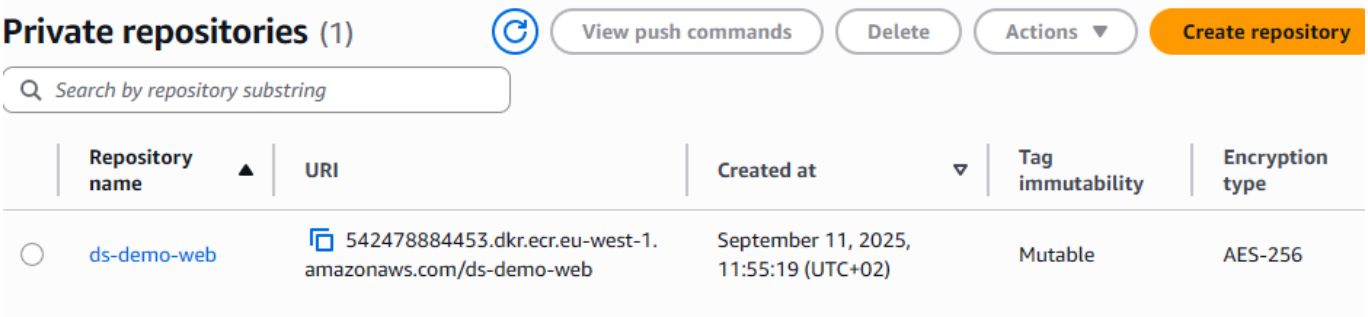
Problem: Multi-arch build (amd64+arm64) misslyckades med .NET 9 och QEMU (exit code 134).

Lösning: Byggde endast för amd64 eftersom alla EC2-instanser använder denna arkitektur.

```
docker buildx build \
  --platform linux/amd64 \
  -t ${REPO_URI}:v1 \
  -f Dockerfile \
  --push \
  .
```

```
=> => pushing manifest for 542478884453.dkr.ecr.eu-west-1.amazonaws.com/ds-demo-web:v1@sha256:4c568412b2e2118a7c9f9778f62920e678a148c251e0f5fa94366efda349f6b6
```

Verifiera i ECR:



6.2 Uppdatera deployment-skript

Bytte ut nginx-imagen mot MVC-app från ECR:

```
# Förut:
image: nginx:stable-alpine

# Nu:
image: 542478884453.dkr.ecr.eu-west-1.amazonaws.com/ds-demo-web:v1
```

6.3 Åtgärda ECR-access

Problem: EC2-instanserna saknade behörighet att hämta imagen från privat ECR.

Lösning:

1. **Raderade gamla EC2-stacken:**

```
aws cloudformation delete-stack --stack-name swarm-ec2
aws cloudformation wait stack-delete-complete --stack-name swarm-ec2
```

2. **Uppdaterade ec2.yaml** med IAM Role (`EC2-ECR-Access`) och Instance Profile

3. **Skapade ny stack med IAM:**

```
aws cloudformation create-stack \
  --stack-name swarm-ec2 \
  --template-body file://templates/ec2.yaml \
  --capabilities CAPABILITY_NAMED_IAM \
  --parameters ParameterKey=SubnetId,ParameterValue=subnet-058684dd8c601d55d
```

4. **Återskapade Swarm-klustret** (steg 2.2-2.5)

5. **Autentiserade Docker på alla noder** (IAM-rollen propagerade inte direkt):

```
# På alla 3 noder:
aws ecr get-login-password --region eu-west-1 | sudo docker login --username AWS -
-password-stdin 542478884453.dkr.ecr.eu-west-1.amazonaws.com
```

6.4 Deploya MVC-appen

```
# Kopiera uppdaterat skript
scp -i Keyswarm1029.pem templates/docker-stack.sh ec2-user@<manager-ip>:~/deploy-
swarm.sh

# SSH och kör
ssh -i Keyswarm1029.pem ec2-user@<manager-ip>
chmod +x deploy-swarm.sh
./deploy-swarm.sh
```

Verifiera:

```
sudo docker service ps myapp_web
sudo docker service logs myapp_web --tail 20
```

Testa i webbläsare:

- http:/// (MVC-app)
- http://:8080/ (Visualizer)

Resultat: MVC-appen körs med 3 replicas, koordinerade av Manager över alla noder! 🎉

Sammanfattning

Vad som skapats

En komplett, skalbar Docker Swarm-miljö på AWS med:

- Infrastructure as Code (CloudFormation)
- Automatiserad deployment (Bash-skript)
- Säker ECR-integration (IAM)
- Containeriserad .NET MVC-applikation
- 3 replicas för hög tillgänglighet
- Orchestrering där Manager koordinerar container-placering

Lärdomar

- **IaC är kraftfullt:** Hela miljön kan återskapas på minuter
- **IAM är kritiskt:** Behörigheter måste vara på plats för privata registries
- **Swarm är enkelt:** Mindre komplext än Kubernetes för små/medium setups
- **Orchestrering ger kontroll:** Manager's koordinering säkerställer optimal fördelning
- **Multi-stage builds:** Minskar image-storlek betydligt

