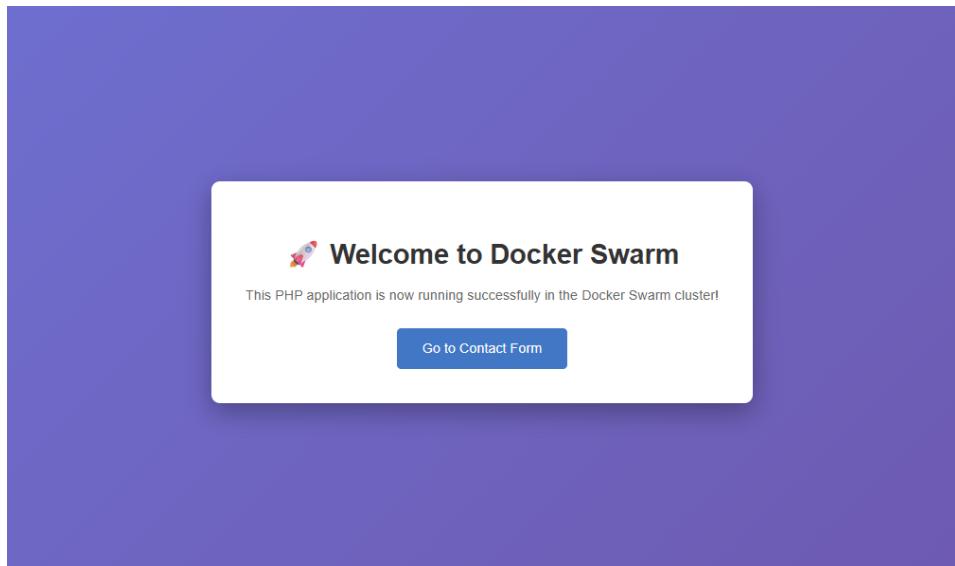


AWS - Inlämningsuppgift 2 - Max Oredson

Docker Swarm



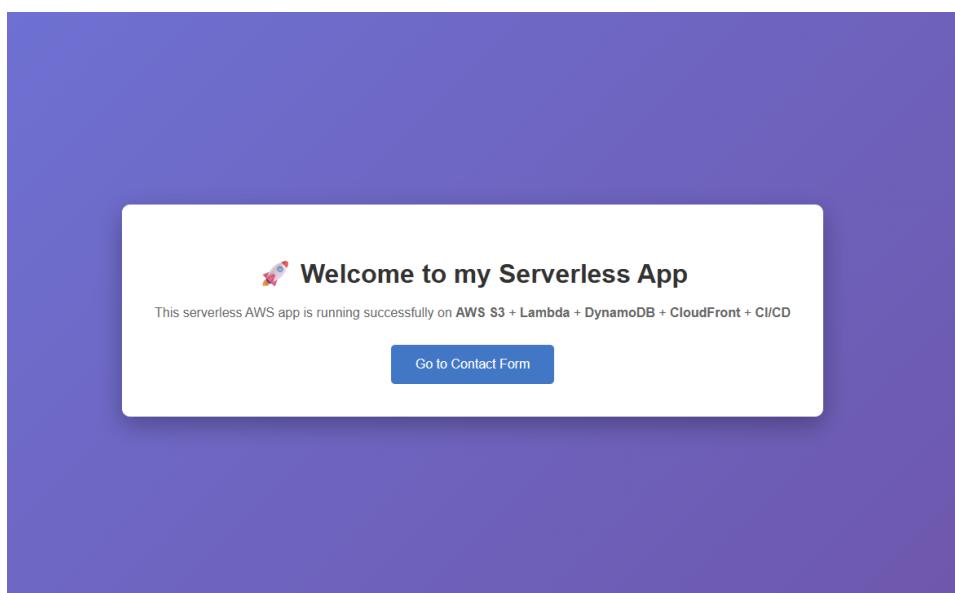
GitHub Repo: <https://github.com/91maxore-hub/docker-swarm-app>

<https://wavy.se>

Serverless App

GitHub Repo: <https://github.com/91maxore-hub/serverless-app>

<https://d3vbj5bvefx3w.cloudfront.net>



Docker Swarm

I detta projekt har jag byggt en skalbar och robust miljö för en webbapplikation med **Docker Swarm** på AWS. Miljön består av tre virtuella EC2-servrar, där en fungerar som manager och två som worker-noder. Applikationen, som är utvecklad med HTML, PHP och CSS, körs i tre separata containrar – en på varje server – vilket ger hög tillgänglighet och enkel skalning.

För att hantera inkommande trafik och säkerställa säkra anslutningar har jag implementerat **Traefik** som reverse proxy med stöd för HTTPS. För övervakning och visualisering av klustret används **Docker Visualizer**, vilket ger en tydlig överblick över vilka containrar som körs på vilka noder. Dessutom har jag kopplat CI/CD via **GitHub Actions**, vilket gör att uppdateringar av applikationen automatiskt byggs och distribueras till klustret.

Denna lösning visar hur containerteknologi och molninfrastruktur kan kombineras för att skapa en flexibel, skalbar och lättunderhållens webbmiljö, samtidigt som den säkerställer säkerhet, pålitlighet och tydlig översikt över klustrets status.

Noterbart är att i detta projekt har jag utnyttjat följande molntjänster från AWS:

- **EC2 (Elastic Compute Cloud):** Tre virtuella servrar används för att köra Docker Swarm – en som manager och två som worker-noder.
- **GitHub Actions:** För CI/CD, vilket möjliggör automatiska bygg och deployment av webbapplikationen.

Tillsammans skapar dessa tjänster en skalbar, flexibel och säker miljö för webbapplikationen.

Komponentöversikt och Syfte

Komponent	Beskrivning	Användningsområde	Kommentar
EC2-servrar	Virtuella servrar i AWS som utgör infrastrukturen för Docker Swarm-klustret	Körning av applikationens containrar och Swarm-noder	En server som manager, två som workers för skalbarhet och redundans
Docker Swarm	Containerorkesteringssystem som hanterar distribution av containrar	Säkerställer att applikationen körs i flera noder	Gör applikationen skalbar och tillgänglig även vid noder som går ner
Applikationscontainrar	Containeriserad webbapplikation (HTML, PHP, CSS)	Kör själva webbapplikationen på Swarm-servrarna	En container per server för hög tillgänglighet
Traefik	Reverse proxy och load balancer med HTTPS-stöd	Hantering av inkommande trafik och säkra anslutningar	Automatiserar certifikat via HTTPS och styr trafiken till rätt container
GitHub CI/CD	Automatiserat bygg- och deployflöde	Uppdateringar och deployment av applikationen	Säkerställer att nya versioner distribueras snabbt och pålitligt
Docker Visualizer	Grafiskt verktyg som visar status och fördelning av containrar i Swarm	Övervakning och visualisering av Swarm-klustret	Hjälper till att se vilka containrar som körs på vilka noder i realtid

Regler för säkerhetsgruppen docker-swarm-sg

Tillämpning / Resurser	Tillåtna portar	Protokoll	Syfte
EC2-servrar i Swarm-klustret	22	TCP	SSH-åtkomst för administration
EC2-servrar i Swarm-klustret	80	TCP	HTTP-trafik till webbapplikationen
EC2-servrar i Swarm-klustret	443	TCP	HTTPS-trafik till webbapplikationen
EC2-servrar i Swarm-klustret	4789	UDP	Overlay Network för Swarm-tjänster
EC2-servrar i Swarm-klustret	7946	TCP	Swarm intern kommunikation
EC2-servrar i Swarm-klustret	7946	UDP	Swarm intern kommunikation
EC2-servrar i Swarm-klustret	2377	TCP	Swarm management trafik mellan manager och workers
EC2-servrar i Swarm-klustret	8080	TCP	Traefik – reverse proxy med dashboard
EC2-servrar i Swarm-klustret	8081	TCP	Docker Visualizer dashboard

Mappstruktur

Katalog / Fil	Typ	Beskrivning
docker-swarm-app/	Mapp	Rotmappen för hela projektet.
└── Dockerfile	Fil	Bygger Docker-imagen och definierar miljö/beroenden.
└── index.html	Fil	Huvudsidan för webbapplikationen.
└── contact_form.html	Fil	Sida med kontaktformulär.
└── process_contact_form.php	Fil	PHP-script som hanterar formulärdata.
└── style.css	Fil	CSS-stilmall för webbplatsen.
└── .github/workflows/	Mapp	Mapp för GitHub Actions workflows.
└── deploy.yml	Fil	Workflow som hanterar CI/CD och deployment.

Provisionera Amazon EC2-server

Denna guide beskriver hur man provisionerar Amazon EC2-instanser som ska ingå i ett Docker Swarm-kluster. Målet är att skapa en stabil och skalbar miljö med en **Swarm Manager** och två **Swarm Workers**. EC2-instanserna kommer att konfigureras med nödvändig nätverksåtkomst, säkerhetsgrupper och grundläggande systemkrav för att stödja containerorkestrering med Docker Swarm.

Steg 1: Bege dig till aws.amazon.com

The screenshot shows the AWS Console Home page. At the top, there's a navigation bar with the AWS logo, a search bar, and account information. Below the navigation bar, the 'Console Home' section is visible. It includes a sidebar with 'Recently visited' services like EC2, CloudFront, S3, Lambda, etc. To the right, there are several cards: 'Applications' (0), 'Welcome to AWS' (with sections for Getting started with AWS, Training and certification, and AWS Builder Center), 'AWS Health' (with Open issues, Scheduled changes, and Other notifications), and 'Cost and usage' (showing current month cost of \$0.34, forecasted month end data unavailable, and savings opportunities). A large 'Go to myApplications' button is at the bottom right of the applications card.

Steg 2: Ange EC2 i sökrutan och välj "EC2 - Virtual Servers in the Cloud"

The screenshot shows the AWS Services page. On the left, there's a sidebar with links to Services, Features, Resources, Documentation, Knowledge articles, Marketplace, Blog posts, Events, and Tutorials. The main area is titled 'Services' and contains a card for 'EC2 Virtual Servers in the Cloud'. Below it are cards for 'EC2 Image Builder' (A managed service to automate build, customize and deploy OS images) and 'Recycle Bin' (Protect resources from accidental deletion). On the right, there are 'Show more' buttons for Services, Features, and a 'Dashboard' section. The search bar at the top is set to 'EC2'.

Steg 3: Välj "Launch Instance"

Steg 4: Ange ett namn för din server, operativsystem (AMI), instanstyp, samt skapa SSH-nyckel för säker åtkomst.

Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags [Info](#)

Name
swarm-manager [Add additional tags](#)

Application and OS Images (Amazon Machine Image) [Info](#)

An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose [Browse more AMIs](#).

Recents [Quick Start](#)

Amazon Linux	macOS	Ubuntu	Windows	Red Hat	SUSE Linux	Debian

Amazon Machine Image (AMI)

Amazon Linux 2023 kernel-6.1 AMI
ami-0870af38096a5355b (64-bit (x86), uefi-preferred) / ami-041f2b319c071bcb8 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

Description
Amazon Linux 2023 (kernel-6.1) is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Amazon Linux 2023 AMI 2023.9.20251110.1 x86_64 HVM kernel-6.1

Architecture	Boot mode	AMI ID	Publish Date	Username	Verified provider
64-bit (x86)	uefi-preferred	ami-0870af38096a5355b	2025-11-08	ec2-user	Verified provider

Summary
Number of instances [Info](#)
1

Software Image (AMI)
Amazon Linux 2023 AMI 2023.9.2... [read more](#)
ami-0870af38096a5355b

Virtual server type (instance type)
t3.small

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

[Launch instance](#)

6 / 64

Steg 5: Välj sedan säkerhetsgruppen (docker-swarm-sg) som ansvarar för vilka portar som ska användas för vårt Docker Swarm-kluster. Resten kan lämnas som det är.

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - **required**

swarm-manager-key ▼  [Create new key pair](#)

▼ Network settings [Info](#) [Edit](#)

Network [Info](#)
vpc-04a930fdf061e6a90

Subnet [Info](#)
No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)
Enable

Firewall (security groups) [Info](#)
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group Select existing security group

Common security groups [Info](#)
Select security groups ▼

docker-swarm-sg sg-05f60262028697eb7   [Compare security group rules](#)

Security groups that you add or remove here will be added to or removed from all your network interfaces.

▼ Configure storage [Info](#) [Advanced](#)

1x 8 GiB gp3 ▼ Root volume, 3000 IOPS, Not encrypted

[Add new volume](#)

 Click refresh to view backup information 
The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems [Edit](#)

► Advanced details [Info](#)

Steg 6: Gå sedan längst ner till Advanced details -> User data och klistra in följande:

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

User data - optional | [Info](#)

Upload a file with your user data or enter it in the field.

 [Choose file](#)

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

 User data has already been base64 encoded**Steg 6: Du får sedan en kort översikt över EC2-servern längst upp till höger. Välj "Launch instance"**

- Repetera nu likadant för **swarm-worker-1** och **swarm-worker-2**

▼ Summary**Number of instances** | [Info](#)

1

Software Image (AMI)Amazon Linux 2023 AMI 2023.9.2...[read more](#)

ami-0870af38096a5355b

Virtual server type (instance type)

t3.small

Firewall (security group)

docker-swarm-sg

Storage (volumes)

1 volume(s) - 8 GiB

[Cancel](#)[Launch instance](#) [Preview code](#)

Steg 7: Du bör nu se en översikt som nedan för samtliga EC2-servrar som kommer används i vårt Docker Swarm-kluster.

Instances (3) Info		Connect		Instance state		Actions		Launch instances		
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
<input type="checkbox"/>	swarm-worker-1	i-0122663740f818169	Running View Logs	t3.small	3/3 checks passed View alarms +	eu-west-1a	ec2-54-170-222-173.eu...	54.170.222.173	...	-
<input type="checkbox"/>	swarm-worker-2	i-064a3b5ef76da8f27	Running View Logs	t3.small	3/3 checks passed View alarms +	eu-west-1a	ec2-52-214-90-15.eu...	52.214.90.15	...	-
<input type="checkbox"/>	swarm-manager	i-0e76978203457f787	Running View Logs	t3.small	3/3 checks passed View alarms +	eu-west-1a	ec2-34-246-185-128.eu...	34.246.185.128	...	-

Skapandet av ett Docker Hub-repository

Innan vi initierar Docker Swarm-klustret är det viktigt att vi förbereder den applikations-image som klustret ska använda. Eftersom min app är PHP-baserad behöver projektet först paketeras i en Docker-image och publiceras på Docker Hub, så att Swarm-noderna kan dra ned samma version av imagen oavsett vilken nod som kör tjänsten.

När initieringen av Docker Swarm-klustret senare är klar kommer vi att kunna deploya stacken direkt från denna image. Men för att detta ska fungera måste vi först skapa ett repository på Docker Hub som ska lagra och distribuera Docker-imagen. I mitt fall döpte jag detta repository till **docker-swarm-app** (se bilden nedan).

Därefter behöver vi skapa en **Dockerfile** som använder en PHP-image med inbyggd webbserver, vi kommer att använda **php:8.2-apache**, och som kopierar in alla projektets filer. Denna image byggs sedan lokalt och pushas upp till Docker Hub så att den kan användas av hela Swarm-klustret vid deployment.

The screenshot shows the Docker Hub interface for the user '91maxore'. On the left, there's a sidebar with options like 'Repositories', 'Hardened Images', 'Collaborations', 'Settings', 'Default privacy', and 'Notifications'. The 'Repositories' tab is selected. In the main area, it says 'All repositories within the 91maxore namespace.' Below that is a search bar with 'Search by repository name' and a dropdown for 'All content'. A blue button on the right says 'Create a repository'. The repository list shows one entry: '91maxore/docker-swarm-app'. To the right of the repository name are columns for 'Name', 'Last Pushed', 'Contains', 'Visibility', and 'Scout'. The last push was 26 minutes ago, it contains an image, it's public, and its visibility is inactive.

Följ stegen nedan för att skapa ett **Docker Hub-repository**

Steg 1: Logga in på Docker Hub:

Gå till <https://hub.docker.com/repositories/ditt-användarnamn>

Steg 2: Navigera till dina repositories:

Du kommer direkt till listan över repositories under ditt konto.

The screenshot shows the Docker Hub user profile interface for the user '91maxore'. At the top, there's a blue profile icon, the username '91maxore', and the text 'Docker Personal'. Below this is a navigation bar with several options: 'Repositories' (which is highlighted in grey), 'Hardened Images', 'Collaborations', 'Settings', 'Default privacy', and 'Notifications'. The main area below the navigation bar lists the user's repositories.

Steg 3: Skapa ett nytt repository genom att klicka på "Create a Repository" längst bort till höger.

The screenshot shows the 'Repositories' page for the '91maxore' namespace. At the top, there's a search bar labeled 'Search by repository name', a dropdown menu set to 'All content', and a blue button labeled 'Create a repository'. Below this is a table listing the existing repository '91maxore/docker-swarm-app'. The table columns are 'Name', 'Last Pushed', 'Contains', 'Visibility', and 'Scout'. The repository details show it was last pushed 26 minutes ago, contains an image, is public, and is inactive.