Bilaga: Att använda IaC Generator

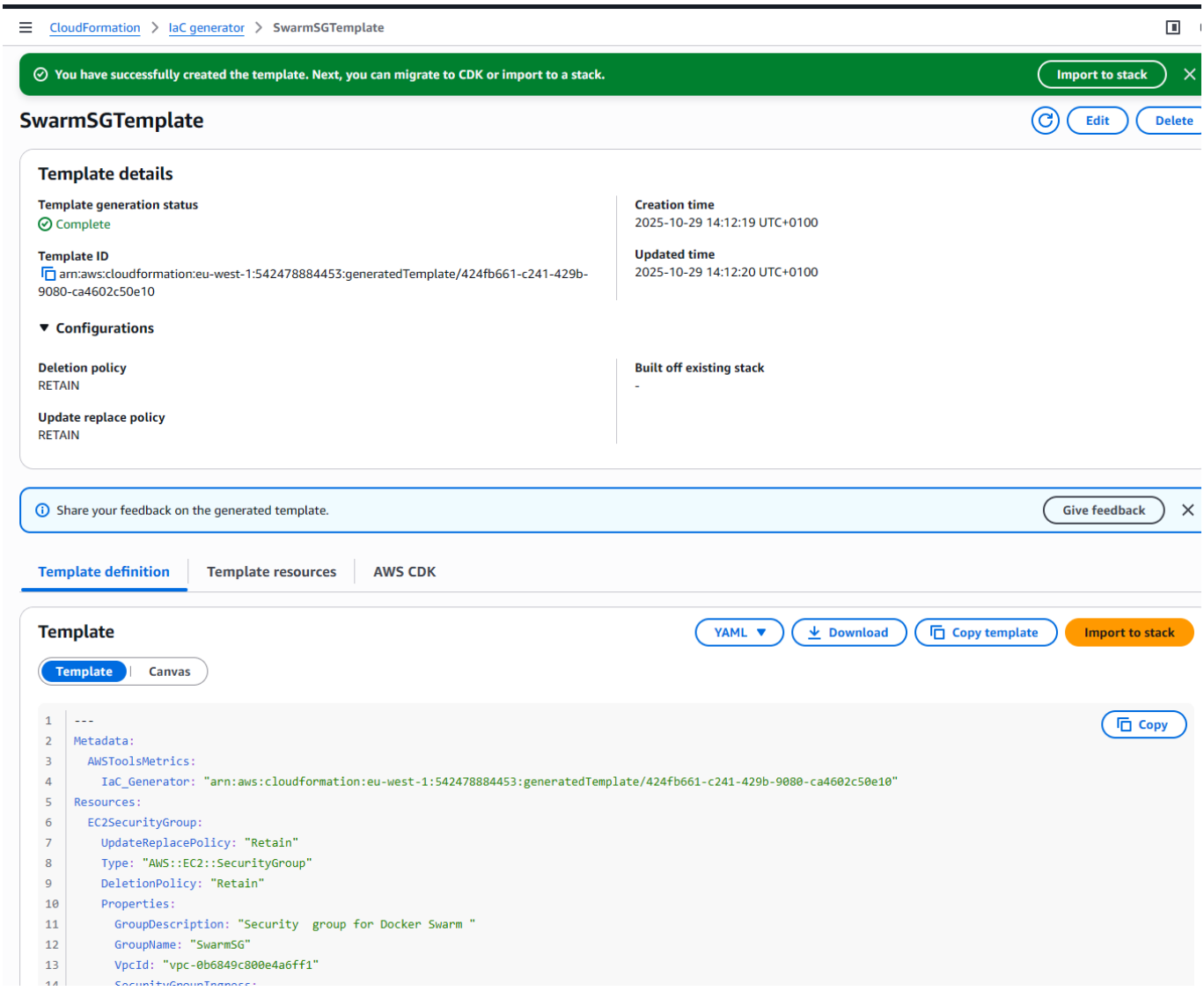
Som jag nämnde tidigare har jag använt CloudFormation för att skapa vissa resurser. Nedan följer ett exempel på hur man använder IaC Generator. Principen är densamma från det att resursen man vill använda till templaten är klar.

Steg 1: Skapa Security Group manuellt

1.1 Grundinställningar

Namnge Security Group och ange beskrivning.

Inbound rules:



- **VPC:** Default
- **SSH:** Port 22 (rekommenderas att använda Your IP address)
- **HTTP:** Port 80, Source: 0.0.0.0/0
- **Custom TCP (Visualizer):** Port 8080, Source: 0.0.0.0/0

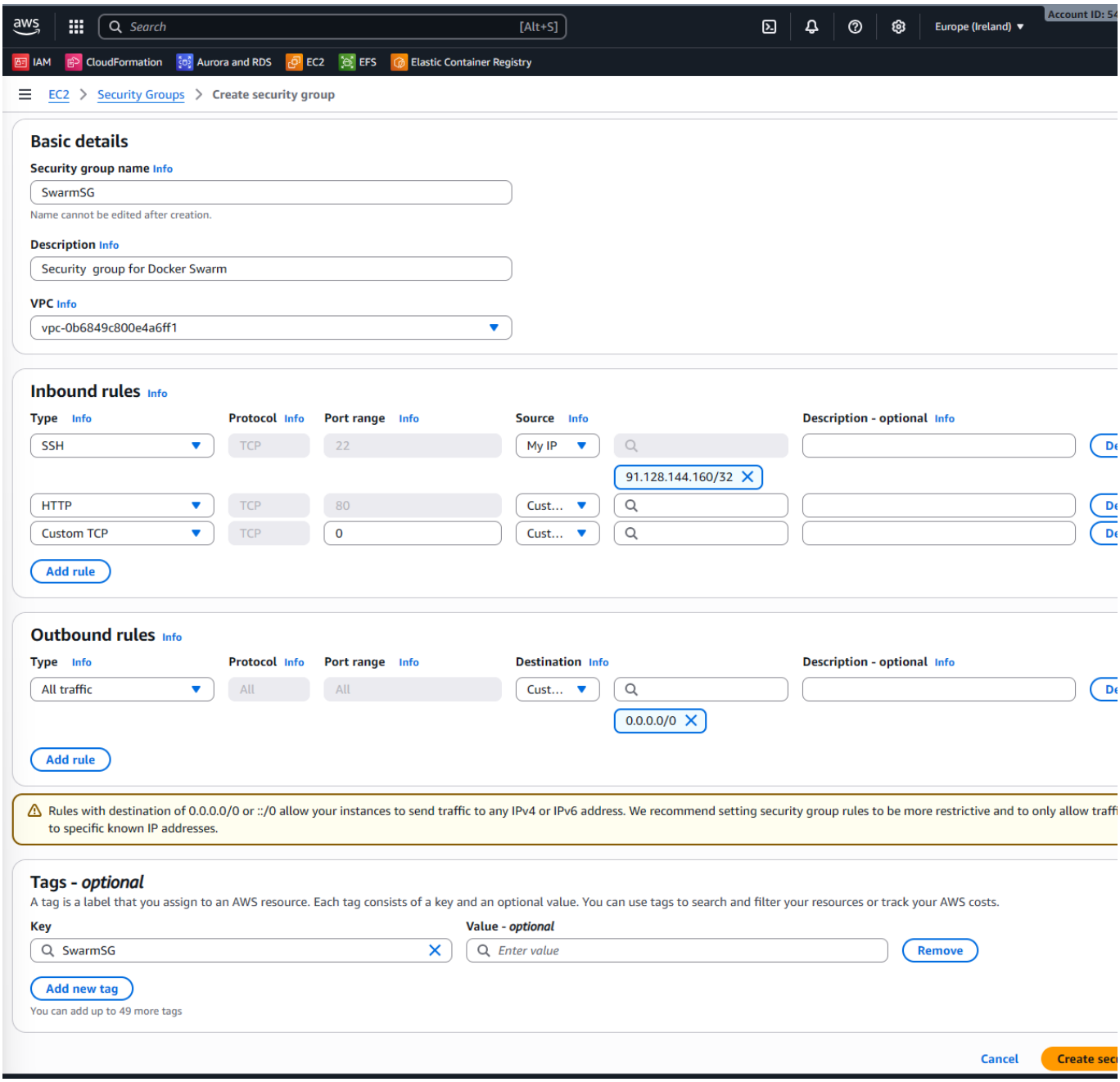
Jag glömde 8080 här, så fick göra det manuellt vid steg 4.1

Outbound rules:

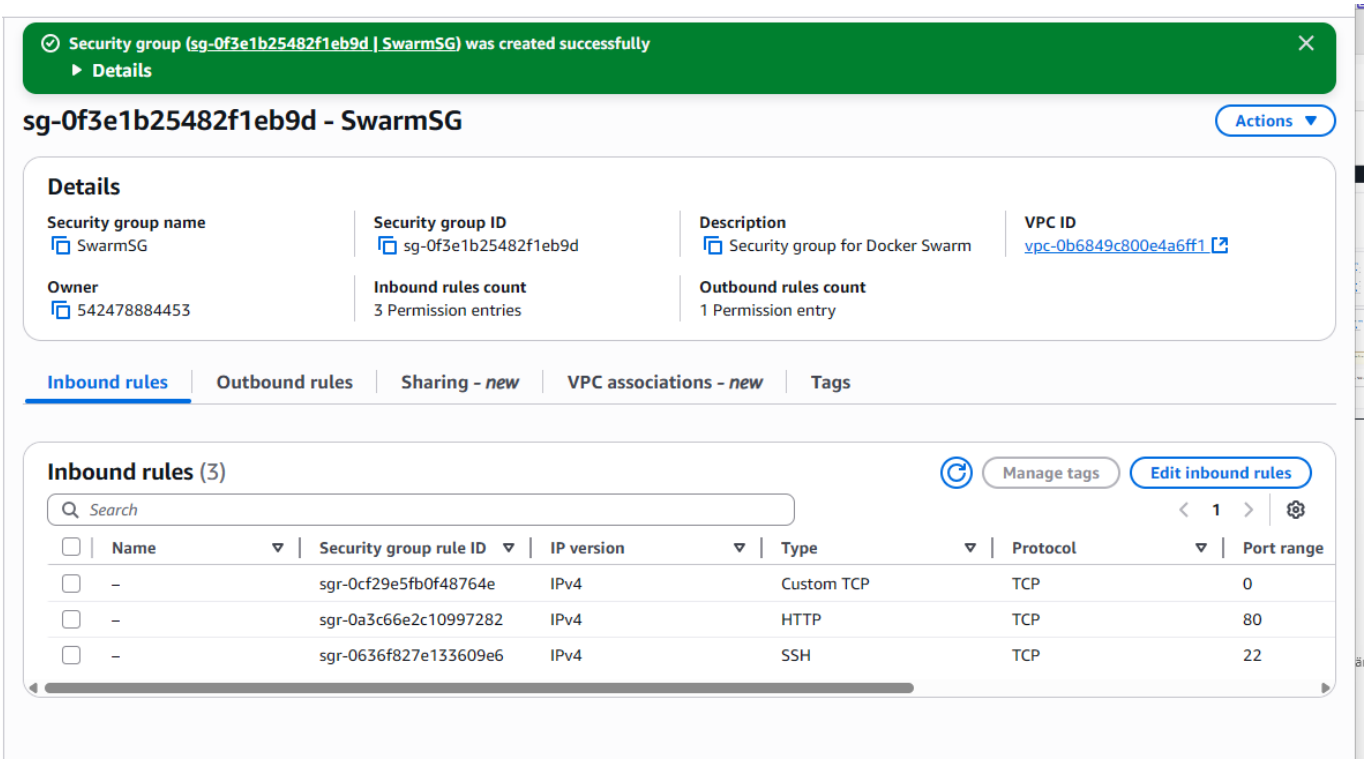
- All traffic, Destination: 0.0.0.0/0

Lägg till tags om så önskas.

Create security group



Då ska det se ut så här:



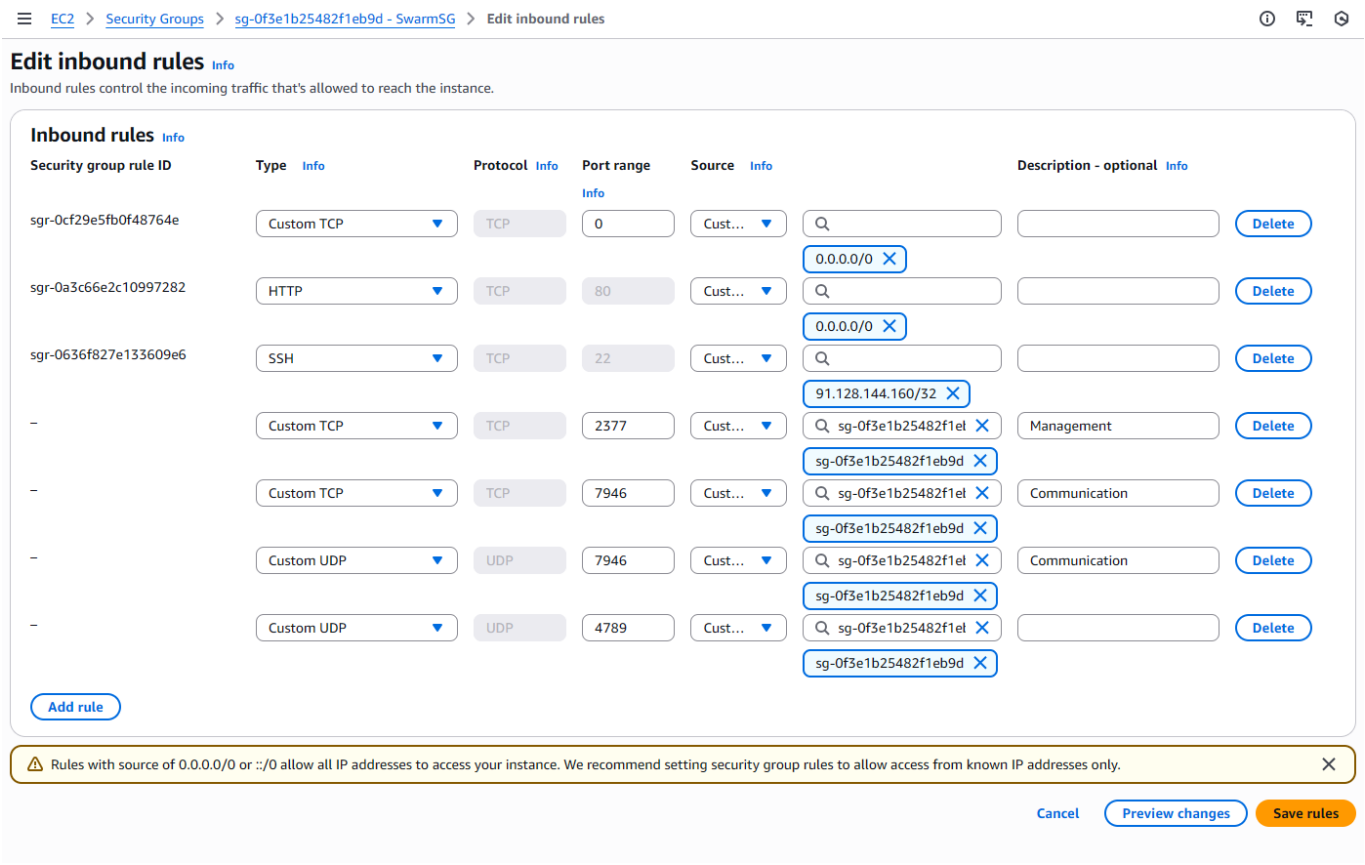
1.2 Self-reference för Swarm-kommunikation