Name	Last Pushed	Contains	Visibility	Scout
91maxore/docker-swarm-app	26 minutes ago	IMAGE	Public	Inactive

Steg 4: Fyll i repository-information:

- **Repository Name:** Ange ett namn för ditt repo, t.ex. `docker-swarm-app` kommer att bli `ditt-användarnamn/docker-swarm-app` senare när du ska bygga och pusha Docker-image
- **Visibility:** Välj om ditt repo ska vara **Public** eller **Private**
- **Description:** Lägg till en kort beskrivning om vad repot innehåller
- Klicka på "**Create**"

[Repositories](#) / [Create](#)

Create repository

Repository Name *

 Short description

A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. [Get more](#)

Public  Appears in Docker Hub search results

Private  Only visible to you

[Cancel](#) [Create](#)

Skapandet av Dockerfile

Jag skapade därefter en Dockerfile som använder PHP 8.2 med Apache och kopierar in mina applikationsfiler från projektmappen.

Kortfattat: en Dockerfile är en fil som beskriver hur ens Docker-image ska byggas.

Dockerfile (docker-swarm-app/Dockerfile) gör följande:

1. Använder officiell PHP 8.2 med Apache som grundimage.
2. Aktiverar Apache-modulen `mod_rewrite` för att möjliggöra URL-omskrivningar.
3. Kopierar alla applikationsfiler från projektmappen till Apache:s webbroot (`/var/www/html/`).
4. Exponerar port 80 så att webbservern kan ta emot HTTP-trafik.

Byggandet av Docker Image och uppladdning till Docker Hub

Nu är det dags att gå igenom stegen för att paketera projektet i en Docker-image och publicera den på Docker Hub

Steg 1: Byggandet av Docker Image

Jag använde terminalen i **Visual Studio Code** och angav följande kommando utifrån min projektmapp (där appens samtliga filer finns) för att bygga mina applikations-filer till en Docker-image och ge den en tagg.

91maxore = användarnamn

docker-swarm-app = repo på Docker Hub

```
docker build -t 91maxore/docker-swarm-app:latest .
```

Steg 2: Logga in på Docker Hub

Logga in på Docker Hub via terminalen:

```
docker login
```

- Angav mitt användarnamn och lösenord som jag använder till Docker Hub.

Steg 3: Pusha Docker-image till Docker Hub

När imagen är byggd och du är inloggad, pusha imagen till Docker Hub med:

```
docker push 91maxore/docker-swarm-app:latest
```

Detta pushar min nyskapade Docker-image till Docker Hub och är redo för användning.

Nu ligger den på Docker Hub:

🔗 <https://hub.docker.com/repository/docker/91maxore/docker-swarm-app>

- När man skapar eller uppdaterar en Docker Swarm-service skickar manager-noden instruktionen till alla workers. Om ens worker inte har den image-version som behövs, hämtar den automatiskt (pull) imagen från Docker Hub eller den angivna registry. Man behöver alltså inte göra pull manuellt på varje worker.

Initiering av Docker Swarm

För att initiera ett Docker Swarm-kluster, anslut till din swarm-manager via SSH och kör kommandot nedan som startar klustret och utser noden till manager.

Steg 1: Börja med att SSHa in på vår nyskapade EC2 genom att ange:

```
ssh -i ~/Downloads/swarm-manager-key.pem ec2-user@34.246.185.128
```

Notera att du får ändra sökvägen till din SSH-nyckel, samt den publika IP-adress till din EC2-instans

```
Max_Oredson@DESKTOP-SN4B6V8 MINGW64 ~/Downloads
$ ssh -i swarm-manager-key.pem ec2-user@34.246.185.128
,
~\_ ##### Amazon Linux 2023
~~ \##### https://aws.amazon.com/linux/amazon-linux-2023
~~ \|##|
~~ \#/ __ V~' '-->
~~ / \
~~ / \
~~ / \
Last login: Sun Nov 16 13:23:30 2025 from 80.217.195.172
[ec2-user@ip-172-31-23-10 ~]$
```

Steg 2: Initiera Docker Swarm-kluster på manager genom att köra följande:

```
docker swarm init --advertise-addr 34.246.185.128
```

Notera att du får byta ut IP-adressen mot den publika IP som din swarm-manager har

- Kopiera nu kommandot med dess token som skrivs ut för att ansluta våra övriga worker-noder till Docker Swarm-klustret, du bör få något som ser ut så här:

```
docker swarm join --token SWMTKN-1-
1qb2x87bw5wx75p5opwk8qqqoy513l2piskjrcze19acy8da3c-ec79bgjfs3q8doy3cpw3306js
172.31.23.10:2377
```

Anslutning och hantering av Swarm-workernoder

Steg 1: Kör nu följande för att ansluta swarm-worker-1 och swarm-worker-2 till Docker Swarm-klustret:

```
docker swarm join --token SWMTKN-1-  
1qb2x87bw5wx75p5opwk8qqoy513l2piskjrcze19acy8da3c-ec79bgjfs3q8doy3cpw3306js  
172.31.23.10:2377
```

- Notera att du får byta ut token.

Steg 2: Verifera sedan på swarm-manager att worker-noderna har lagts till i klustret genom att ange:

```
docker node ls
```

- Du bör då se något liknande:

```
[ec2-user@ip-172-31-23-10 ~]$ docker node ls  
ID           HOSTNAME   STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION  
i8xm23liainh482l06otg9ts  ip-172-31-16-170.eu-west-1.compute.internal  Ready  Active  
so3r924ftotfuias57fz17q8s9 * ip-172-31-23-10.eu-west-1.compute.internal  Ready  Active  Leader  25.0.13  
z2io6pjcurxxdn4iq896zun8e  ip-172-31-30-160.eu-west-1.compute.internal  Ready  Active  
[ec2-user@ip-172-31-23-10 ~]$ |
```

Detta bekräftar att vårt Docker Swarm-kluster är nu skapad med 1 manager och 2 workers.

Steg 3: Skapa Docker Stack-fil

- En **docker-stack.yml** behövs för att definiera hela applikationens tjänster, nätverk och inställningar på ett och samma ställe, så att Docker Swarm kan deployna och hantera allt som en komplett stack.
- På swarm-managern, skapa stackfilen **docker-stack.yml** och klistra in följande:

```
version: "3.8"
services:
  web:
    image: 91maxore/docker-swarm-app:latest
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
      update_config:
        parallelism: 1
        delay: 5s
    ports:
      - "80:80"
    networks: [webnet]

networks:
  webnet:
    driver: overlay
```

Beskrivning (Webbapplikationen)

- Kör **91maxore/docker-swarm-app** som en Swarm-tjänst med 3 repliker.
- Startar om repliker automatiskt vid fel.
- Exponerar tjänsten på port 80.
- Använder overlay-nätverk (**webnet**) så att tjänsten kan kommunicera med andra tjänster i klustret.

Steg 4: Distribuera Docker Swarm-stacken genom att ange följande:

```
sudo docker stack deploy -c docker-stack.yml docker-swarm-app
```

- **docker-swarm-app** blir namnet på stacken eftersom vår stack kommer i slutändan innehålla flera tjänster: **web**, **viz** och **traefik**
- Samtliga tjänster kommer befina sig på följande benämningar: **docker-swarm-app_web**, **docker-swarm-app_viz** och **docker-swarm-app_traefik**

Steg 5: Vi kan nu kontrollera statusen för varje instans av webbapplikationen, se på vilken nod de körs och verifiera att alla tre repliker fungerar som de ska. Detta görs med följande kommando:

```
sudo docker service ps docker-swarm-app_web
```

- Webbapplikationen kör nu stabilt och som förväntat på alla tre noder i Swarm-klustret, vilket bekräftar att deploymenten fungerar korrekt.
- Kort sagt: bilden visar var och hur min web-app körs inom Swarm-klustret
- Det fungerar på samma sätt genom att senare ange **docker-swarm-app_viz** för **Docker Vizualizer** och **docker-swarm-app_traefik** för **Traefik**.

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
11nfs1kb6e55	docker-swarm-app_web.1	91maxore/docker-swarm-app:latest	ip-172-31-16-170.eu-west-1.compute.internal	Running	Running 3 hours ago
uzauums3ylql	└ docker-swarm-app_web.1	91maxore/docker-swarm-app:latest	ip-172-31-16-170.eu-west-1.compute.internal	Shutdown	Shutdown 3 hours ago
8dr111ygxdo	└ docker-swarm-app_web.1	91maxore/docker-swarm-app:latest	ip-172-31-16-170.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
lkppzu3ch5qb	└ docker-swarm-app_web.1	91maxore/docker-swarm-app:latest	ip-172-31-16-170.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
6ae01knmtwlv	└ docker-swarm-app_web.1	91maxore/docker-swarm-app:latest	ip-172-31-16-170.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
72ache0sr6ue	docker-swarm-app_web.2	91maxore/docker-swarm-app:latest	ip-172-31-30-160.eu-west-1.compute.internal	Running	Running 3 hours ago
urgbnh8wkzs0	└ docker-swarm-app_web.2	91maxore/docker-swarm-app:latest	ip-172-31-30-160.eu-west-1.compute.internal	Shutdown	Shutdown 3 hours ago
vo7u6g2ordcv	└ docker-swarm-app_web.2	91maxore/docker-swarm-app:latest	ip-172-31-30-160.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
u863pxssfrfz	└ docker-swarm-app_web.2	91maxore/docker-swarm-app:latest	ip-172-31-30-160.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
x7nit75ka45x	└ docker-swarm-app_web.2	91maxore/docker-swarm-app:latest	ip-172-31-30-160.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
zdiwjgj1ly33q	docker-swarm-app_web.3	91maxore/docker-swarm-app:latest	ip-172-31-23-10.eu-west-1.compute.internal	Running	Running 3 hours ago
zgh5gjmzrn38	└ docker-swarm-app_web.3	91maxore/docker-swarm-app:latest	ip-172-31-23-10.eu-west-1.compute.internal	Shutdown	Shutdown 3 hours ago
ry3t9j0z13w	└ docker-swarm-app_web.3	91maxore/docker-swarm-app:latest	ip-172-31-23-10.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
urkp6upvb8qs	└ docker-swarm-app_web.3	91maxore/docker-swarm-app:latest	ip-172-31-23-10.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago
pr1omrlugnvq	└ docker-swarm-app_web.3	91maxore/docker-swarm-app:latest	ip-172-31-23-10.eu-west-1.compute.internal	Shutdown	Shutdown 4 hours ago

Steg 6: Vi kan även verifiera att swarm-worker 1 och swarm-worker 2 tagit del av samma docker-image och att webbapplikationen är replikerad utöver alla 3 noder med följande kommando:

```
docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
qpwqobw27qe2	docker-swarm-app_traefik	replicated	1/1	traefik:v2.11.3	*:80->80/tcp, *:443->443/tcp, *:8080->8080/tcp
4ko4udt8rfm	docker-swarm-app_viz	replicated	1/1	dockersamples/visualizer:stable	*:8081->8080/tcp
v9h3cbe9drc0	docker-swarm-app_web	replicated	3/3	91maxore/docker-swarm-app:latest	

- Som du kan se kör mitt Docker Swarm-kluster även **Traefik** för reverse proxy och HTTPS-hantering, och detta kommer jag att gå igenom mer detaljerat senare i guiden.
- Dessutom kör mitt Docker Swarm-kluster även **Docker Visualizer** för att enkelt kunna se noder, tjänster och containrar i realtid, och detta kommer jag att gå igenom mer detaljerat i nästa steg.

Docker Vizualiser

Docker Visualizer är ett verktyg som ger en grafisk översikt över ditt Docker Swarm-kluster. Det visar alla noder, både manager och worker, samt vilka containrar som körs på respektive nod i realtid. Vizualizer är ett utmärkt sätt att snabbt förstå klustrets struktur, övervaka distributionen av tjänster och kontrollera att skalning och repliker fungerar som förväntat.

Steg 1: Börja med att addera följande till docker-stack.yml som vi skapade tidigare för att lägga till Vizualiser som tjänst till vår stack:

- Eftersom **Docker Vizualiser** är en tjänst listar vi även den under **services** som nedan.

```
services:  
  viz:  
    image: dockersamples/visualizer:stable  
    deploy:  
      placement:  
        constraints:  
          - node.role == manager  
    ports:  
      - "8081:8080"                      # Visualizer-webbgränssnitt  
    volumes:  
      - /var/run/docker.sock:/var/run/docker.sock  
    networks:  
      - webnet  
  
networks:  
  webnet:  
    driver: overlay
```

Beskrivning (Docker Vizualizer)

- Kör Visualizer som en Swarm-tjänst på manager-noden.
- Mountar Docker-socket för att kunna läsa klustrets noder och containrar.
- Exponerar Visualizer på port 8081
- Använder overlay-nätverk så den kan kommunicera med andra tjänster om det behövs.

Steg 2: Deploya återigen stacken genom att köra detta på manager-noden:

```
docker stack deploy -c docker-stack.yml docker-swarm-app
```

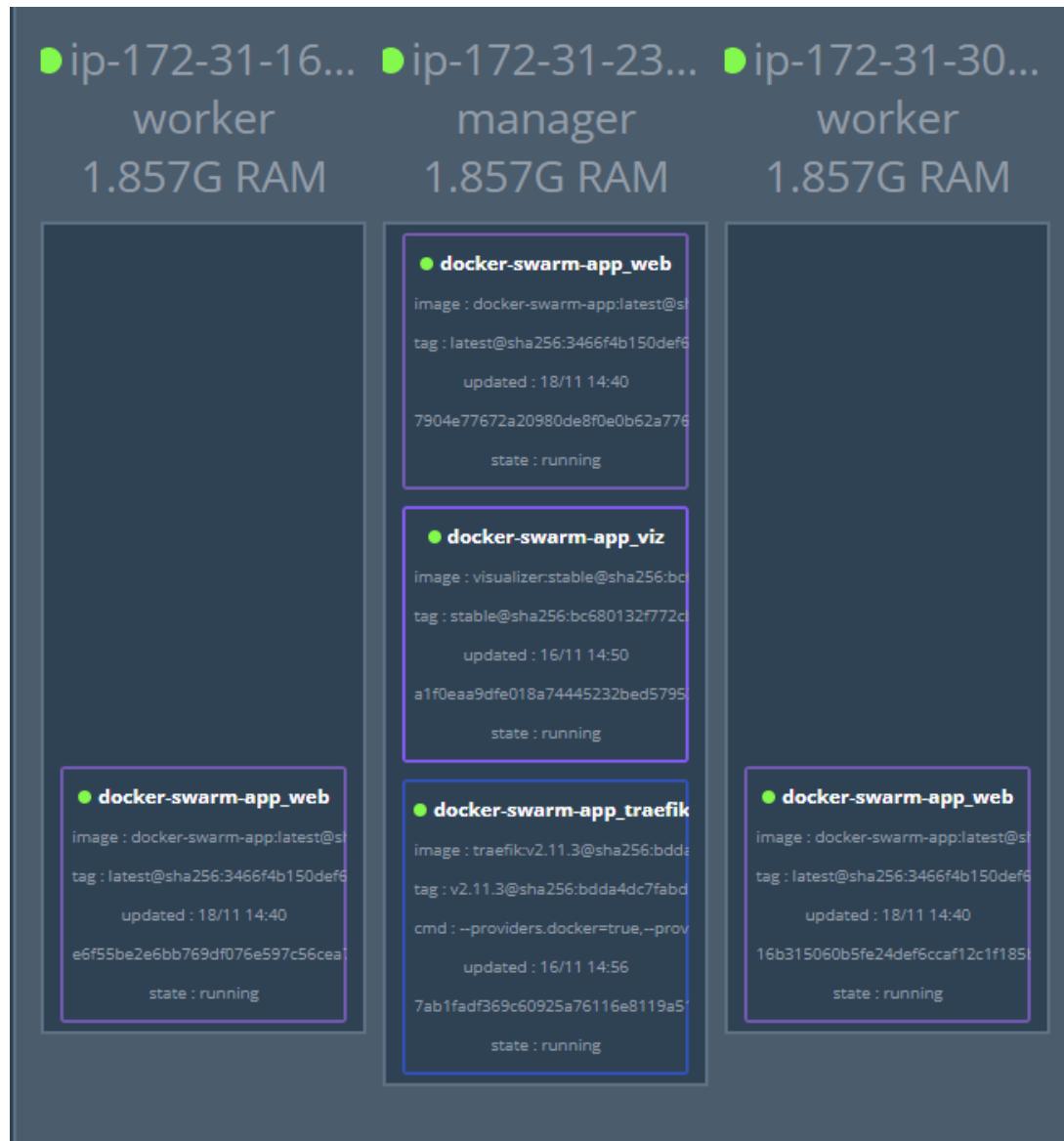
Steg 3: Kontrollera att tjänsten körs:

```
docker service ps docker-swarm-app_viz
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
rweui9nai1gi	docker-swarm-app_viz.1	dockersamples/visualizer:stable	ip-172-31-23-10.eu-west-1.compute.internal	Running	Running	2 days ago	

Steg 4: Öppna Visualizer

- Surfa in till managers publika IP följt av port 8081, alltså i mitt fall: <http://34.246.185.128:8081>
- Du ser nu alla noder och containrar i ditt Swarm-kluster visuellt.



Sammanfattningsvis:

- Docker Visualizer** körs som en separat service på manager, exponerar ett webbläsargränssnitt och visar i realtid alla noder och containrar i Swarm-klustret.

Traefik

Traefik är en dynamisk reverse proxy och lastbalanserare designad för **Docker Swarm**.

I min miljö körs **Traefik** på managern, där den automatiskt upptäcker alla tjänster och repliker som körs ute på klustrets noder. Detta gör att min applikation, oavsett om dess containrar körs på manager-noden eller på mina två workers, alltid nås via en central och smart styrd ingångspunkt.

Ett av huvudskälen att använda Traefik i min kluster är dess automatiserade hantering av HTTPS via Let's Encrypt. Med ACME-integration bygger Traefik själv ut, förnyar och lagrar certifikat utan att du behöver göra något manuellt — vilket ger en trygg och självgenererande säkerhetslösning på både port 80 och 443.

Utöver detta fungerar Traefik som en dynamisk reverse proxy, där routning uppdateras i realtid när tjänster skalias upp eller ned. All trafik lastbalanseras automatiskt över mina tre repliker av [web](#)-tjänsten och fördelar jämnt oavsett vilken nod de körs på.

Med Traefiks dashboard, som jag exponerar på port 8080, får jag dessutom en tydlig visuell överblick över routers, tjänster, certifikat och trafikflöden i realtid — perfekt för att verifiera att lastbalansering, HTTPS och routning fungerar som tänkt.

- Traefik är därför en komplett och självgående lösning för att hantera reverse proxy, trafikstyrning och automatiska HTTPS-certifikat i mitt Docker Swarm-kluster.

Steg 1: Börja med att återigen addera följande till docker-stack.yml som vi skapade tidigare för att lägga till Traefik som tjänst till vår stack:

- Eftersom Traefik är en tjänst listar vi även den under **services** som nedan.

```
services:  
  traefik:  
    image: traefik:v2.11.3  
    command:  
      - "--providers.docker=true"  
      - "--providers.docker.swarmMode=true"  
      - "--providers.docker.exposedbydefault=false"  
      - "--entrypoints.web.address=:80"  
      - "--entrypoints.websecure.address=:443"  
      - "--entrypoints.web.http.redirects.entrypoint.to=websecure"  
      - "--entrypoints.web.http.redirects.entrypoint.scheme=https"  
      - "--certificatesresolvers.myresolver.acme.httpchallenge=true"  
      - "--certificatesresolvers.myresolver.acme.httpchallenge.entrypoint=web"  
      - "--certificatesresolvers.myresolver.acme.email=91maxore@afe.molndal.se"  
      - "--certificatesresolvers.myresolver.acme.storage=/letsencrypt/acme.json"  
      - "--api.insecure=true"  
      - "--log.level=DEBUG"  
    ports:  
      - "80:80"  
      - "443:443"  
      - "8080:8080"  
    volumes:  
      - /var/run/docker.sock:/var/run/docker.sock:ro  
      - traefik_letsencrypt:/letsencrypt  
    deploy:  
      placement:  
        constraints:  
          - node.role == manager  
      restart_policy:  
        condition: on-failure  
    networks:  
      - webnet  
  
networks:  
  webnet:  
    driver: overlay  
  
volumes:  
  traefik_letsencrypt:
```

Beskrivning (Traefik)

- Kör Traefik som en Swarm-tjänst placerad på manager-noden.
- Använder Docker-socket för att automatiskt upptäcka tjänster och repliker i Swarm-klustret.
- Fungerar som en reverse proxy och lastbalanserare för alla tjänster som har Traefik-labels.
- Hanterar HTTPS automatiskt med Let's Encrypt via ACME.
- Exponerar HTTP (80), HTTPS (443) och Traefik Dashboard (8080) på manager-noden.
- Dirigerar all trafik från HTTP → HTTPS med automatisk omdirigering.
- Lagrar certifikat i en persistent volym för att undvika att certifikat återskapas vid omstart.
- Körs i overlay-nätverk ([webnet](#)) för att kunna nå alla tjänster i Swarm-klustret.

Steg 2: Deploya återigen stacken genom att köra detta på manager-noden:

```
docker stack deploy -c docker-stack.yml docker-swarm-app
```

Nu när vi konfigurerat alla tre tjänster inom stacken så kommer stacken starta:

- Traefik på managern
- Min app med 3 repliker fördelade över noderna inom Docker Swarm-klustret
- Docker Visualizer på managern

Steg 3: Kontrollera att tjänsten körs

```
docker service ps docker-swarm-app_traefik
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
gv35hk62z5rf	docker-swarm-app_traefik.1	traefik:v2.11.3	ip-172-31-23-10.eu-west-1.compute.internal	Running	Running 2 days ago
nxoiv7zbeozi	_ docker-swarm-app_traefik.1	traefik:v2.11.3	ip-172-31-23-10.eu-west-1.compute.internal	Shutdown	Shutdown 2 days ago

Steg 4: Kontrollera att samtliga tjänster körs

```
docker service ls
```

Detta borde visa att samtliga tjänster inom stacken vi konfigurerat körs och är replikerade.

- **Traefik:** Reverse proxy med HTTPS via Let's Encrypt, dashboard på port 8080.
- **Web:** Applikation med flera repliker, lastbalanseras av Traefik.
- **Docker Visualizer:** Visar klustrets noder och containrar i realtid på port 8081.

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
qpwqobw27qe2	docker-swarm-app_traefik	replicated	1/1	traefik:v2.11.3	*:80->80/tcp, *:443->443/tcp, *:8080->8080/tcp
4ko4udt8wrfm	docker-swarm-app_viz	replicated	1/1	dockersamples/visualizer:stable	*:8081->8080/tcp
v9h3cbe9drco	docker-swarm-app_web	replicated	3/3	91maxore/docker-swarm-app:latest	

Steg 5: Öppna Traefiks Dashboard

- Surfa in till managers publika IP följt av port 8080, alltså i mitt fall: <http://34.246.185.128:8080>
- Du ser nu alla routers, tjänster och trafikflöden i ditt Swarm-kluster visuellt via Traefiks dashboard.
- Notera att jag konfigurerat **wavvy.se** domänen via Loopia så den konfigurationen är inte inkluderad i denna guide.

The screenshot shows the Traefik 2.11.3 dashboard with a dark header bar. The header includes the Traefik logo, version 2.11.3, and navigation links for Dashboard, HTTP, TCP, UDP, and Plugins. Below the header, there's a section titled "→] Entrypoints" with three cards:

- TRAEFIK**: Port **:8080**
- WEB**: Port **:80**
- WEBSECURE**: Port **:443**

Steg 6: För att HTTPS ska fungera korrekt behöver vi konfigurera Traefik-labels på web-tjänsten så att den kan routa trafiken till min domän. Detta gör vi genom att ersätta nuvarande konfiguration för web i docker-stack.yml med följande:

```
web:
  image: 91maxore/docker-swarm-app:latest
  deploy:
    replicas: 3
    restart_policy:
      condition: on-failure
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.web.rule=Host(`wavvy.se`)"
      - "traefik.http.routers.web.entrypoints=websecure"
      - "traefik.http.routers.web.tls=true"
      - "traefik.http.routers.web.tls.certresolver=myresolver"
      - "traefik.http.services.web.loadbalancer.server.port=80"
  networks:
    - webnet
```

- Ersätt därmed denna med den tidigare web-konfiguration i stack-filen vi använde oss av.
- Dessa labels gör att Traefik vet vilken domän trafiken ska routas till, vilka entrypoints som ska användas, att TLS ska aktiveras, och vilken certifikatlösare som ska hantera Let's Encrypt-certifikaten.
- Traefik-labels konfigurerar web-tjänsten så att HTTPS fungerar och all HTTP-trafik automatiskt dirigeras till HTTPS.

Steg 7: Verifiera HTTPS

- Surfa nu in till **https://wavvy.se**
- Vi kan därmed granska att appen fungerar som den ska med HTTPS/SSL. Du kan även se på bilden att anslutningen är säker och att certifikatet är giltigt.



Traefik

- Tar emot trafiken
- Skapar certifikat automatiskt via Let's Encrypt
- Lastbalanserar över mina 3 web-repliker
- Dirigerar all HTTP → HTTPS

Sammanfattningsvis:

- Traefik körs som en separat service på manager, exponerar ett webbläsargränssnitt och visar i realtid alla routers, tjänster och trafikflöden i Swarm-klustret.

Beskrivning av de tre tjänsterna i min stack:

- **docker-swarm-app_web (Applikationen)** Webbapplikationen hanterar allt innehåll, som HTML och PHP, och körs som flera repliker som fördelas mellan manager och worker-noder i Swarm-klustret. Det gör att applikationen kan skalas och distribueras över flera noder, vilket ger hög tillgänglighet och jämn belastning utan att påverka användarupplevelsen.
- **docker-swarm_viz (Docker Visualizer)** Visualizer är ett grafiskt verktyg som visar Swarm-klustret i realtid, inklusive manager- och worker-noder samt alla containrar. Det gör det enkelt att övervaka hur tjänster och repliker distribueras över klustret, vilket ger snabb insikt i klustrets status och hjälper till att upptäcka problem med belastning eller distribution.
- **docker-swarm_traefik (Traefik)** Traefik är en reverse proxy och lastbalanserare som körs i Swarm på manager-noden och som hanterar inkommande trafik. Den hanterar automatiskt routing av trafik till mina tjänster, distribuerar trafiken till min web-applikation, skapar och förnyar HTTPS-certifikat via Let's Encrypt, och ger en visuell översikt över routers, tjänster och trafikflöden via dashboarden.

Automatiserad deployment med GitHub Actions (CI/CD)

Steg 1: Skapa ett GitHub-repo

- Bege dig över till ditt GitHub-konto
- Skapa ett nytt repo på GitHub genom att klicka på "**New repository**"
- Jag döpte min till **docker-swarm-app2** enbart för att demonstrera
- Välj **Public** eller **Private** beroende på behov.
- Klicka på "**Create repository**"

Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).
Required fields are marked with an asterisk (*).

1 General

Owner * Repository name *

91maxore-hub docker-swarm-app2

docker-swarm-app2 is available.

Great repository names are short and memorable. How about [psychic-rotary-phone](#)?

Description

0 / 350 characters

2 Configuration

Choose visibility * Public

Choose who can see and commit to this repository

Add README Off

READMEds can be used as longer descriptions. [About READMEs](#)

Add .gitignore No .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)

Add license No license

Licenses explain how others can use your code. [About licenses](#)

Connect GitHub Apps 1 app selected

Connect this repository to apps 91maxore-hub is subscribed to

Create repository

Efter att du skapat ditt repo kommer du bli hänvisad till följande instruktioner som du kan se nedan på bilden.
Kopiera **Quick setup**-länken och följ vidare instruktionerna på mitt nästa steg.

The screenshot shows a GitHub repository page for 'docker-swarm-app2'. At the top, there are buttons for Pin, Watch (0), Fork (0), and Star (0). Below the header, there are sections for 'Set up GitHub Copilot' and 'Add collaborators to this repository'. The main section is titled 'Quick setup — if you've done this kind of thing before'. It provides instructions for cloning the repository using 'Set up in Desktop' (with options for HTTPS or SSH) or by copying the URL (<https://github.com/91maxore-hub/docker-swarm-app2.git>). It also suggests creating a new file or uploading an existing one, and recommends including a README, LICENSE, and .gitignore. Below this, there are sections for '...or create a new repository on the command line' and '...or push an existing repository from the command line', each containing a code snippet for terminal commands.

Steg 2: Bege dig till projektmappen

- Öppna terminalen och bege dig till projektmappen där appens filer ligger på din lokala dator ex.

```
cd ~/docker-swarm-app
```

Steg 3: Initiera ett nytt Git-repo

```
git init
```

Steg 4: Lägg till README.md

```
git add README.md
```

Steg 5: Commit:a ändringarna:

```
git commit -m "CI/CD Pipeline"
```

Steg 6: Anslut lokalt repo till GitHub:

```
git remote add origin git@github.com:91maxore-hub/docker-swarm-app.git
```

- Ersätt med quick-setup länken vi kopierade tidigare.

Steg 7: Pusha till GitHub

```
git push -u origin main
```

Steg 8: Sen varje gång du gör ändringar i en eller flera filer kan du enkelt ange följande kommando för att pusha samtliga ändringar i filer till GitHub:

```
git add . && git commit -m "CI/CD Pipeline" && git push origin main
```

- Detta kommer endast pusha ändrade filer till GitHub och därifrån utgöra en CI/CD-automatiserings deployment så att Docker-imagen alltid håller sig uppdaterad, och därav samma med container-hosten som hostar appen.

Jag har nu initierat GitHub-repot och det är redo att användas för CI/CD-deployments.

Steg 9. Skapa GitHub Actions workflow

- Nästa steg är att skapa en **deploy.yml** för upprätthålla en CI/CD.
- Så skapa mappen och workflow-filen enligt strukturen som nedan:

```
mkdir -p .github/workflows
```

Workflow-filen (.github/workflows/deploy.yml) gör följande:

- Checkoutar koden från GitHub-repot.
- Sätter upp Docker Buildx för multi-platform builds.
- Loggar in på Docker Hub med GitHub Secrets.
- Bygger Docker-imagen för applikationen.
- Pushar imagen till Docker Hub.
- Ansluter till Swarm-manager via SSH med GitHub Secrets.
- Deployar stacken på Docker Swarm med **docker stack deploy -c docker-stack.yml**, uppdaterar tjänster och rullar ut den nya imagen automatiskt.

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Log in to DockerHub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and push Docker image
        run: |
          docker build -t 91maxore/docker-swarm-app:latest .
          docker push 91maxore/docker-swarm-app:latest

      - name: Deploy to Swarm
        uses: appleboy/ssh-action@v0.1.7
        with:
          host: ${{ secrets.SSH_HOST }}
          username: ${{ secrets.SSH_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            docker stack deploy -c /home/ec2-user/docker-stack.yml docker-swarm-app
```

- Innan vi kan gå vidare med att **deploya deploy.yml-filen** behöver vi sätta upp lite **GitHub Secrets**.

GitHub Secrets-konfigurationer

Repository secrets		New repository secret
Name	Last updated	
DOCKER_PASSWORD	3 days ago	 
DOCKER_USERNAME	3 days ago	 
SSH_HOST	3 days ago	 
SSH_PRIVATE_KEY	3 days ago	 
SSH_USER	3 days ago	 

GitHub Secrets-tabell

Secret	Beskrivning
DOCKER_USERNAME	Mitt användarnamn på Docker Hub, används för att logga in och pusha images - 91maxore
DOCKER_PASSWORD	Mitt lösenord för Docker Hub
SSH_HOST	IP-adress till Swarm-manager där stacken deployas - 34.246.185.128
SSH_USER	Användarnamnet som används för SSH-anslutningen till manager-noden - ec2-user
SSH_PRIVATE_KEY	Privat SSH-nyckel som matchar en publik nyckel på Swarm-manager för autentisering

Skapandet av en GitHub Secret

- Öppna ditt repo på GitHub (ex. <https://github.com/91maxore-hub/docker-swarm-app>)
- Navigera till fliken **Settings**
- Navigera till **Secrets and variables** → **Actions**
- Klicka på "**New repository secret**"
- Fyll i:
 - **Name** – t.ex. `SSH_HOST`
 - **Secret** – `34.246.185.128`
- Spara med "**Add secret**"

Enligt bästa praxis ska inga känsliga värden, såsom IP-adresser, domännamn, SSH-nycklar eller e-postadresser etc. hårdkodas i koden. Istället lagras dessa uppgifter säkert som **GitHub Secrets** i repot för att skydda dem från obehörig åtkomst och för att underlätta säker hantering.

Steg 1: Lägg till workflow och pusha

För att kontrollera att workflow-filen och CI/CD-deployment fungerar korrekt, pusha ändringarna i ett steg:

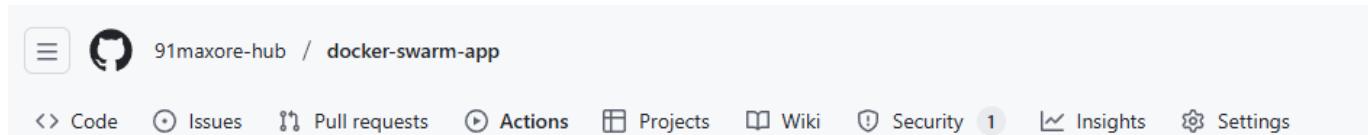
```
git add .github/workflows/deploy.yml && git commit -m "Lägg till GitHub Actions workflow för CI/CD" && git push origin main
```

Steg 2: Verifiering av CI/CD funktionalitet

Gå till ditt GitHub-repo, exempelvis:

<https://github.com/91maxore-hub/docker-swarm-app>

Navigera sedan till fliken **Actions**



Om CI/CD är korrekt konfigurerat bör du se att de senaste körningarna är markerade med en grön bok som nedan:

All workflows

Showing runs from all workflows

Filter workflow runs

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.