Gå in på **Edit inbound rules**. Nu ska vi referera SG till sig själv för intern Swarm-kommunikation.

Lägg till följande regler som alla refererar till samma Security Group:

- **Swarm Management:** Port 2377 (TCP)
- **Swarm Communication:** Port 7946 (TCP & UDP)
- **Overlay Network:** Port 4789 (UDP)

Spara

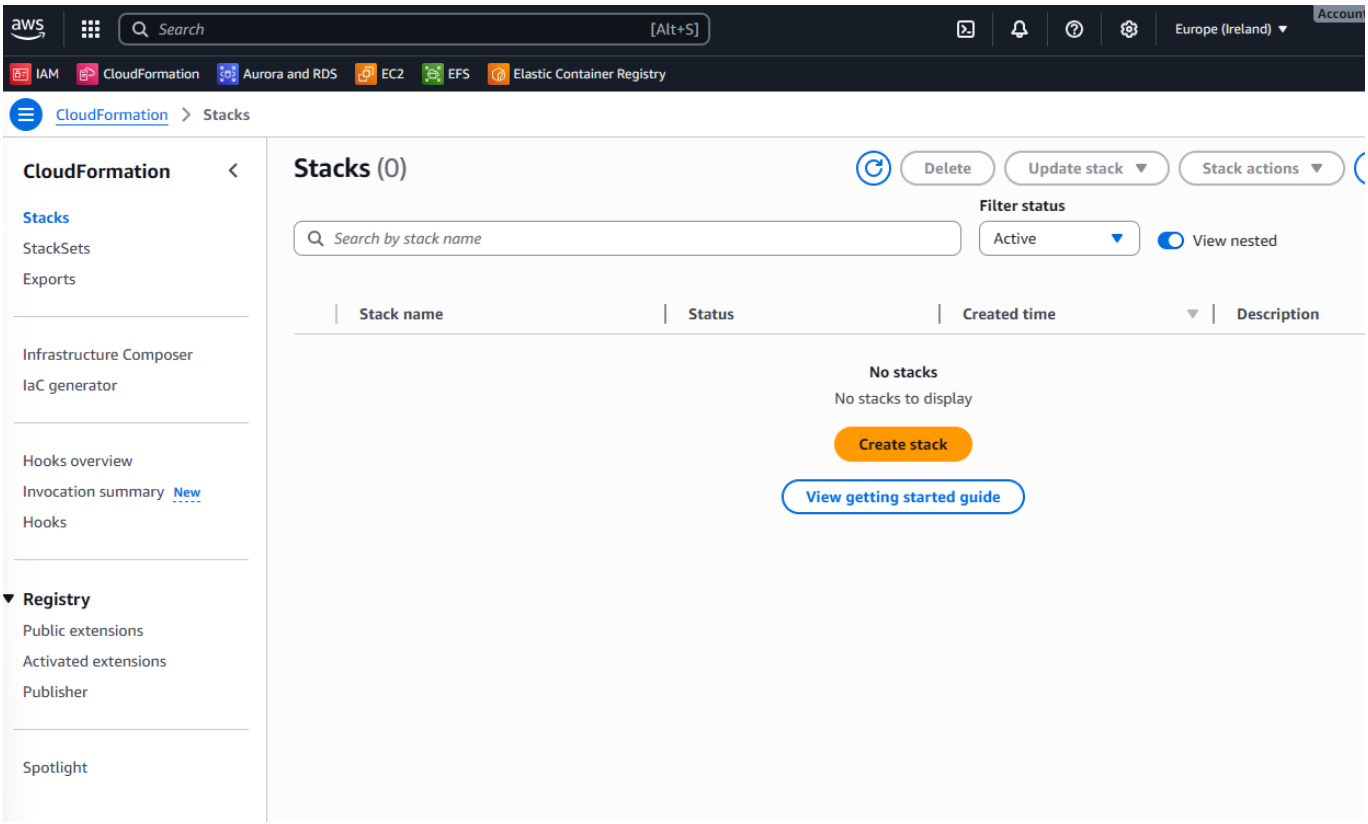


Tips: Skriv ner Resource identifier - den behövs till IaC Generator.