Give feedback X

19 workflow runs

		Event	Status	Branch	Actor	...
✓ CI/CD Pipeline	CI/CD Pipeline #19: Commit 8314f6e pushed by 91maxore-hub	main	Success	Nov 18, 11:45 PM GMT+1	44s	...
✓ CI/CD Pipeline	CI/CD Pipeline #18: Commit c0c9eec pushed by 91maxore-hub	main	Success	Nov 18, 11:45 PM GMT+1	57s	...
✓ CI/CD Pipeline	CI/CD Pipeline #17: Commit f368100 pushed by 91maxore-hub	main	Success	Nov 18, 7:11 PM GMT+1	40s	...
✓ CI/CD Pipeline	CI/CD Pipeline #16: Commit 3031ef7 pushed by 91maxore-hub	main	Success	Nov 18, 3:57 PM GMT+1	41s	...
✓ CI/CD Pipeline	CI/CD Pipeline #15: Commit df71732 pushed by 91maxore-hub	main	Success	Nov 18, 2:39 PM GMT+1	47s	...
✓ CI/CD Pipeline	CI/CD Pipeline #14: Commit 2036145 pushed by 91maxore-hub	main	Success	Nov 18, 2:39 PM GMT+1	44s	...
✓ CI/CD Pipeline	CI/CD Pipeline #13: Commit 555d152 pushed by 91maxore-hub	main	Success	Nov 18, 2:14 PM GMT+1	34s	...
✓ CI/CD Pipeline	CI/CD Pipeline #12: Commit 2134b79 pushed by 91maxore-hub	main	Success	Nov 18, 2:11 PM GMT+1	47s	...
✓ CI/CD Pipeline	CI/CD Pipeline #11: Commit 9237c05 pushed by 91maxore-hub	main	Success	Nov 18, 2:07 PM GMT+1	45s	...
✓ CI/CD Pipeline	CI/CD Pipeline #10: Commit ba34ec6 pushed by 91maxore-hub	main	Success	Nov 18, 2:05 PM GMT+1	42s	...

Dessutom en **status** som visar **Success**. Exempel på ett lyckat arbetsflöde:

build-and-deploy — Success

← CI/CD Pipeline

✓ CI/CD Pipeline #19

Summary

Triggered via push 11 hours ago

91maxore-hub pushed → [8314f6e](#) main

Status: Success Total duration: 44s Artifacts: —

Jobs

✓ build-and-deploy

Run details

Usage

Workflow file

deploy.yml

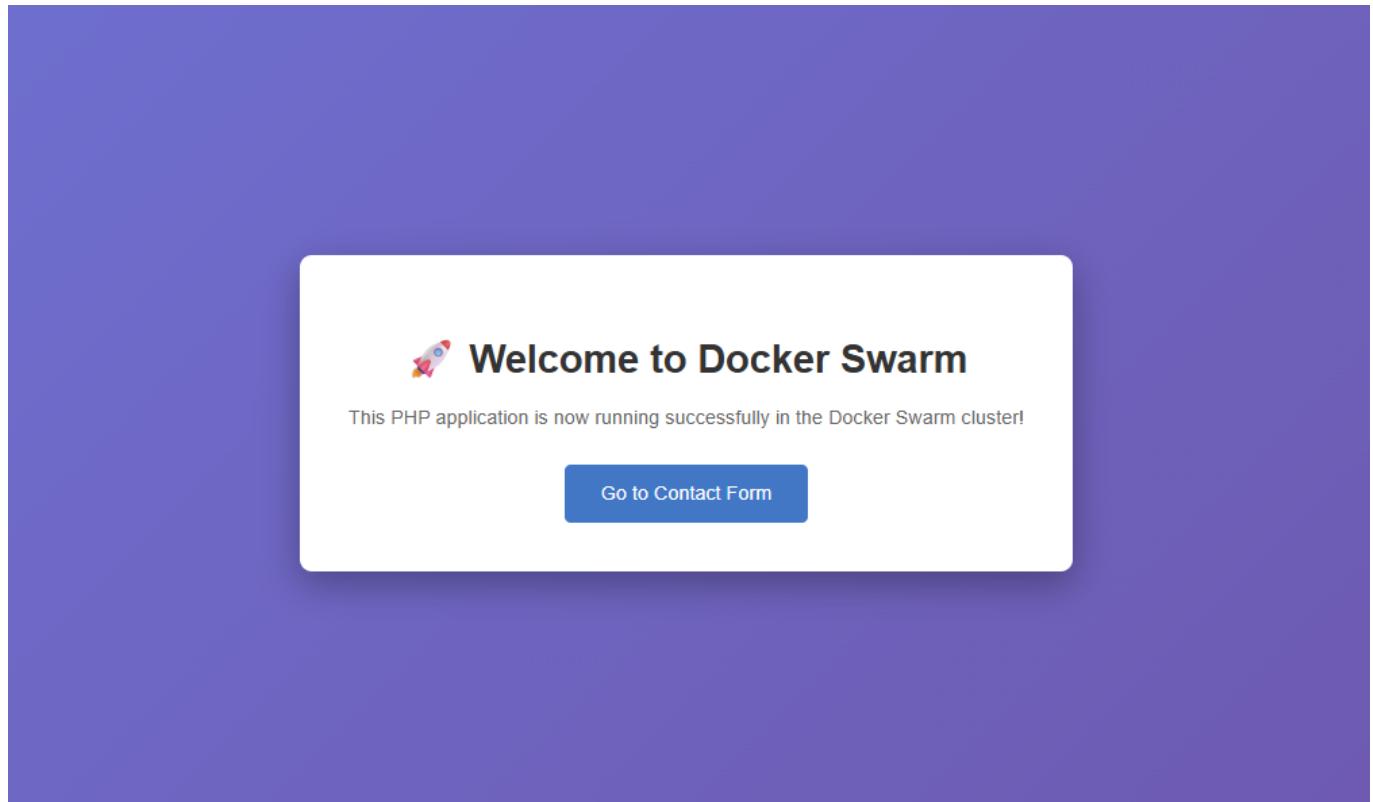
on: push

✓ build-and-deploy 39s

Resultat

Efter att allt var uppsatt och CI/CD-deployment gick igenom kunde jag gå till:  <https://wavvy.se>

Min PHP-app laddas med giltigt SSL-certifikat, automatisk HTTPS och reverse proxy som hanterar trafiken smidigt genom **Traefik**. Allt detta sker helt automatiskt – både deployment och certifikatförflyelse.



Användningen av Infrastructure as Code (IaC)

Jag använder Infrastructure as Code (IaC) genom att definiera applikationens infrastruktur med kod, främst med hjälp av Docker och Docker Swarm. Min miljö består av tre EC2-servrar, där en fungerar som manager och två som workers, vilket möjliggör en skalbar och robust containerbaserad miljö. Applikationen, tillsammans med **Traefik** som reverse proxy och **Docker Visualizer** för övervakning, distribueras helt via min `docker-stack.yml`-fil. Detta säkerställer att hela stacken kan köras konsekvent oavsett miljö, och gör det enkelt att lägga till eller ta bort servrar vid behov.

Jag använder också **GitHub Actions** för att automatisera hela deployment-processen. När jag pushar till main-branchen byggs en Docker-image automatiskt och deployas till mina servrar via SSH. Alla känsliga värden, såsom domännamn och SSH-nycklar, hanteras säkert via GitHub Secrets istället för att hårdkodas i koden.

Genom att definiera och hantera infrastrukturen som kod – allt från containrar och webbserver till certifikat och nätverksinställningar – blir uppdateringar och skalning av miljön mycket smidigare.

Användningen av säkerhet

Säkerheten i min lösning hanteras på flera nivåer. Känsliga värden som SSH-nycklar, domännamn och certifikat lagras aldrig i koden, utan hanteras säkert via GitHub Secrets vid deployment.

Traefik används som dynamisk reverse proxy och lastbalanserare i mitt Docker Swarm-kluster. Den körs på manager-noden och upptäcker automatiskt alla tjänster och repliker som körs på både manager och worker-noder, vilket gör att applikationen alltid nås via en central och smart styrd ingångspunkt. Traefik hanterar också HTTPS via Let's Encrypt med ACME-integration, vilket innebär att certifikat byggs, förnyas och lagras automatiskt utan manuell hantering. Trafiken omdirigeras dessutom automatiskt från HTTP till HTTPS, vilket säkerställer krypterad och trygg åtkomst för användare.

Utöver detta fungerar Traefik som en dynamisk reverse proxy där routing uppdateras i realtid när tjänster skalas upp eller ned. All trafik lastbalanseras automatiskt över mina tre repliker av `web`-tjänsten, vilket ger jämn fördelning oavsett vilken nod containrarna körs på. Genom Traefiks dashboard på port 8080 kan jag dessutom övervaka routers, tjänster, certifikat och trafikflöden i realtid, vilket ger en tydlig visuell överblick över säkerhet och trafikstyrning.

Varje tjänst körs dessutom i en egen container, vilket isolerar komponenterna och begränsar påverkan vid eventuella säkerhetsincidenter. Regelbundna uppdateringar och patchning av containrar bidrar ytterligare till en robust och säker lösning.

Serverless App

I detta projekt har jag byggt en skalbar och kostnadseffektiv serverless-miljö för en webbapplikation på AWS. Applikationen använder **AWS S3** för hosting av statiska filer, **AWS Lambda** för backend-logik, **API Gateway** för att hantera HTTP-förfrågningar och **DynamoDB** för lagring av formulärsvärden. För att säkerställa snabb och säker åtkomst till applikationen används **CloudFront** som reverse proxy med stöd för HTTPS.

Applikationen är helt serverlös, vilket innebär att infrastrukturen automatiskt skalar baserat på belastning, utan behov av att hantera servrar eller operativsystem. Detta gör lösningen både flexibel och lättunderhållbar, samtidigt som den erbjuder hög tillgänglighet och prestanda.

För att underlätta utveckling och deployment har jag implementerat **CI/CD med AWS CodePipeline kopplat till GitHub**, vilket automatiskt bygger och distribuerar nya versioner av applikationen till S3 och Lambda. Detta säkerställer snabb iteration och pålitlig uppdatering av applikationen utan manuella steg.

Denna lösning visar hur serverless-teknologi kan kombineras med molntjänster för att skapa en modern webbmiljö som är skalbar, säker och kostnadseffektiv.

Noterbart är att i detta projekt har jag utnyttjat följande molntjänster från AWS:

- **S3:** Hosting av statiska filer som HTML, CSS och JavaScript.
- **Lambda:** Serverlös körning av backend-logik för formulärhantering.
- **API Gateway:** Hantering av HTTP-förfrågningar och routing till Lambda-funktioner.
- **DynamoDB:** Lagring av formulärsvärden.
- **CloudFront:** Reverse proxy med HTTPS för säker och snabb åtkomst.
- **CodePipeline + GitHub:** CI/CD som möjliggör automatiska bygg och deployment av applikationen.

Komponentöversikt och Syfte

Komponent	Beskrivning	Användningsområde	Kommentar
S3	Lagrar och serverar statiska filer som HTML och CSS	Hosting av frontend	Ger hög tillgänglighet och enkel skalning utan serverhantering
Lambda	Serverlösa funktioner som kör backend	Hantering av formulärdata	Skalar automatiskt baserat på belastning, inga servrar att hantera
API Gateway	Hanterar HTTP-förfrågningar och routear dem till Lambda	Exponering av backendfunktioner som API	Säkerställer HTTP API-kommunikation mellan frontend och Lambda
DynamoDB	Databas för lagring av formulärsvar	Databas för applikationen	Fullt hanterad, serverlös, mycket låg latency
CloudFront	Reverse Proxy med HTTPS	Snabb och säker åtkomst till applikationen	Minskar latens globalt och ger HTTPS-stöd
CodePipeline + GitHub	CI/CD-pipeline som bygger och deployar applikationen automatiskt	Automatiserad deployment	Säkerställer snabb iteration och pålitlig uppdatering

Mappstruktur

Katalog / Fil	Typ	Beskrivning
serverless-app/	Mapp	Rootmapp för webbapplikationen
index.html	HTML-fil	Huvudsidan för webbapplikationen
contact_form.html	HTML-fil	Sida med kontaktformulär för användare
thankyou.html	HTML-fil	Sida som visas efter att formuläret skickas
style.css	CSS-fil	Stilark som styr utseende och layout för webbapplikationen
contactFormHandler.js	JavaScript-fil	Backend-funktion (AWS Lambda) som hanterar formulärinlämning och sparar data i DynamoDB

Konfiguration av Amazon S3-bucket

Denna guide beskriver hur man skapar och konfigurerar en **Amazon S3-bucket** för att hosta statiska webbapplikationsfiler. Målet är att tillhandahålla en högpresterande och skalbar hostingmiljö för HTML-, CSS- och övriga statiska resurser. Bucketen kommer att konfigureras med offentlig läsbehörighet för hosting, samt integreras med CloudFront för snabb distribution och HTTPS-stöd.

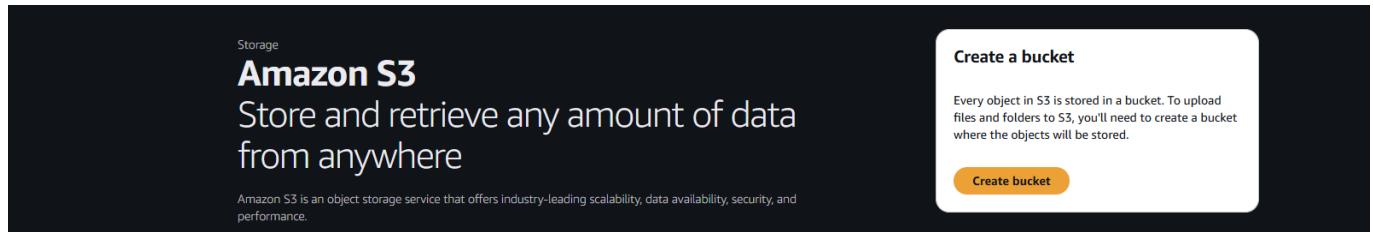
Steg 1: Bege dig till aws.amazon.com

The screenshot shows the AWS Console Home page. At the top, there's a navigation bar with the AWS logo, a search bar, and account information. Below the navigation bar is the 'Console Home' header. On the left, there's a sidebar with 'Recently visited' services: EC2, CloudFront, S3, Lambda, DynamoDB, API Gateway, CloudWatch, and IAM. To the right of the sidebar are several cards: 'Applications' (0), 'Welcome to AWS' (with sections for Getting started with AWS, Training and certification, and AWS Builder Center), and 'Cost and usage' (showing current month cost of \$0.34, forecasted month end cost as 'Data unavailable', and a bar chart of monthly costs from Jun 25 to Nov 25). There are also links to 'Go to myApplications', 'Go to AWS Health', and 'Go to Billing and Cost Management'.

Steg 2: Ange S3 i sökrutan och välj "S3 - Scalable Storage in the Cloud"

The screenshot shows the AWS Services page. The search bar at the top contains the text 'S3'. The main content area is titled 'Services' and lists three items: 'S3 Scalable Storage in the Cloud', 'S3 Glacier Archive Storage in the Cloud', and 'AWS Snow Family Large Scale Data Transport'. Each item has a star icon to its right. Below this section is another titled 'Features' with three items: 'S3 on Outposts' (with a sub-item 'AWS Outposts feature'), 'Exports to S3' (with a sub-item 'DynamoDB feature'), and 'S3 Access Grants' (with a sub-item 'S3 feature'). Each feature item has a star icon to its right. There are also 'Show more' buttons for both the services and features sections.

Steg 3: Välj Create Bucket



Steg 4: Ange ett namn för vår S3-bucket, jag kommer namnge den "serverless-bucket-2025"

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region
Europe (Ireland) eu-west-1

Bucket type Info

- General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.
- Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info
serverless-bucket-2025

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn more ↗](#)

Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.

Choose bucket
Format: s3://bucket/prefix

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

Object Ownership

- ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
- ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership
Bucket owner enforced

Steg 5: Gå sedan lite längre ner och bocka ur "Block Public Access settings for this bucket" eftersom vi vill ju komma åt våra filer genom appen

- Bucket Versioning kan aktiveras för att automatiskt behålla tidigare versioner av filer, vilket underlättar återställning vid oavsiktliga ändringar eller borttagningar. Men i detta fall skippar vi det.
- Resten kan lämnas som det är.

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more ↗](#)

Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through new public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- Block public and cross-account access to buckets and objects through any public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

⚠ Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more ↗](#)

Bucket Versioning

- Disable
- Enable

Steg 6: Välj slutligen "Create bucket"

Steg 7: Du bör nu få en översikt över din nyskapade S3-bucket

General purpose buckets (2) [Info](#)

Buckets are containers for data stored in S3.

Name	AWS Region	Creation date
serverless-bucket-2025	Europe (Ireland) eu-west-1	November 16, 2025, 18:07:22 (UTC+01:00)

[Create bucket](#)

Steg 8: Gå sedan in på vår nyskapade S3-bucket och klicka på "Upload" längst bort till höger

serverless-bucket-2025 [Info](#)

[Objects](#) [Metadata](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (4)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Name	Type	Size
Find objects by prefix		

[Actions](#) [Create folder](#) [Upload](#)

Steg 9: Välj därefter att ladda upp mappar eller filer. I vårt ändamål kommer vi att ladda upp endast appens filer. Så vi väljer "Add files" följt av "Upload" längst ner

Files and folders (0)

All files and folders in this table will be uploaded.

Name	Type	Size
Find by name		

[Remove](#) [Add files](#) [Add folder](#)

Destination [Info](#)

Destination [s3://serverless-bucket-2025](#)

Destination details
Bucket settings that impact new objects stored in the specified destination.

Permissions
Grant public access and access to other AWS accounts.

Properties
Specify storage class, encryption settings, tags, and more.