Steg 2: Skapa CloudFormation template med IaC Generator

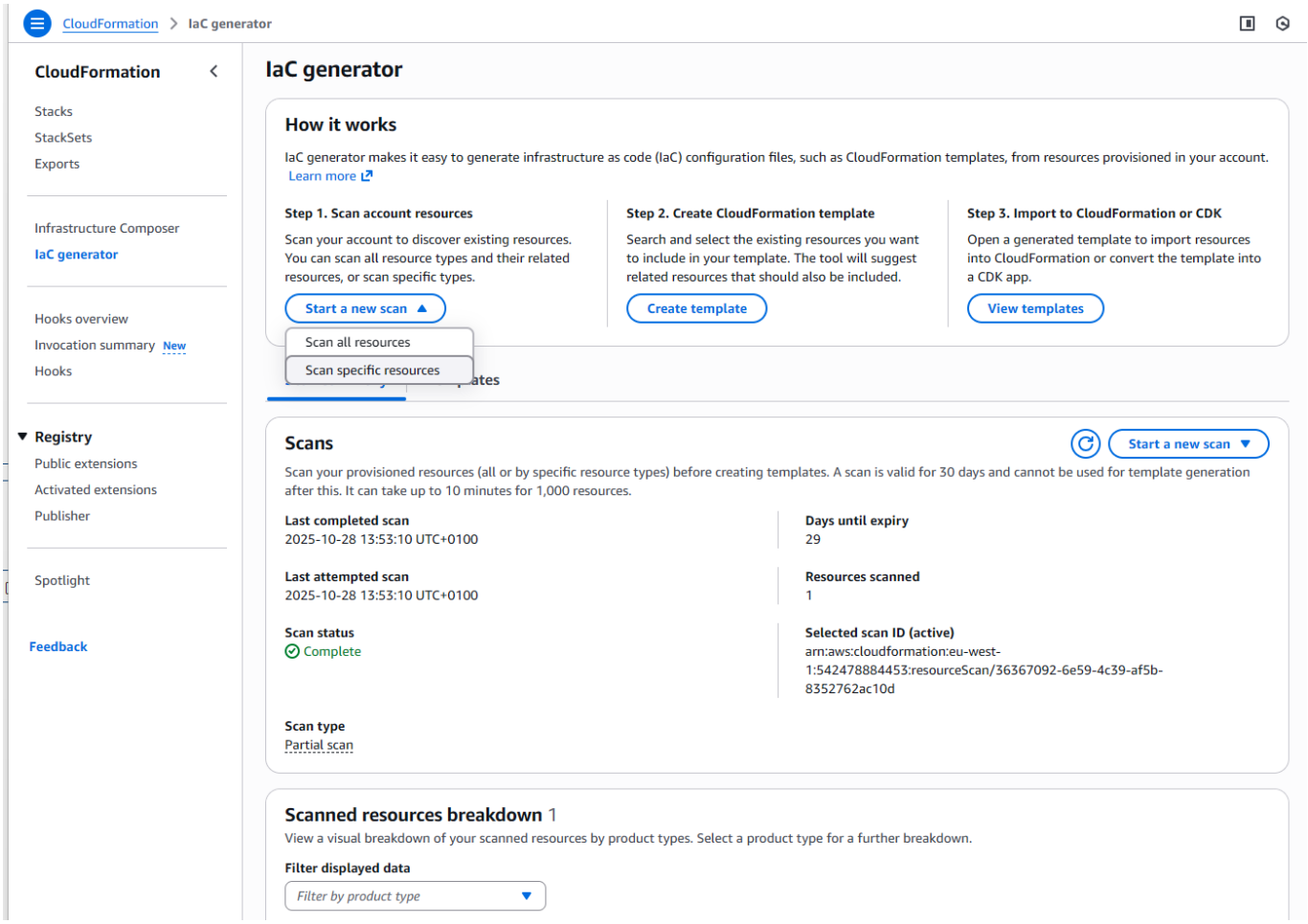
2.1 Navigera till IaC Generator

Gå till **CloudFormation → IaC Generator**



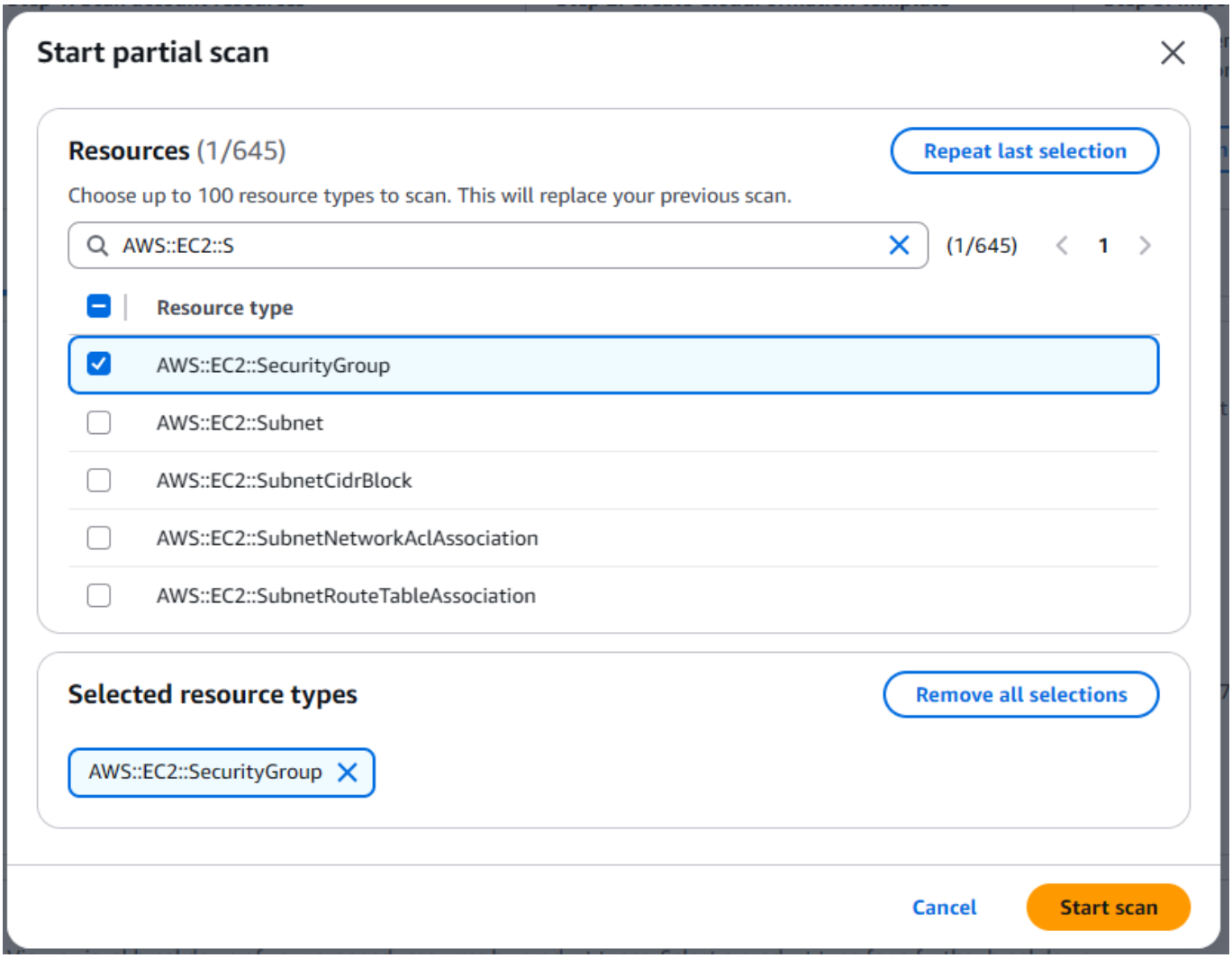
2.2 Starta scan