[Cancel](#) [Upload](#)

Steg 10: Slutligen bör du se en översikt över filerna vi precis laddade upp.

serverless-bucket-2025 [Info](#)

[Objects](#) [Metadata](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (4)

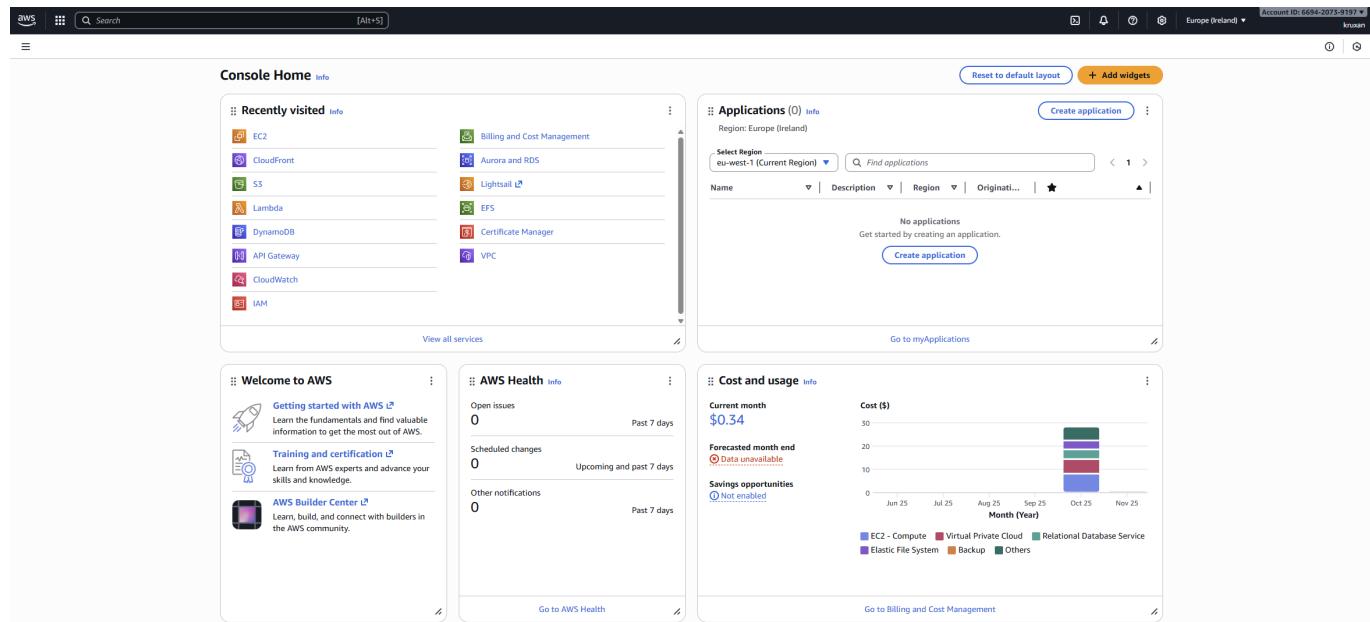
Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Name	Type	Last modified	Size	Storage class
contact_form.html	html	November 18, 2025, 13:56:57 (UTC+01:00)	2.0 KB	Standard
index.html	html	November 18, 2025, 13:56:57 (UTC+01:00)	821.0 B	Standard
style.css	css	November 18, 2025, 13:56:57 (UTC+01:00)	2.0 KB	Standard
thankyou.html	html	November 18, 2025, 13:56:57 (UTC+01:00)	1.2 KB	Standard

Konfiguration av Amazon DynamoDB för lagring av formulärsvar

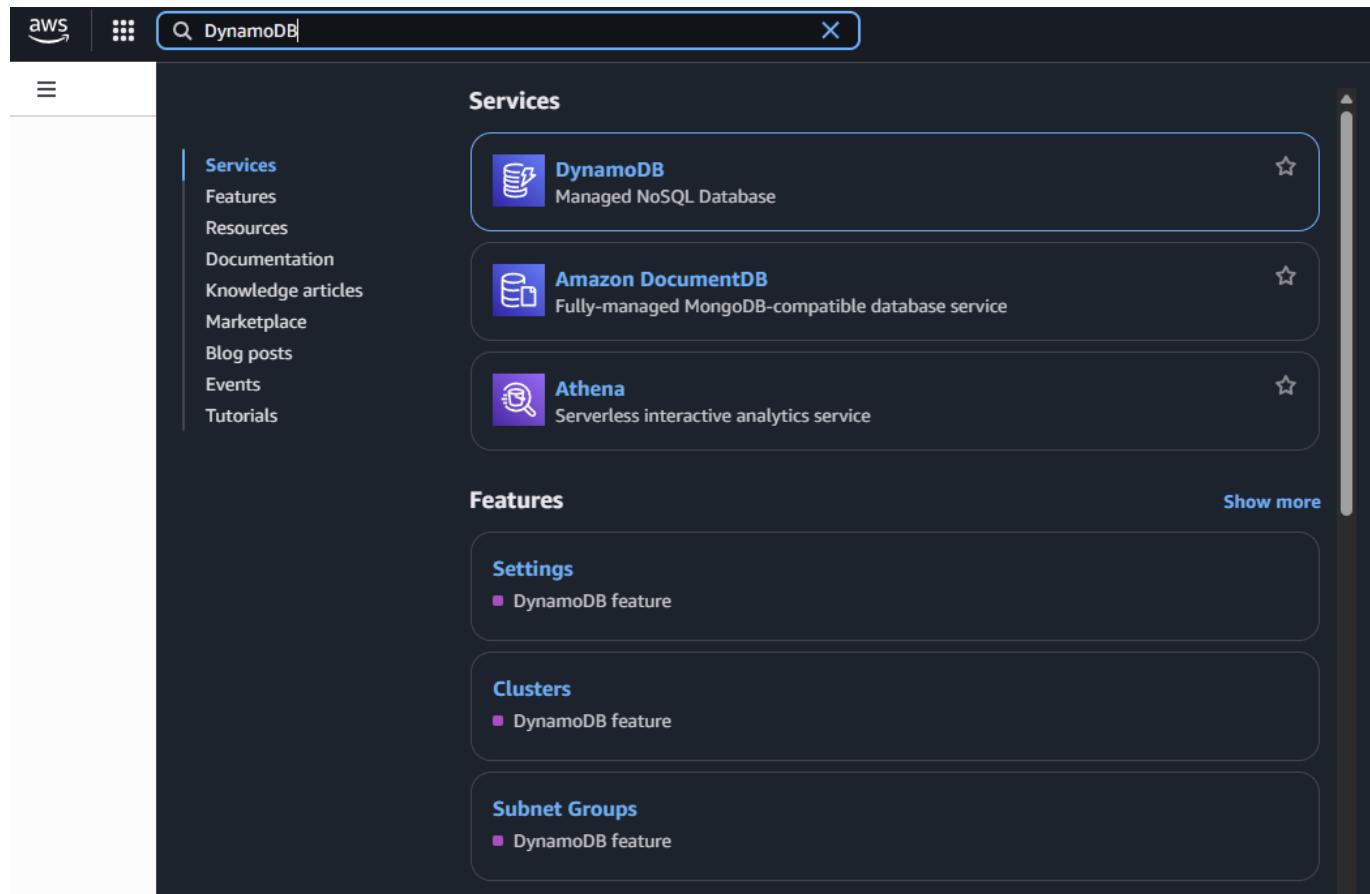
Denna guide beskriver hur man skapar och konfigurerar en **Amazon DynamoDB**-tabell för att lagra data från webbapplikationens kontaktformulär. Målet är att tillhandahålla en högpresterande, serverlös och skalbar databaslösning som kan hantera varierande trafik utan att behöva hantera servrar.

Steg 1: Bege dig till aws.amazon.com



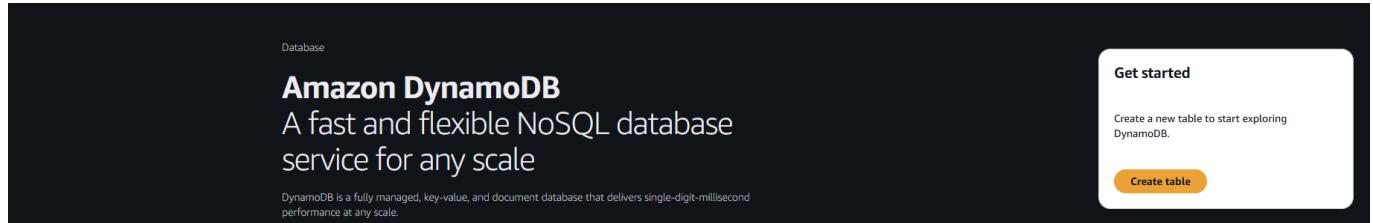
The screenshot shows the AWS Console Home page. On the left, there's a sidebar with 'Recently visited' services like EC2, CloudFront, S3, Lambda, DynamoDB, API Gateway, CloudWatch, and IAM. The main area has several cards: 'Welcome to AWS' (Getting started with AWS, Training and certification, AWS Builder Center), 'AWS Health' (Open issues, Scheduled changes, Other notifications), and 'Cost and usage' (Current month cost \$0.34, Forecasted month end, Savings opportunities). A search bar at the top right says 'Search' and has a dropdown for 'Region: Europe (Ireland)'. A top navigation bar includes links for 'Console Home', 'Info', 'Reset to default layout', '+ Add widgets', 'Create application', 'Go to myApplications', and account details.

Steg 2: Ange DynamoDB i sökrutan och välj "DynamoDB - Managed NoSQL Database"



The screenshot shows the AWS Services page with a search bar at the top containing 'DynamoDB'. The results are displayed in sections: 'Services' (DynamoDB Managed NoSQL Database, Amazon DocumentDB Fully-managed MongoDB-compatible database service, Athena Serverless interactive analytics service), 'Features' (Settings, Clusters, Subnet Groups, all listed as 'DynamoDB feature'), and a 'Show more' link. A sidebar on the left lists 'Services', 'Features', 'Resources', 'Documentation', 'Knowledge articles', 'Marketplace', 'Blog posts', 'Events', and 'Tutorials'.

Steg 3: Välj "Create table"



Steg 4: Ange ett namn för vår DynamoDB-databas, jag kommer namnge den ContactFormMessages

- Ange även **id** som Partition key och värdet ska vara **String**

Create table

Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
 1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 1 to 255 characters and case sensitive.

Table settings
 Default settings
The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.
 Customize settings
Use these advanced features to make DynamoDB work better for your needs.

Steg 5: Slutligen bör du se en översikt över databasen vi precis skapade

Tables (1) <small>Info</small>		Actions											
		Name	Status	Partition key	Sort key	Indexes	Replication Regions	Deletion protection	Favorite	Read capacity mode	Write capacity mode	Total size	Table class
<input type="checkbox"/>	ContactFormMessages	Active	id (\$)	-	0	0	Off	☆	On-demand	On-demand	5.6 kilobytes	Standard	

Konfiguration av AWS Lambda

Denna guide beskriver hur man skapar och konfigurerar **AWS Lambda**-funktioner för att hantera backend-logik i webbapplikationen. Målet är att tillhandahålla en skalbar, serverlös miljö där funktioner automatiskt kan exekveras som svar på HTTP-förfrågningar via API Gateway. Lambda-funktionerna kommer att hantera inlämning av formulärd data, validering av inkommande data och lagring i DynamoDB, utan att kräva några underhålls krav på servrar.

Steg 1: Bege dig till aws.amazon.com

The screenshot shows the AWS Console Home page. On the left, there's a sidebar with links to services like EC2, CloudFront, S3, Lambda, DynamoDB, API Gateway, CloudWatch, and IAM. The main area has several cards: "Recently visited" (EC2, CloudFront, S3, Lambda, DynamoDB, API Gateway, CloudWatch, IAM), "Welcome to AWS" (Getting started with AWS, Training and certification, AWS Builder Center), "AWS Health" (Open issues 0, Scheduled changes 0, Other notifications 0), and "Cost and usage" (Current month \$0.34, Forecasted month end Data unavailable, Savings opportunities Not enabled). A top navigation bar includes a search bar, account ID, and region (Europe (Ireland)).

Steg 2: Ange Lambda i sökrutan och välj "Lambda - Run code without thinking about servers"

The screenshot shows the AWS Services search results for "Lambda". The search bar at the top contains "Lambda". Below it, the "Services" section lists Lambda (Run code without thinking about servers), CodeBuild (Build and Test Code), and AWS Signer (Ensuring trust and integrity of your code). The "Features" section lists Lambda Insights (CloudWatch feature), Object Lambda Access Points (S3 feature), and Local processing (IoT Core feature). A "Show more" button is visible at the top right of the features section.

Steg 3: Navigera till "Functions"



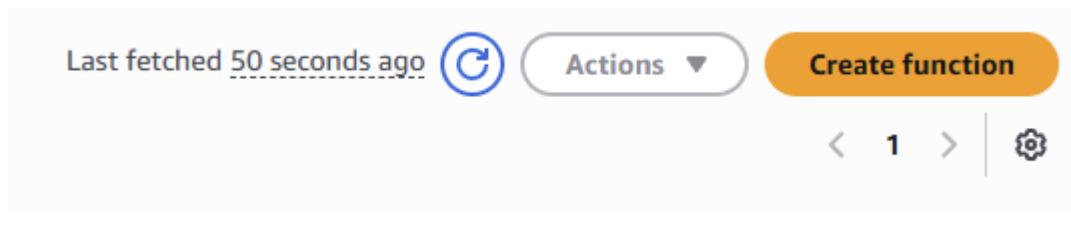
Lambda

Dashboard

Applications

Functions

Steg 4: Välj "Create function" längst till höger



Steg 5: Ange ett namn för vår Lambda-funktion, jag kommer namnge den contactFormHandler

- Välj "Author from scratch"
- Resten kan lämnas som det är

Create function Info

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Container image
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture Info
Choose the instruction set architecture you want for your function code.
 arm64
 x86_64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).
 Create a new role with basic Lambda permissions
 Use an existing role
 Create a new role from AWS policy templates

Role creation might take a few minutes. Do not delete the role or edit the trust or permissions policies in this role.

Lambda will create an execution role named contactFormHandler-role-vripu5zb, with permission to upload logs to Amazon CloudWatch Logs.

Steg 6: Slutligen bör du se en översikt över funktionen (contactFormHandler.js) vi precis skapade

Functions (1)					
<input type="text"/> Search by attributes or search by keyword					
Function name	Description	Package type	Runtime	Last modified	
contactFormHandler	-	Zip	Node.js 22.x	23 hours ago	

Steg 7: Gå in på vår nyskapade Lambda-funktion (contactFormHandler.js) och klistra in följande kod:

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
const crypto = require("crypto");

const db = new DynamoDBClient({ region: "eu-west-1" });

exports.handler = async (event) => {
    console.log("Incoming event:", JSON.stringify(event));

    if (event.httpMethod === "OPTIONS") {
        return {
            statusCode: 204,
            headers: {
                "Access-Control-Allow-Origin": "https://d3vbjy5bvefx3w.cloudfront.net",
                "Access-Control-Allow-Methods": "POST,OPTIONS",
                "Access-Control-Allow-Headers": "Content-Type",
            },
            body: ""
        };
    }

    let data;
    try {
        data = JSON.parse(event.body || "{}");
    } catch (e) {
        console.error("JSON parse error:", e);
        return {
            statusCode: 400,
            headers: { "Access-Control-Allow-Origin":
"https://d3vbjy5bvefx3w.cloudfront.net" },
            body: JSON.stringify({ error: "Invalid JSON in request" })
        };
    }

    const id = crypto.randomUUID();

    const params = {
        TableName: "ContactFormMessages",
        Item: {
            id: { S: id },
            name: { S: data.name || "UNKNOWN" },
            email: { S: data.email || "UNKNOWN" },
            message: { S: data.message || "EMPTY" },
            createdAt: { S: new Date().toISOString() }
        }
    };

    try {
        console.log("Trying to write to Dynamo:", params);
        await db.send(new PutItemCommand(params));
        console.log("Write SUCCESS");
    }
```

```
return {
  statusCode: 200,
  headers: {
    "Access-Control-Allow-Origin": "https://d3vbjy5bvefx3w.cloudfront.net",
    "Access-Control-Allow-Headers": "Content-Type",
    "Access-Control-Allow-Methods": "POST,OPTIONS",
  },
  body: JSON.stringify({
    success: true,
    id,
    ...data
  })
};