Starta en **new scan** och välj **Scan specific resource**



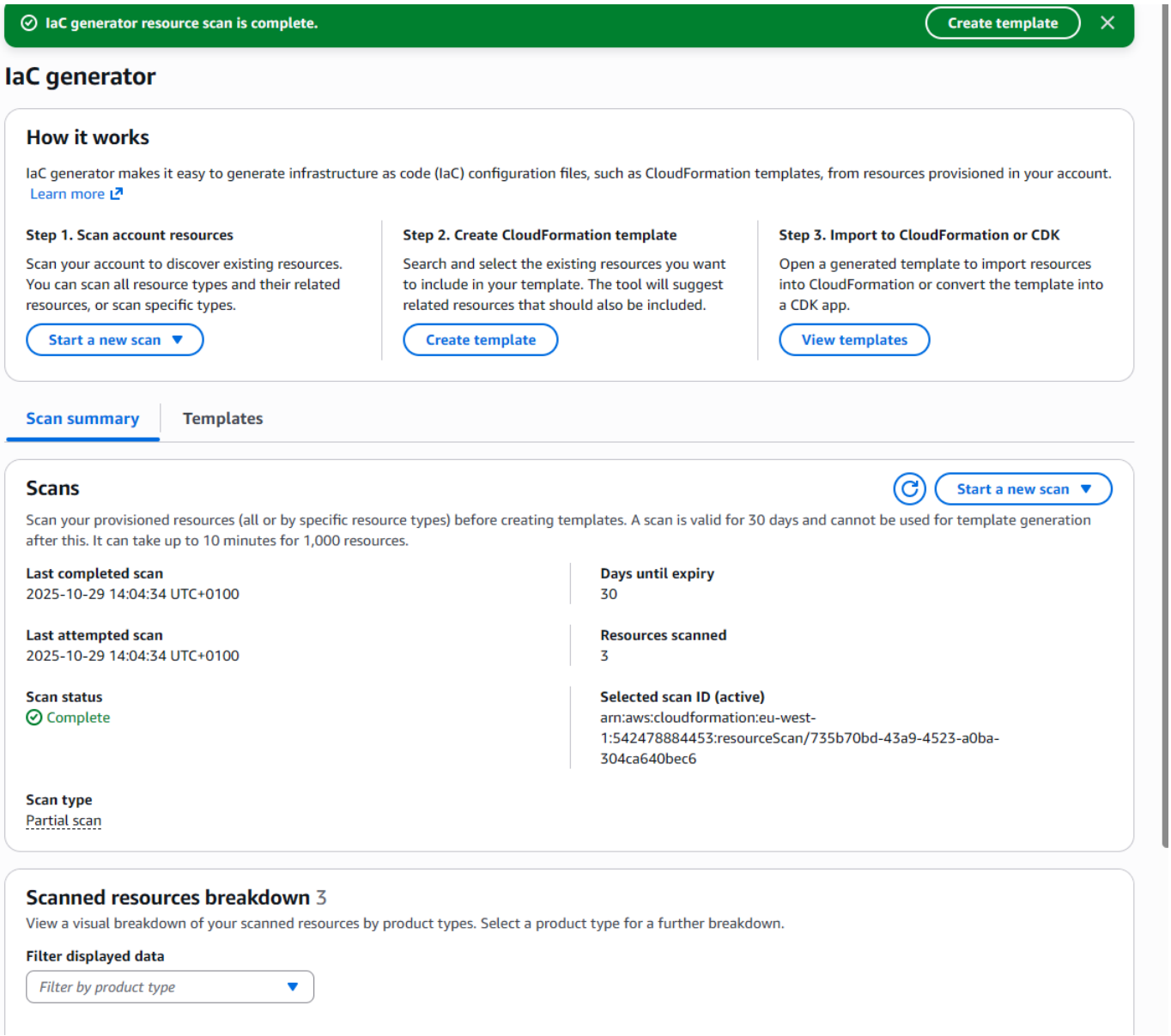
2.3 Välj Security Group

Sök efter SG, bocka i din Security Group och klicka **Start scan**



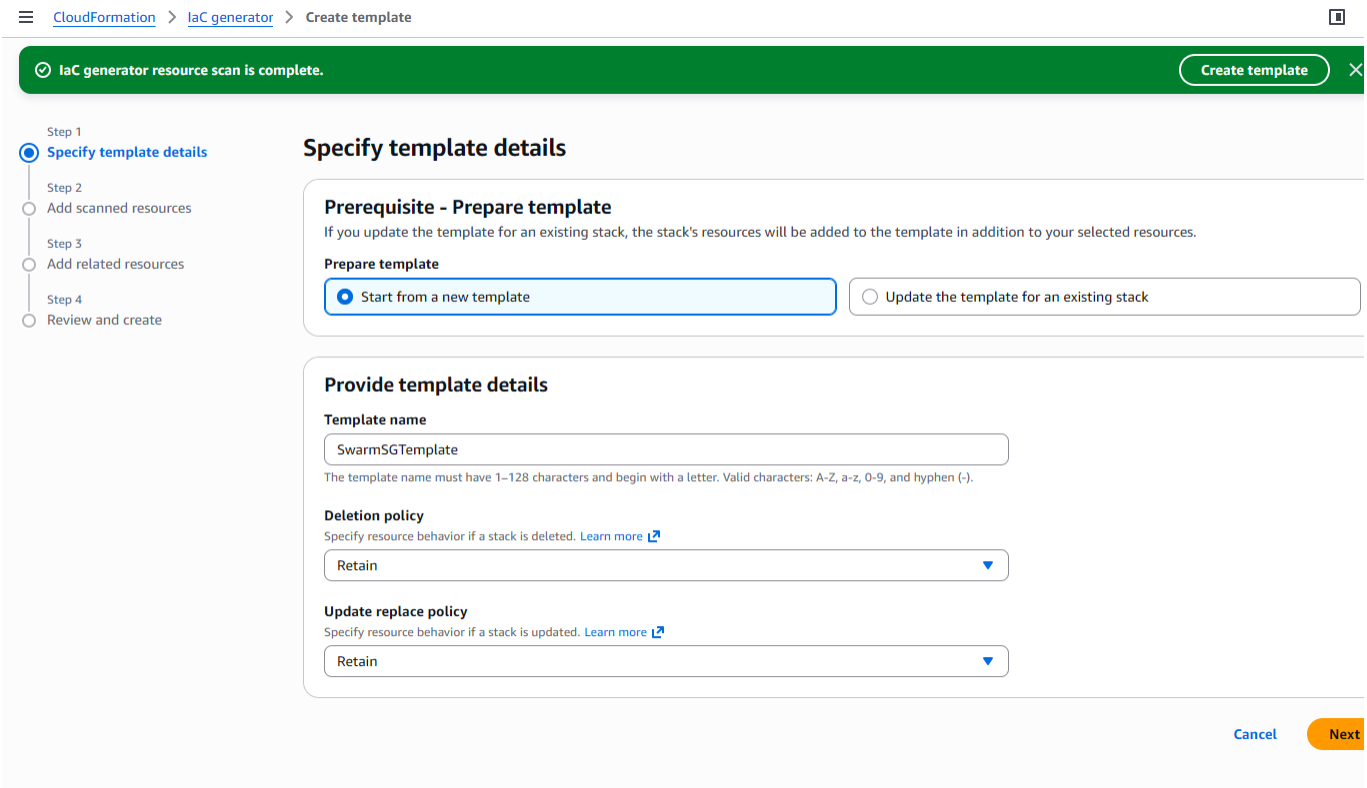
2.4 Skapa template

När scan är klar, klicka **Create template**



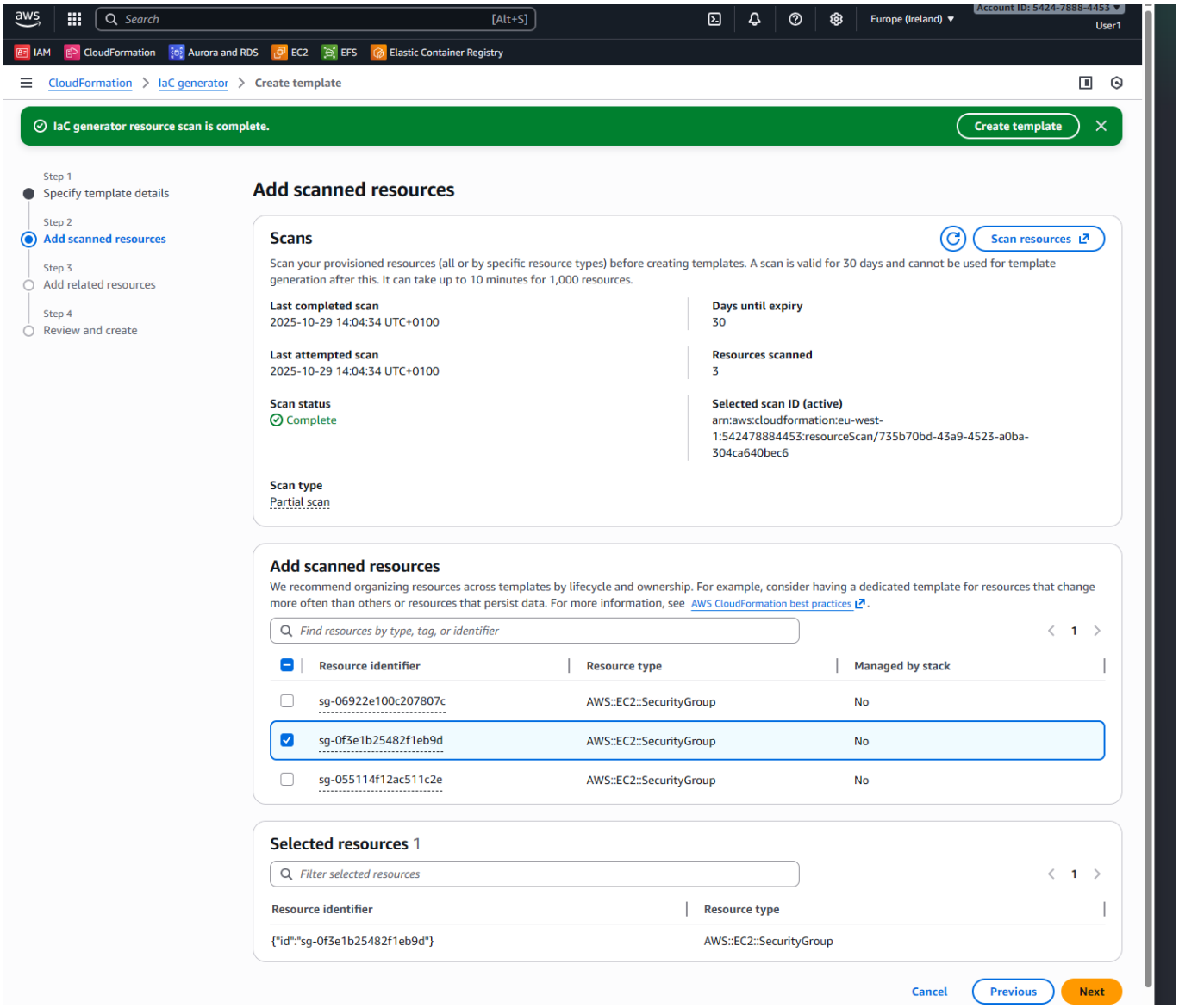
2.5 Namnge template

Namnge templaten och välj **Next**



2.6 Välj rätt resource

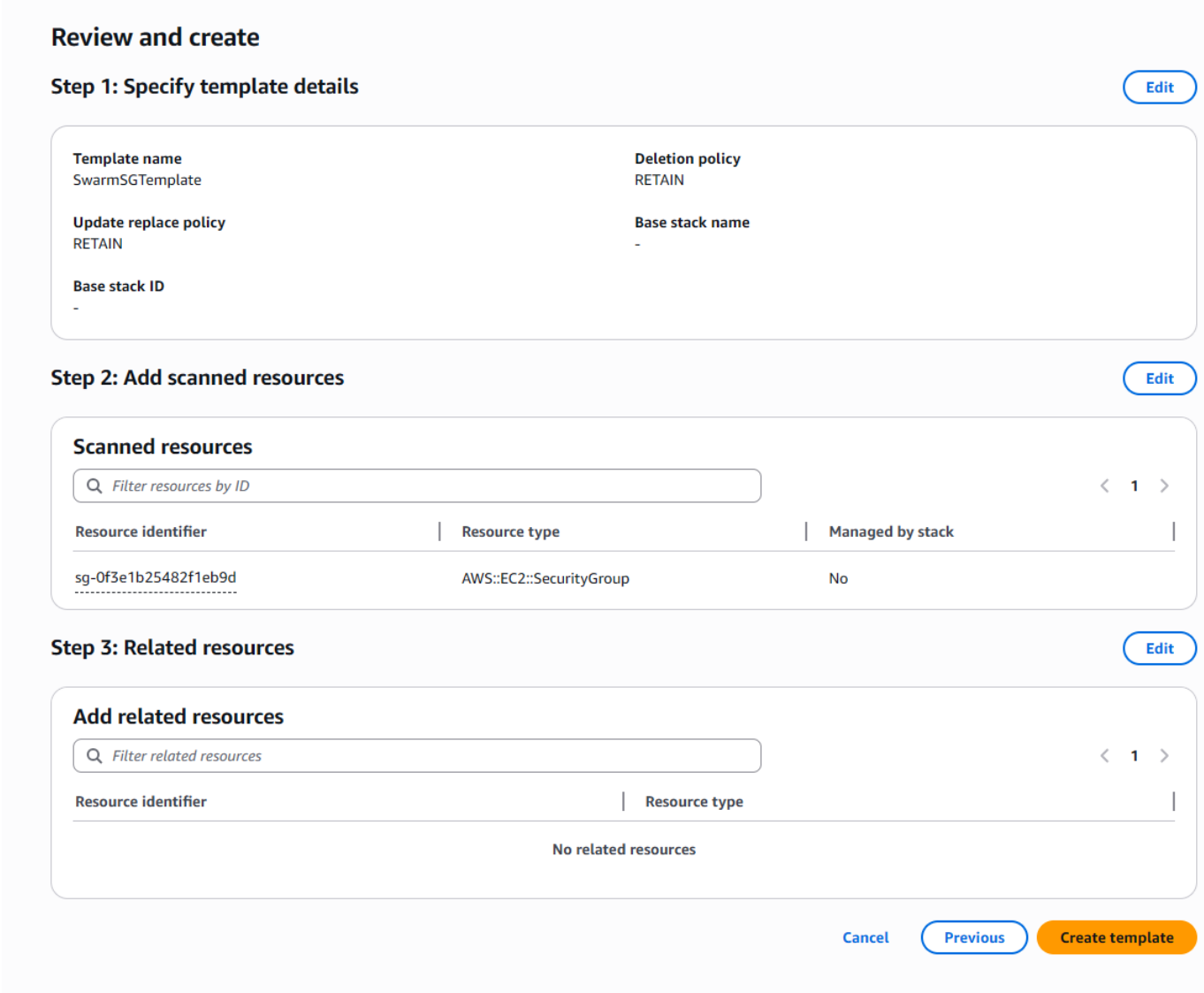
Nu får du upp alla SG du har skapat. Se till att välja rätt **Resource identifier**.