} catch (err) {
  console.error("DynamoDB ERROR:", err);

  return {
    statusCode: 500,
    headers: {
      "Access-Control-Allow-Origin": "https://d3vbjy5bvefx3w.cloudfront.net"
    },
    body: JSON.stringify({ error: "Could not save message", details: err.message
  })
}
};
```

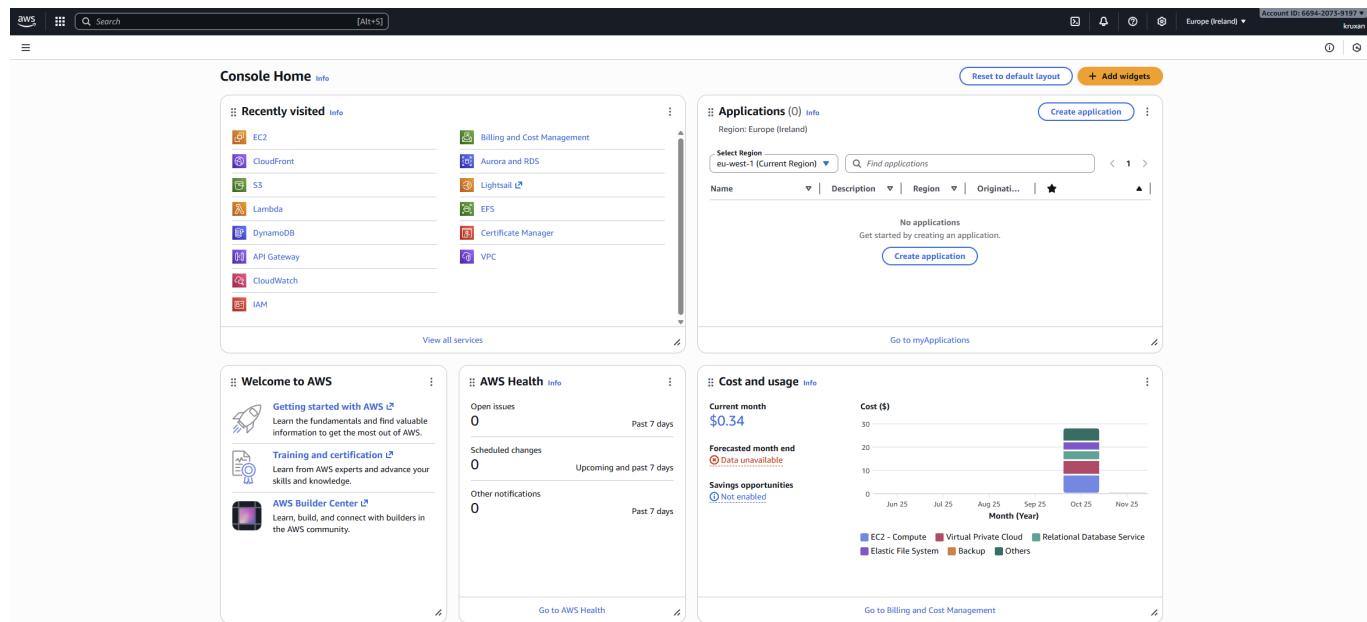
Beskrivning (Lambda – contactFormHandler.js)

- Körs som en serverlös AWS Lambda-funktion som hanterar inkommande HTTP-förfrågningar via API Gateway.
- Tar emot formulärdata från frontend och validerar JSON-innehållet.
- Genererar ett unikt ID för varje formulärinlämning med `crypto.randomUUID()`.
- Sparar formulärdata (`name`, `email`, `message`, `createdAt`) i DynamoDB-tabellen `ContactFormMessages`.
- Hanterar CORS (Cross-Origin Resource Sharing) för att möjliggöra anrop från frontend-distributionen på CloudFront.
- Notera att jag har min CloudFront-distribution för Access-Control-Allow-Origin, alltså den enda domänen som har tillstånd att använda min Lambda-fuction. Jag kommer gå igenom hur man sätter upp **API Gateway** och **CloudFront** i nästkommande steg.
- Du kan för tillfället ange din S3-Bucket för att testa dess funktionalitet men det fungerar på samma sätt eftersom du talar endast om för Lambda-funktionen vilken domän som är tillåten att använda den.

Uppsättning av Amazon API Gateway för HTTP API

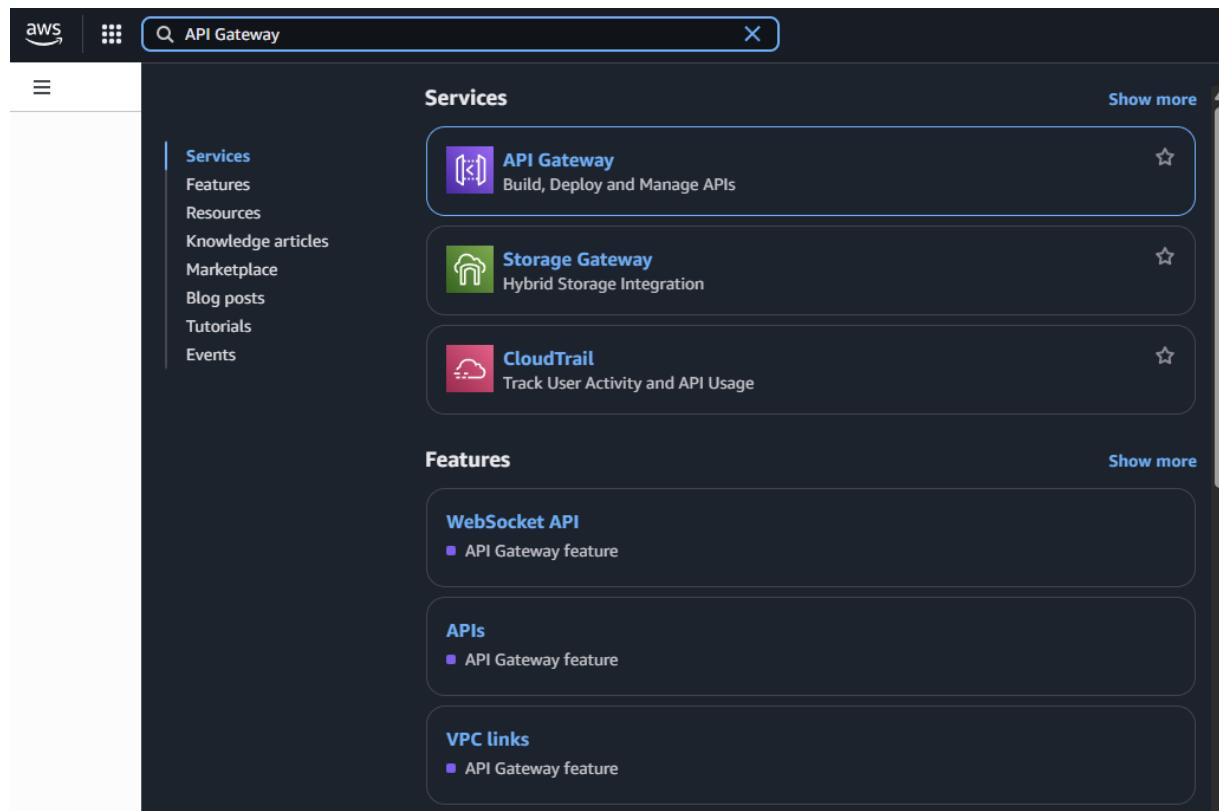
Denna guide beskriver hur man skapar och konfigurerar **Amazon API Gateway** som en HTTP API för att hantera kommunikationen mellan frontend och serverlösa Lambda-funktioner. Målet är att tillhandahålla en skalbar, säker och lättanvänd ingångspunkt för webbapplikationen, som möjliggör REST-likt interaktioner utan att behöva hantera servrar. **API Gateway** kommer att routa inkommande HTTP-förfrågningar till Lambda-funktionerna, hantera CORS och säkerställa att data från formulär kan skickas och tas emot på ett pålitligt sätt.

Steg 1: Bege dig till aws.amazon.com



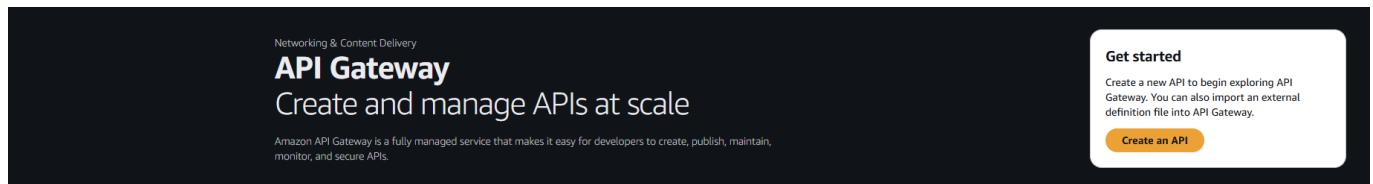
The screenshot shows the AWS Console Home page. On the left, there's a sidebar with 'Recently visited' services like EC2, CloudFront, S3, Lambda, DynamoDB, API Gateway, CloudWatch, and IAM. The main area has several cards: 'Welcome to AWS' (Getting started with AWS, Training and certification, AWS Builder Center), 'AWS Health' (Open issues, Scheduled changes, Other notifications), and 'Cost and usage' (Current month cost \$0.34, Forecasted month end, Savings opportunities). The 'Applications' card on the right shows a table with one row: 'No applications'. It includes a 'Create application' button and a link to 'Go to myApplications'.

Steg 2: Ange API Gateway i sökrutan och välj "API Gateway - Build, Deploy and Manage APIs"



The screenshot shows the AWS Services search results for 'API Gateway'. The search bar at the top contains 'API Gateway'. The results are categorized into 'Services' and 'Features'. Under 'Services', there are three items: 'API Gateway' (Build, Deploy and Manage APIs), 'Storage Gateway' (Hybrid Storage Integration), and 'CloudTrail' (Track User Activity and API Usage). Under 'Features', there are three items: 'WebSocket API' (API Gateway feature), 'APIs' (API Gateway feature), and 'VPC links' (API Gateway feature). Each item has a star icon to its right.

Steg 3: Välj "Create an API" längst till höger



Steg 4: Välj "HTTP API"

A screenshot of the "Choose an API type" page. It shows "HTTP API" selected. Description: "Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support." It also lists "Works with the following: Lambda, HTTP backends". At the bottom are "Import" and "Build" buttons.

Steg 5: Ange ett namn för vår API. Jag kommer namnge den contactHandlerFormAPI

- Välj även vår Lambda-funktion vi skapade under förgående steg under "**Integrations**"

A screenshot of the "Configure API" page. Under "API details", the "API name" is set to "contactHandlerFormAPI". Under "IP address type", "IPv4" is selected. In the "Integrations" section, there is one entry for "Lambda" with "eu-west-1" as the AWS Region, "arn:aws:lambda:eu-west-1:669420739197:function:contactFormHandler" as the Lambda function, and version "2.0". At the bottom, there are "Cancel", "Review and create", and "Next" buttons.

Steg 6: Här behöver vi ange en route till vår API som ska nås via vår Lambda-funktion

Fyll i:

- **Method** – POST
- **Resource path** – /contact
- **Integration target** - contactFormHandler
- Spara med "Add route"

Configure routes - optional

Configure routes [Info](#)

API Gateway uses routes to expose integrations to consumers of your API. Routes for HTTP APIs consist of two parts: an HTTP method and a resource path (e.g., GET /pets). You can define specific HTTP methods for your integration (GET, POST, PUT, PATCH, HEAD, OPTIONS, and DELETE) or use the ANY method to match all methods that you haven't defined on a given resource.

Method	Resource path	Integration target
POST	/contact	contactFormHandler

Add route Remove

Cancel Review and create Previous Next

Steg 7: Här kan du ange en stage för vår API.

Alltså ifall vi skapar en stage som heter "prod" så kan API:n nås via API-URL/prod/api

Men vi väljer att köra "\$default" för enkelhetens skull, detta resulterar med att vi kan nå vår API genom API-URL/api

Define stages - optional

Configure stages [Info](#)

Stages are independently configurable environments that your API can be deployed to. You must deploy to a stage for API configuration changes to take effect, unless that stage is configured to autodeploy. By default, all HTTP APIs created through the console have a default stage named \$default. All changes that you make to your API are autodeployed to that stage. You can add stages that represent environments such as development or production.

Stage name	Auto-deploy
\$default	<input checked="" type="checkbox"/> Remove

Add stage

Cancel Previous Next

Steg 8: Slutligen får vi en översikt över vår API med dess konfigurationer

- Välj därefter "Create"

Review and create

API name and integrations	Edit
API name contactHandlerFormAPI	IP address type IPv4
Integrations • contactFormHandler (Lambda)	
Routes	Edit
Routes	
• POST /contact → contactFormHandler (Lambda)	
Stages	Edit
Stages	
• \$default (Auto-deploy: enabled)	
Create	

Steg 9: Du bör nu se vår nyskapade API som enligt bilden nedan:

API Gateway > APIs	
APIs (1/1)	
<input type="button" value="Delete"/>	<input type="button" value="Create API"/>
<input type="button" value="Find APIs"/>	< 1 >
Name	Description
contactHandlerFormAPI	dkt6vurI6i
ID	Protocol
	HTTP
API endpoint type	Created
Regional	2025-11-16

Steg 10: Gå sedan in APIIn och granska att vår "Routes" och "Integrations" har skapats korrekt:

Routes:

The screenshot shows the AWS API Gateway 'Routes' section. On the left, a sidebar lists 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', and an API named 'API: contactHandlerF...'. Under 'Develop', 'Routes' is selected. The main area shows a list titled 'Routes for contactHandlerFormAPI' with one item: 'POST /contact'. To the right, a 'Route details' panel shows the ARN as 'arn:aws:apigateway:eu-west-1::apis/dkt6vuri6i/routes/x80oho8' and the integration as 'ezc3neg'. Buttons for 'Delete', 'Edit', 'Stage: -', and 'Deploy' are at the top right.

AWS Lambda Integration:

The screenshot shows the AWS API Gateway 'Integrations' section. The sidebar includes 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', and the same 'API: contactHandlerF...' API. Under 'Develop', 'Integrations' is selected. The main area shows 'Routes for contactHandlerFormAPI' with a 'POST /contact' route. To the right, an 'Integration details for route' panel shows the Lambda function as 'contactFormHandler (eu-west-1)' and the integration ID as 'ezc3neg'. It also includes sections for 'Description', 'Payload format version', 'Invoke permissions', 'Example policy statement', 'Timeout', 'Request parameter mapping', and 'Response parameter mappings', all currently set to 'Not configured'. Buttons for 'Attach integrations to routes', 'Manage integrations', 'Detach integration', and 'Manage integration' are at the top right.

Steg 11: Slutligen behöver vi sätta upp CORS så att Lambda-funktionen kan nås genom vår webbapplikation

Fyll i:

- **Access-Control-Allow-Origin** – S3-bucket: <http://serverless-bucket-2025.s3-website-eu-west-1.amazonaws.com/>
- **Access-Control-Allow-Methods** – POST, OPTIONS
- **Access-Control-Allow-Headers** - content-type
- Spara med "Save"
- Detta gör att endast S3-domänen kan använda vår Lambda-funktion

För tillfället anger vi vår S3-bucket, men vi kommer byta denna senare till CloudFront-distributionen (såsom jag har det konfigurerat enligt bilden) när vi konfigurerat upp CloudFront.

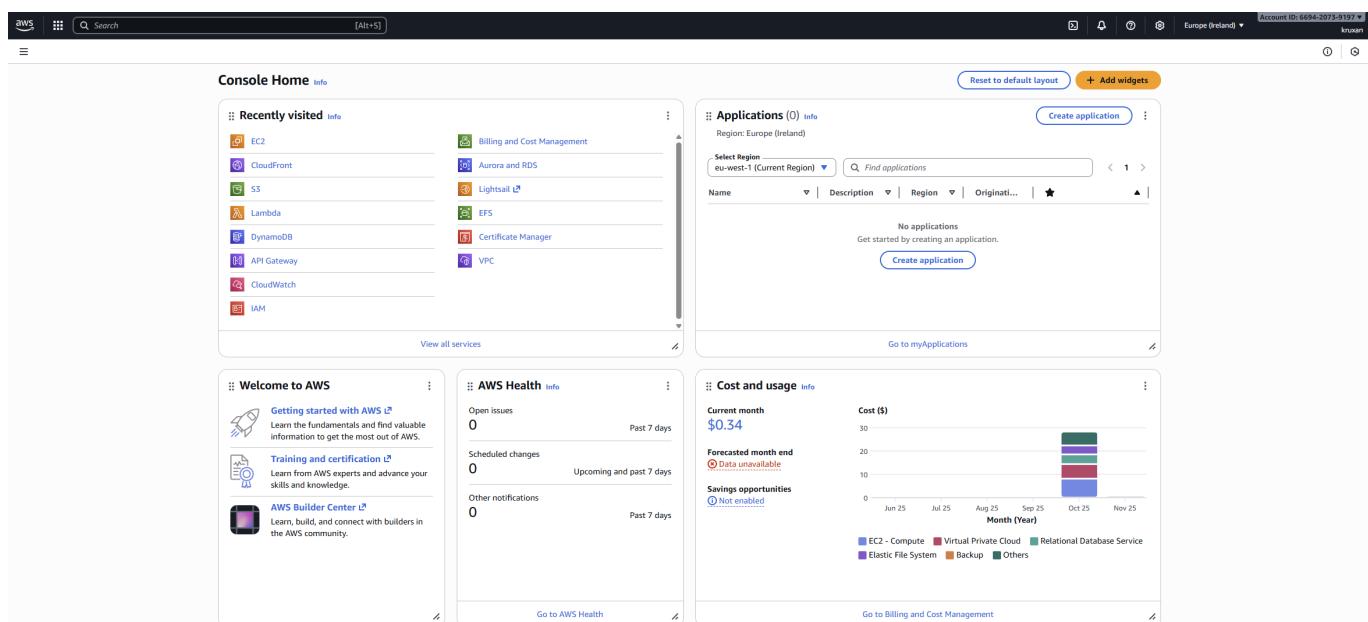
Steg 12: Slutligen kopiera APIs "Default endpoint" och ersätt följande API-URL på raden i contact_form.html som innehåller:

```
const apiUrl = "https://dkt6vuri6i.execute-api.eu-west-1.amazonaws.com/contact";
```

Konfiguration av Amazon CloudFront som Reverse proxy med HTTPS

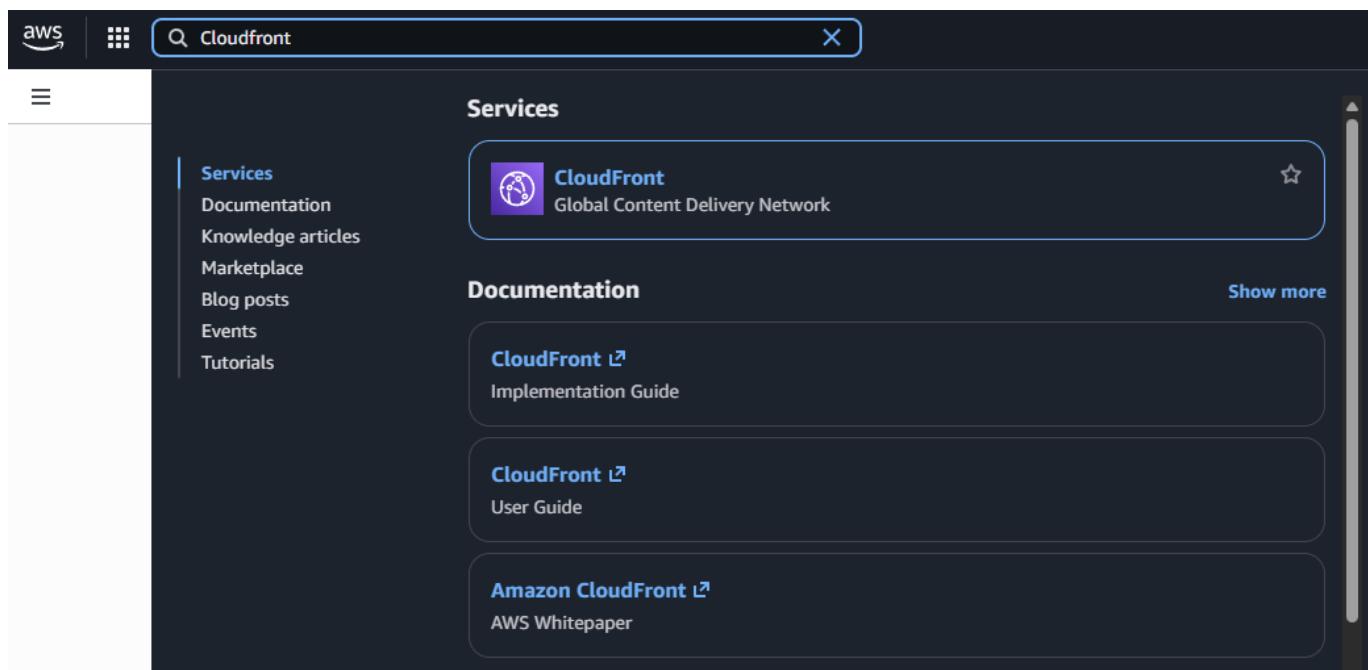
Denna guide beskriver hur man konfigurerar **Amazon CloudFront** för att distribuera frontend-filer från S3 och ge säker åtkomst via HTTPS. Målet är att skapa en snabb, säker och skalbar distribution av webbapplikationens statiska innehåll. CloudFront fungerar som en reverse proxy som hanterar HTTPS-anslutningar, och som säkerställer att användare alltid får en pålitlig och krypterad anslutning till webbapplikationen.

Steg 1: Bege dig till aws.amazon.com



The screenshot shows the AWS Console Home page. On the left, there's a sidebar with 'Recently visited' services: EC2, CloudFront, S3, Lambda, DynamoDB, API Gateway, CloudWatch, and IAM. Below this is a 'Welcome to AWS' section with links to 'Getting started with AWS', 'Training and certification', and 'AWS Builder Center'. The main area has sections for 'AWS Health' (Open issues: 0, Past 7 days), 'Cost and usage' (Current month: \$0.34, Forecasted month end: Data unavailable, Savings opportunities: Not enabled), and 'Billing and Cost Management'. A search bar at the top right says 'Search [Alt+S]' and shows 'Europe (Ireland)' and 'Account ID: 6694-2073-9197'.

Steg 2: Ange Cloudfront i sökrutan och välj "CloudFront - Global Content Delivery Network"



The screenshot shows the AWS Services search results for 'Cloudfront'. The search bar at the top contains 'Cloudfront'. The results are categorized under 'Services' and 'Documentation'. Under 'Services', there is a card for 'CloudFront - Global Content Delivery Network'. Under 'Documentation', there are three cards: 'CloudFront Implementation Guide', 'CloudFront User Guide', and 'Amazon CloudFront AWS Whitepaper'. A sidebar on the left lists 'Services', 'Documentation', 'Knowledge articles', 'Marketplace', 'Blog posts', 'Events', and 'Tutorials'.

Steg 3: Välj "Create a CloudFront distribution"

Networking & Content Delivery

Amazon CloudFront

Securely deliver content with low latency and high transfer speeds

Amazon CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency and high transfer speeds.

Get started with CloudFront

Enable accelerated, reliable and secure content delivery for Amazon S3 buckets, Application Load Balancers, Amazon API Gateway APIs, and more in 5 minutes or less.

[Create a CloudFront distribution](#)

Steg 4: För betalningsplan är enklast att välja "pay-as-you-go" för vårt ändamål eftersom vi kommer ändå inte hantera större mängder trafik

Choose a plan

<input type="radio"/> Free \$0/month <small>For hobbyists, learners, and developers getting started.</small> Usage allowance 1M requests / 100GB per month	<ul style="list-style-type: none"> ✓ Always-on DDoS protection ✓ Protect against common web threats with AWS WAF ✓ IP-based rate limiting ✓ Geographic traffic blocking ✓ Serverless edge compute ✓ Smart routing ✓ Global CDN 	<ul style="list-style-type: none"> ✓ DNS ✓ Free TLS certificate ✓ Tiered caching ✓ Default caching rules ✓ Fast cache invalidations ✓ 5GB S3 storage included
<input type="radio"/> Pro \$15/month <small>Launch and grow small websites, blogs, and applications.</small> Everything in Free, plus: <ul style="list-style-type: none"> ✓ Protections for WordPress, PHP, and SQL databases ✓ Header-based threat filtering ✓ Edge key-value store ✓ Logging ✓ 50GB S3 storage included Usage allowance 10M requests / 50TB per month	<input type="radio"/> Business \$200/month <small>Protect and accelerate business applications.</small> Everything in Pro, plus: <ul style="list-style-type: none"> ✓ Advanced DDoS protection ✓ Bot management and analytics ✓ Regex-based threat filtering ✓ JavaScript challenge ✓ Custom caching rules ✓ Private origins within VPC ✓ Uptime SLA ✓ 1TB S3 storage included Usage allowance 125M requests / 50TB per month	<input type="radio"/> Premium \$1000/month <small>Scale and protect business and mission-critical applications.</small> Everything in Business, plus: <ul style="list-style-type: none"> ✓ High-speed origin routing ✓ Origin load reduction ✓ Automatic origin failover ✓ 5TB S3 storage included Usage allowance 500M requests / 50TB per month
<p><input checked="" type="radio"/> Pay as you go <small>Pricing varies with usage</small></p> <p>Pay only for what you use based on traffic and enabled features related to this distribution in CloudFront, AWS WAF, Route 53, S3, and CloudWatch Logs. Choose this option if you serve more than 50TB or 500M requests per month, need control over feature selection (including features not available in pricing plans), or want to use your custom rates.</p>		

[Cancel](#)[Next](#)

Steg 5: Ange ett namn för vår CloudFront-distribution.

Jag kommer namnge den "serverless-app-cloudfront"

Get started

Connect your websites, apps, files, video streams, and other content to CloudFront. We optimize the performance, reliability, and security for your web traffic.

Distribution options Info

Distribution name

Name will be stored as a tag on the resource. You can change the name, or more tags, later.

Description - optional

Distribution type

Single website or app

Choose if each website or application will have a unique configuration.

Multi-tenant architecture - New

Choose when you have multiple domains that need to share configurations. This is a common architecture for SaaS providers.

Domain Info

Route 53 managed domain - optional

Enter a domain that's already registered with Route 53 in your AWS account. CloudFront will provision a TLS certificate for you. If you have a domain from a different DNS provider, skip this step and configure your domain later.

► Tags - optional

Steg 6: Välj sedan "Amazon S3" och bläddra fram vår S3-bucket. Resten kan du lämna som det är

Specify origin

Origin type

Origin type

Amazon S3

Deliver static assets like files and images, statically generated websites or single page applications (SPA).

Elastic Load Balancer

Deliver applications hosted behind ELB such as dynamic websites, web services, and APIs.

API Gateway

Deliver API endpoints for REST APIs hosted on API Gateway.

Elemental MediaPackage

Deliver end-to-end live events or video on demand (VOD).

VPC origin

Deliver applications and content hosted within private VPCs, such as EC2 instances and Application Load Balancers.

Other

Refer to any AWS or non-AWS origin through its publicly resolvable URL.

Origin

S3 origin

Choose an AWS origin, or enter your origin's domain name. [Learn more](#)

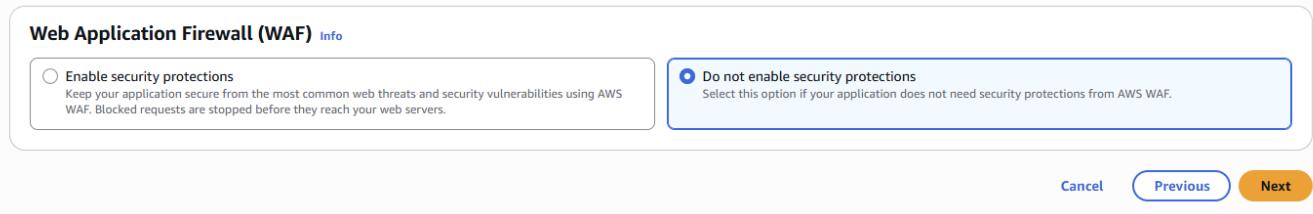
⚠ This S3 bucket has static web hosting enabled. If you plan to use this distribution as a website, we recommend using the S3 website endpoint rather than the bucket endpoint.

Origin path - optional

The directory path within your origin where your content is stored. [Learn more](#)

Steg 7: Vi kan hoppa över att sätta upp säkerheten med WAF eftersom vår CloudFront är ändå endast i testnings-syfte

Enable security



Steg 8: Du får nu översikt över vår CloudFront-distribution.

- Gå vidare genom att välja "Create distribution"

Review and create

General configuration

Distribution name serverless-app-cloudfront	Description -	Billing Pay-as-you-go (\$0/month)
--	------------------	--------------------------------------

Origin

S3 origin serverless-bucket-2025.s3.eu-west-1.amazonaws.com	Origin path -	Grant CloudFront access to origin Yes	Enable Origin Shield No
Connection attempts 3	Connection timeout 10		

Cache settings

CloudFront will apply default cache settings tailored to serving content from a S3 origin. You can customize settings after you create your distribution.

Security

Security protections None	Use monitor mode No	Use existing WAF configuration No
------------------------------	------------------------	--------------------------------------

Cancel Previous Create distribution

Steg 9: Slutligen bör du se en översikt över CloudFront-distributionen vi precis skapade

Distributions (1) Info

ID	Status	Description	Type	Domain name (standard)	Alternate domain n...	Origins	Prk
EA40LJKJ8UZM7	Enabled	-	Standard	d3vij5bvefx3w.cloudfront.net	-	serverless-bucket-2025.s3.eu-west-1.amazon	Pay

Enable Disable Delete Create distribution

Steg 10: Gå in på vår CloudFront-distribution och börja med att lägga till "index.html" för "Default root object" genom att navigera till "Edit" längst bort till höger

The screenshot shows the 'Settings' tab of a CloudFront distribution. On the right side, there is a blue 'Edit' button. Other visible settings include 'Name: serverless-app-cloudfront', 'Description: -', 'Price class: Use all edge locations (best performance)', 'Supported HTTP versions: HTTP/2, HTTP/1.1, HTTP/1.0', 'Alternate domain names' section with an 'Add domain' button, and a 'Default root object' set to 'index.html'. Standard and cookie logging are both set to 'Off'.

Steg 11: Ange sedan "index.html" för "Default root object"

Edit settings

The screenshot shows the 'Edit settings' page for the CloudFront distribution. Under the 'General' tab, the 'Default root object' field is set to 'index.html'. Other settings shown include 'AWS WAF web ACL - optional' (choose 'Choose web ACL'), 'Alternate domain name (CNAME) - optional' (add 'Add item'), 'Custom SSL certificate - optional' (choose 'Choose certificate' or 'Request certificate'), 'Supported HTTP versions' (selected: 'HTTP/2'), and 'Description - optional'.

Steg 12: Nu behöver vi lägga till vår API som vi skapade tidigare som en origin.

Gör detta genom att navigera till Origins -> "Create origin"

- Origin behövs i CloudFront för att tala om var innehållet ska hämtas ifrån när det inte finns i cachen.

The screenshot shows the 'Origins' section in the AWS CloudFront console. It lists two origins: 'serverless-bucket-2025.s3-website-eu-west-1.amazonaws.com-mi342b31xjm' (S3 type) and 'API Gateway' (API Gateway type). A 'Create origin' button is visible at the top right. The tabs at the top are General, Security, Origins (selected), Behaviors, Error pages, Invalidations, Logging, and Tags.

Steg 13: Välj vår API Gateway i dropdown-listan när du väljer "Origin domain"
Den kommer automatiskt generera din API-url som bilden nedan visar. Resten kan du lämna som det är

CloudFront < Create origin

Settings

Origin domain
Choose an AWS origin, or enter your origin's domain name. [Learn more](#)

[X](#)

Enter a valid DNS domain name, such as an S3 bucket, HTTP server, or VPC origin ID.

Protocol | [Info](#)

- HTTP only
- HTTPS only
- Match viewer

HTTPS port
Enter your origin's HTTPS port. The default is port 443.

443

Minimum Origin SSL protocol
The minimum SSL protocol that CloudFront uses with the origin.

- TLSv1.2
- TLSv1.1
- TLSv1
- SSLv3

Origin path - optional
Enter a URL path to append to the origin domain name for origin requests.

Name
Enter a name for this origin.

dkt6vuri6i.execute-api.eu-west-1.amazonaws.com

**Steg 14: Du bör nu ha två origins för din CloudFront-distribution.
En för din S3-bucket, och en för din API**

Origins (2)			
<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Create origin"/> < 1 > Filter			
Origin name	Origin domain	Origin path	Origin type
<input type="radio"/> serverless-bucket-2025.s3-website-eu-west-1.amazonaws.com-mi342b31xjm	serverless-bucket-2025.s3.eu-west-1.amazonaws.com		S3
<input type="radio"/> API Gateway	dkt6vuri6i.execute-api.eu-west-1.amazonaws.com		API Gateway

Steg 15: Slutligen behöver vi även lägga till två behaviors.

Återigen, en för din S3-bucket, och en för din API.

Gör detta genom att navigera till Behaviors -> "Create behavior"

- Behaviors i CloudFront används för att definiera hur olika URL-mönster hanteras, t.ex. vilken origin som används, hur innehåll cachelagras och vilka säkerhetsinställningar som gäller, såsom krav på HTTPS.

The screenshot shows the 'Behaviors' tab selected in the CloudFront console. There are three behaviors listed:

Index	Path pattern	Origin or origin group	Viewer protocol policy	Cache policy name	Origin request policy name	Response headers policy name
0	/	serverless-bucket-2025.s3-w...	Redirect HTTP to HTTPS	Managed-CachingOptimized	-	-
1	/api/*	API Gateway	HTTPS only	Managed-CachingDisabled	Managed-AllViewerExceptHostHea	-
2	Default (*)	serverless-bucket-2025.s3-w...	Redirect HTTP to HTTPS	Managed-CachingOptimized	-	-

Buttons at the top right include Save, Move up, Move down, Edit, Delete, and Create behavior.

Fyll i följande för S3-bucket:

- Path pattern** – /
- Origin and origin groups** – Välj din S3-bucket
- Viewer protocol policy** - Redirect HTTP to HTTPS
- Allowed HTTP methods** - GET, HEAD
- Cache policy** - CachingOptimized
- Spara med "**Save changes**"

Edit behavior

The screenshot shows the 'Edit behavior' page for a behavior named '/'. The 'Settings' tab is active.

Path pattern: /

Origin and origin groups: serverless-bucket-2025.s3-website-eu-west-1.amazonaws.com-mi342b31xjm

Compress objects automatically: Yes (selected)

Viewer

- Viewer protocol policy**: Redirect HTTP to HTTPS (selected)
- Allowed HTTP methods**: GET, HEAD (selected)

Restrict viewer access: No (selected)

Cache key and origin requests

We recommend using a cache policy and origin request policy to control the cache key and origin requests.

- Cache policy and origin request policy (recommended) (selected)
- Legacy cache settings

Cache policy: Choose an existing cache policy or create a new one.

CachingOptimized: Policy with caching enabled. Supports Gzip and Brotli compression. Recommended for S3 (selected)

Create cache policy ▾ View policy ▾

Fyll i följande för APIn:

- **Path pattern** – `/api/*`
- **Origin and origin groups** – Välj din API Gateway
- **Viewer protocol policy** - `HTTPS only`
- **Allowed HTTP methods** - `GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE`
- **Cache policy** - `CachingDisabled`
- **Origin request policy** - `AllViewerExceptHostHeader`
- Spara med "Save changes"

Edit behavior

Settings

Path pattern | [Info](#)
 X

Origin and origin groups
 ▼

Compress objects automatically | [Info](#)
 No
 Yes

Viewer

Viewer protocol policy
 HTTP and HTTPS
 Redirect HTTP to HTTPS
 HTTPS only

Allowed HTTP methods
 GET, HEAD
 GET, HEAD, OPTIONS
 GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE

Cache HTTP methods
 GET and HEAD methods are cached by default.
 OPTIONS

Allow gRPC requests over HTTP/2 | [Info](#)
 Enable

Restrict viewer access
If you restrict viewer access, viewers must use CloudFront signed URLs or signed cookies to access your content.
 No
 Yes

Cache key and origin requests
We recommend using a cache policy and origin request policy to control the cache key and origin requests.

Cache policy and origin request policy (recommended)
 Legacy cache settings

Cache policy
Choose an existing cache policy or create a new one.
 Recommended for API Gateway ▾ (C)
[Create cache policy](#) [View policy](#)

Origin request policy - optional
Choose an existing origin request policy or create a new one.
 Recommended for API Gateway ▾ (C)
[Create origin request policy](#) [View policy](#)

Steg 16: Slutligen, ifall du redan ersatt API-URL på raden i `contact_form.html` som innehåller följande med din API "Default endpoint":

- Så är det bara slutligen ersätta din S3-bucket URL som vi angav tidigare i `contactFormHandler.js` med din CloudFront-distributions URL, exempelvis: <https://d3vbj5bvefx3w.cloudfront.net>