Tryck **Next** på kommande två fönster.

2.7 Slutför

Klicka **Create Template**

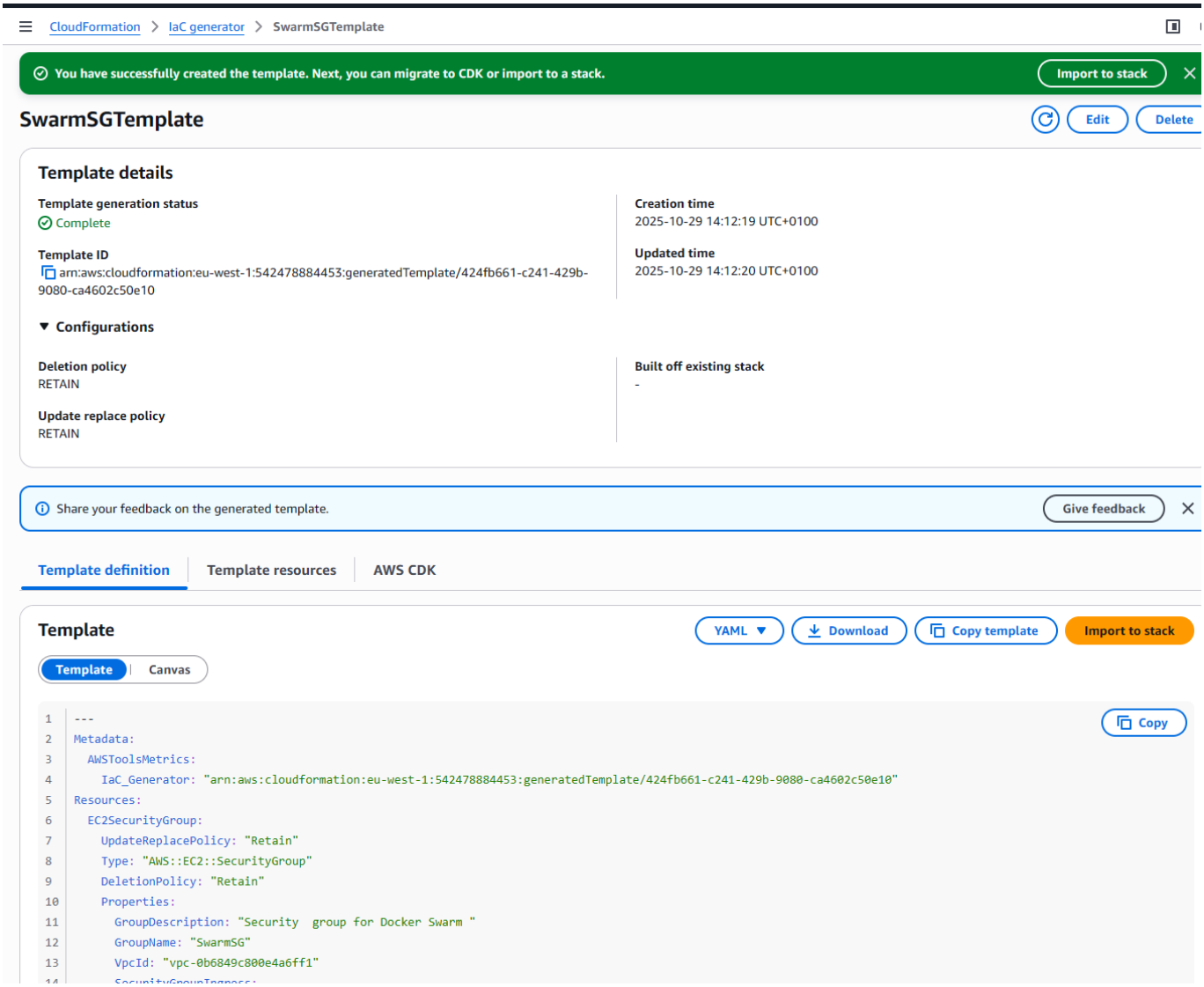


Steg 3: Parametrisera och spara

Nu har templaten skapats med ett långt skript.

Tips: Parametrisera skriptet för återanvändbarhet. Det kan vara knepigt i början, så då kan man be LLM om hjälp.

Välj att spara ner skriptet - då kan du använda det flera gånger som det är eller med ändringar.



OBS! Se till att radera stacken under CloudFormation samt SG innan du kör skriptet, eller uppdatera templaten med andra namn för att undvika konflikter.

Sammanfattning av IaC Generator-processen

1. ☑ Skapa resurs manuellt i AWS Console
2. ☑ Använd IaC Generator för att scanna resursen
3. ☑ Generera CloudFormation template
4. ☑ Parametrisera templaten
5. ☑ Spara och återanvänd för framtida deployments

Fördel: Får korrekt CloudFormation-syntax direkt från befintlig resurs, vilket minskar risken för fel och sparar tid.

Tips, deploya till swarm i produktionsmiljö, inte för utveckling