```
const apiUrl = "https://dkt6vuri6i.execute-api.eu-west-1.amazonaws.com/contact";
```

Uppsättning av AWS CodePipeline för CI/CD

Denna guide beskriver hur man skapar en CI/CD-pipeline med **AWS CodePipeline** kopplad till ett GitHub-repository. Målet är att automatisera bygg och deployment av både frontend-filer till S3 och backend-funktioner till Lambda. Den säkerställer att ändringar i koden automatiskt testas, byggs och distribueras, vilket gör att nya funktioner snabbt och på ett pålitligt sätt blir tillgängliga i produktionsmiljön.

- Notera för att ansluta GitHub ihop med CodePipeline på AWS behövs följande connector:
<https://github.com/marketplace/aws-connector-for-github>

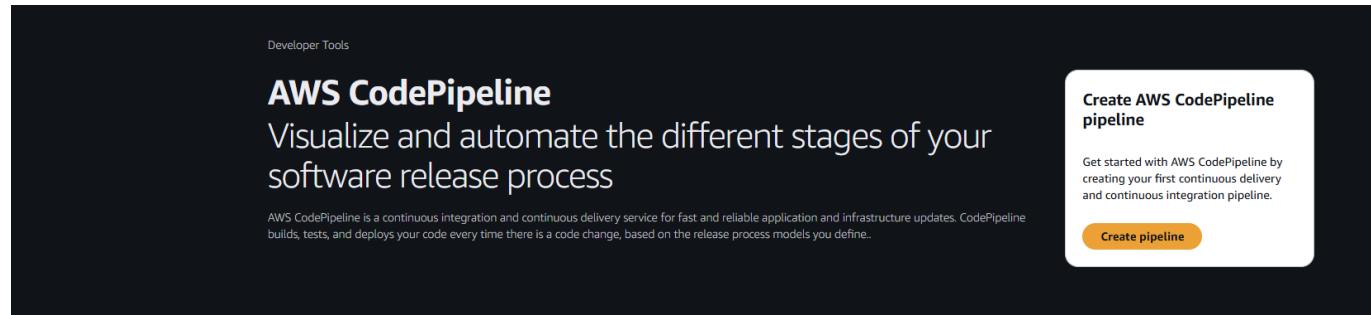
Steg 1: Bege dig till aws.amazon.com

The screenshot shows the AWS Console Home page. On the left, there's a sidebar with links to various services like EC2, CloudFront, S3, Lambda, DynamoDB, API Gateway, CloudWatch, and IAM. The main area has three main sections: 'Applications' (empty), 'Cost and usage' (showing current month cost of \$0.34 and a bar chart for Oct 25), and 'AWS Health' (showing 0 open issues and scheduled changes). There are also 'Welcome to AWS' and 'Getting started with AWS' cards.

Steg 2: Ange Codepipeline i sökrutan och välj "CodePipeline - Release Software using Continuous Delivery"

The screenshot shows the AWS Marketplace search results for 'Codepipeline'. The top result is 'CodePipeline' with the description 'Release Software using Continuous Delivery'. Below it, there are other resources listed under 'Resources in eu-west-1': 'codepipeline-source-trail' (CloudTrail trail), 'AmazonS3Pipeline' (CodePipeline), and 'codepipeline-78334589-artifactsbucket-rule' (Amazon EventBridge rule).

Steg 3: Välj "Create pipeline"



Steg 4: Välj därefter "Build custom pipeline" under Category

This screenshot shows the "Choose creation option" step of the AWS CodePipeline wizard, which is the first step of a 7-step process. On the left, a vertical sidebar lists steps from 1 to 7: Step 1 (Choose creation option), Step 2 (Choose pipeline settings), Step 3 (Add source stage), Step 4 (Add build stage), Step 5 (Add test stage), Step 6 (Add deploy stage), and Step 7 (Review). The current step, Step 1, is highlighted. The main content area has a title "Choose creation option" with an "Info" link, and a sub-instruction "Step 1 of 7". It features a "Category" section with three options: "Deployment", "Continuous Integration", and "Automation", with "Build custom pipeline" selected. At the bottom right are "Cancel" and "Next" buttons.

Steg 5: Ange ett namn för vår CI/CD Pipeline, jag kommer namnge den AmazonS3Pipeline

Välj/Fyll i även följande:

- **Execution Mode** – **Queued**
- **New Service Role** – Låt AWS CodePipeline skapa en IAM-roll åt dig med korrekta rättigheter
- Navigera sedan ner till **Advanced settings** och välj **Custom location**
- Välj därefter din S3-bucket som innehåller dina artifacts för **Custom location**

Du behöver nämligen ha en S3-bucket för att lagra dina artifacts.

Skapa helt enkelt en S3-bucket som tidigare och ge den ett passande, jag döpte min till **artifacts-bucket-2025**

Step 1
Choose creation option

Step 2
Choose pipeline settings

Step 3
Add source stage

Step 4
Add build stage

Step 5
Add test stage

Step 6
Add deploy stage

Step 7
Review

Choose pipeline settings Info
Step 2 of 7

Pipeline settings

Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.
 No more than 100 characters

Execution mode Info
Choose the execution mode for your pipeline. This determines how the pipeline is run.
 Superseded
 Queued
 Parallel

Service role
 New service role
Create a service role in your account
 Existing service role
Choose an existing service role from your account

Role name

 Type your service role name
 Allow AWS CodePipeline to create a service role so it can be used with this new pipeline

Advanced settings
Configure artifact store location, encryption settings, and pipeline variables for your pipeline.

Artifact store
 Default location
Create a default S3 bucket in your account.
 Custom location
Choose an existing S3 location from your account in the same region and account as your pipeline

Bucket

Encryption key
 Default AWS Managed Key
Use the AWS managed customer master key for CodePipeline in your account to encrypt the data in the artifact store.
 Customer Managed Key
To encrypt the data in the artifact store under an AWS KMS customer managed key, specify the key ID, key ARN, or alias ARN.

S3-artifacts i CI/CD är helt enkelt filer som din bygg- och deployprocess sparar i ett tryggt förråd under arbetets gång.

Tänk dig att din CI/CD-pipeline bygger något — till exempel en app, en konfigurationsfil eller ett paket. Resultatet behöver sparas någonstans så att nästa steg i processen kan använda det.

Amazon S3 fungerar då som **en gemensam lagringsplats** där pipelinen kan lägga sina filer och hämta dem när de behövs.

Kort sagt: S3-artifacts är filer som CI/CD-systemet lagrar i S3 så att de kan användas och delas mellan olika steg i automatiseringskedjan.

Steg 6: När vi kommer till "Add source stage" är det dags att koppla samman vår GitHub-repo och AWS CodePipeline

- För att AWS CodePipeline ska kunna hämta koden från GitHub behöver du skapa och välja en anslutning till ditt GitHub-konto. Detta görs genom att:

Välja följande:

- Source provider** – GitHub (via GitHub App)
- Connection** – Klicka på "Connect to GitHub"
- Autentisera med ditt GitHub-konto och ge AWS CodePipeline nödvändiga behörigheter.
- När anslutningen är etablerad kan du välja repository och branch som ska användas som källkod för pipelinen enligt nedan.
- Repository name** – Repot som ska användas för AWS CodePipline
- Default branch** – main eller master (troligtvis main)

Step 1 Choose creation option

Step 2 Choose pipeline settings

Step 3 Add source stage

Step 4 Add build stage

Step 5 Add test stage

Step 6 Add deploy stage

Step 7 Review

Add source stage Info

Step 3 of 7

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (via GitHub App) ▾

Connection
Choose an existing connection that you have already configured, or create a new one and then return to this task.

arn:aws:codeconnections:eu- or

Repository name
Choose a repository in your GitHub account.

91maxore-hub/serverless-app

You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.

Default branch
Default branch will be used only when pipeline execution starts from a different source or manually started.

main

Output artifact format
Choose the output artifact format.

CodePipeline default
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.

Full clone
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions. [Learn more ↗](#)

Enable automatic retry on stage failure

Add test stage och Add build stage kan vi skippa

Steg 7: När vi når "Add deploy stage" behöver vi ange vilken S3-bucket AWS CodePipeline ska använda i CI/CD-deployprocessen. Detta görs genom att specificera vår S3-bucket under fältet "Bucket"

Resten kan du lämna som det är.

The screenshot shows the 'Add deploy stage' configuration screen in AWS CodePipeline. On the left, a vertical sidebar lists steps from 1 to 7: Step 1 (Choose creation option), Step 2 (Choose pipeline settings), Step 3 (Add source stage), Step 4 (Add build stage), Step 5 (Add test stage), Step 6 (Add deploy stage, which is currently selected), and Step 7 (Review). The main area is titled 'Add deploy stage' with an 'Info' link. It shows 'Step 6 of 7'. A callout box highlights a warning: 'You cannot skip this stage' with the note: 'Pipelines must have at least two stages. Your second stage must be either a build, test or deployment stage. Choose a provider for either the build stage, test stage or deployment stage.' Below this, the 'Deploy' section is shown. Under 'Deploy provider', 'Amazon S3' is selected. Under 'Region', 'Europe (Ireland)' is selected. In the 'Input artifacts' section, a 'SourceArtifact' is defined, which is 'Defined by: Source'. There is a note: 'No more than 100 characters'. Under 'Bucket', 'serverless-bucket-2025' is entered. In the 'Deployment path - optional' section, there is an empty input field. At the bottom, there are three checkboxes: 'Extract file before deploy' (checked), 'Configure automatic rollback on stage failure' (checked), and 'Enable automatic retry on stage failure' (unchecked). Navigation buttons at the bottom right include 'Cancel', 'Previous', and 'Next'.

Steg 8: Du får nu översikt över vår AWS CodePipeline. Gå vidare genom att välja "Create pipeline"

Step 1
[Choose creation option](#)

Step 2
[Choose pipeline settings](#)

Step 3
[Add source stage](#)

Step 4
[Add build stage](#)

Step 5
[Add test stage](#)

Step 6
[Add deploy stage](#)

Step 7
Review

Review [Info](#)

Step 7 of 7

Step 2: Choose pipeline settings

Pipeline settings

Pipeline name
AmazonS3Pipeline2

Pipeline type
V2

Execution mode
QUEUED

Artifact location
artifacts-bucket-2025

Service role name
AWSCodePipelineServiceRole-eu-west-1-AmazonS3Pipeline2

Step 3: Add source stage

Source action provider

Source action provider
GitHub (via GitHub App)

OutputArtifactFormat
CODE_ZIP

DetectChanges
true

ConnectionArn
arn:aws:codeconnections:eu-west-1:669420739197:connection/6d033439-6143-4d4e-8d6a-e722a9b8a156

FullRepositoryId
91maxore-hub/serverless-app

Default branch
main

Enable automatic retry on stage failure
Enabled

Steg 9: Slutligen bör du se en översikt över AWS CodePipelinen vi precis skapade som kommer hantera CI/CD deployment

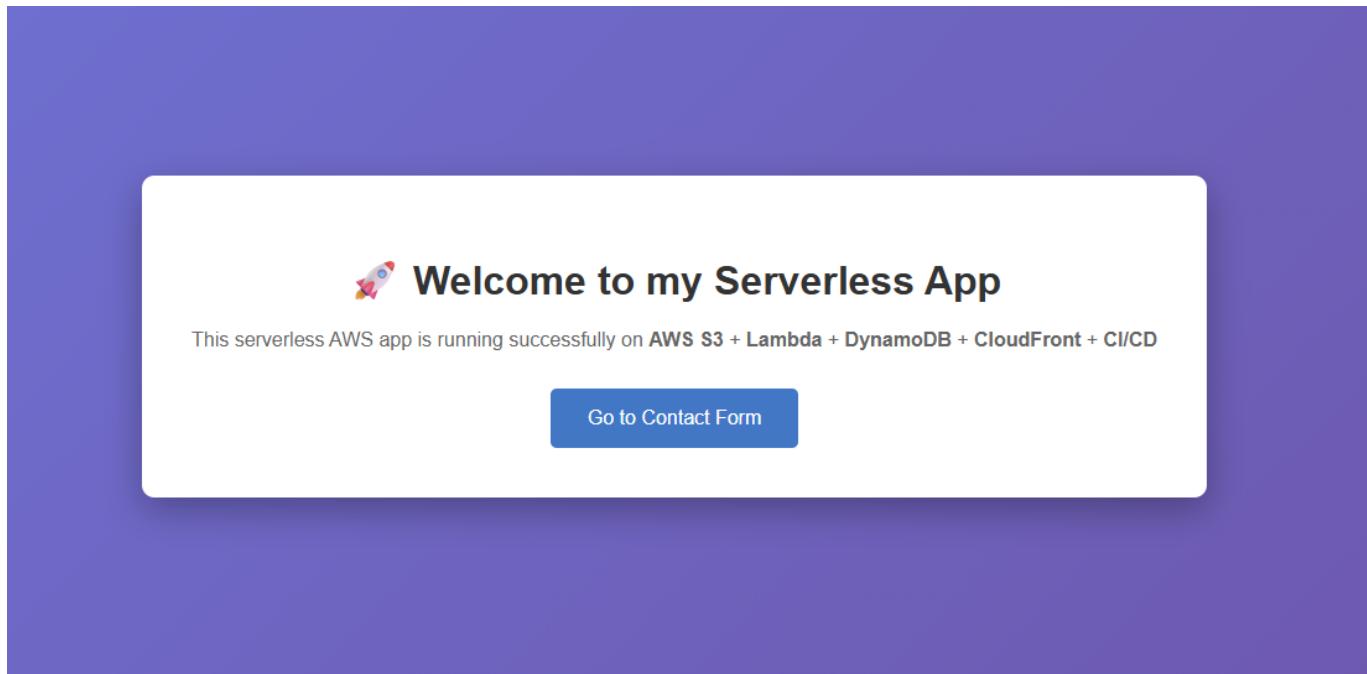
- Som du kan se har den senaste CI/CD-körningen slutförts korrekt med statusen: **Succeeded**

Name	Latest execution status	Latest source revisions	Latest execution started	Most recent executions
AmazonS3Pipeline	Succeeded	Source - face7f9212 - CI/CD Pipeline	1 day ago	5

Resultat

Efter att allt var uppsatt och CI/CD-deployment gick igenom kunde jag gå till: [🔗](https://d3vbj5bvefx3w.cloudfront.net)
https://d3vbj5bvefx3w.cloudfront.net

Min PHP-app laddas med giltigt SSL-certifikat, automatisk HTTPS och reverse proxy som hanterar trafiken smidigt genom CloudFront. Allt detta sker helt automatiskt – både deployment och certifikatförflytelse.



Användningen av Infrastructure as Code (IaC)

Jag använder Infrastructure as Code (IaC) för att definiera och hantera mina **AWS Lambda-funktioner**, där funktionens kod, miljövariabler och konfiguration beskrivs som kod. Mitt Lambda-skript tar emot inkommande HTTP-förfrågningar via API Gateway, hanterar CORS, validerar JSON-data och skriver formulärsvar till **DynamoDB** med unika ID:n. Genom IaC kan denna funktion automatiskt distribueras och konfigureras på ett konsekvent sätt, vilket säkerställer att den alltid körs korrekt och med rätt åtkomst till databasen, oavsett miljö.

Användning av säkerhet

Säkerheten i min serverless-lösning hanteras på flera nivåer. All trafik mellan användare och applikationen går via **CloudFront** med HTTPS, vilket säkerställer krypterad kommunikation och trygg åtkomst. **API Gateway** kontrollerar och hanterar HTTP-förfrågningar, vilket begränsar obehörig åtkomst till Lambda-funktionerna. Data som lagras i **DynamoDB** hanteras säkert och åtkomsten styrs via IAM-behörigheter, vilket skyddar informationen både i vila och under transport.

Genom att använda **CI/CD med AWS CodePipeline** säkerställs att alla uppdateringar till S3 och Lambda distribueras automatiskt och kontrollerat. Detta minimerar risken för manuella misstag och säkerställer att endast korrekt testad kod når produktion, vilket bidrar till en säkrare och mer pålitlig applikation.