**Engineer**

# serverless

☁️ å kjøre kode uten selv å drifte serverne under den

## Hva det er

Serverless er en måte å kjøre kode på der teamet skriver og produksjonssetter applikasjonen, mens en leverandør drifter, skalerer og vedlikeholder maskinene under den. På norsk omtales slike plattformer også som serverløse tjenester. Whitepaperen fra CNCF Serverless Working Group bruker det lengre navnet serverless computing og beskriver det som å bygge og kjøre applikasjoner uten at noen på teamet må administrere en server.

Navnet tar i, og whitepaperen sier det rett ut: Det trengs fortsatt servere for å kjøre en serverless-plattform. Serverne finnes fortsatt, og det gjør driftsarbeidet også. Det som endres, er hvem som gjør det. Teamet velger ikke lenger maskinstørrelser, installerer ikke operativsystemet, patcher det ikke og bestemmer ikke hvor mange kopier som skal kjøre. Det leverer kode eller et container-image og en beskrivelse av når koden skal kjøre.

Whitepaperen deler feltet i to. Functions-as-a-service, forkortet FaaS, er hendelsesdrevet databehandling: små biter logikk som kjører når noe skjer. Det som starter en funksjon, kalles en trigger, og den er som regel resultatet av en hendelse, for eksempel en innkommende HTTP-forespørsel eller en endring i en database. Backend-as-a-service, forkortet BaaS, er den andre delen: tredjepartstjenester som man når gjennom et API, og som erstatter hele deler av en applikasjons funksjonalitet. Funksjonene forventes å være tilstandsløse og kortlivede, så det er på BaaS-siden tilstanden bor.

To produkter viser spennvidden. AWS Lambda beskriver seg selv som en serverless beregningstjeneste som kjører kode uten at man må klargjøre eller administrere servere, og den opprinnelige formen er event-funksjonen: en handler pluss en trigger. Azure Container Apps kaller seg en serverless-plattform

for å kjøre containeriserte applikasjoner uten å administrere infrastrukturen under dem. Den første tar imot en funksjon, den andre en hel container, og begge overlater maskinene til leverandøren.

### Bruk begrepet i en beslutning

Bruk «Hva er serverless?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Servere dimensjonert etter det du faktisk bruker, flyttet inn i skyen eller ut av den igjen, med månedsregningen regnet ut på forhånd.

**Se tjenestene våre** Ta kontakt →

### Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Driftet 20 Docker-apper på Google Cloud og AWS — 20 containeriserte applikasjoner produksjonssatt og overvåket

**Driftet 20 Docker-apper på Google Cloud og AWS** Ta kontakt →

## Hvorfor det er viktig

Faktureringsmodellen er den første grunnen til at team ser på det. Whitepaperen beskriver prisingen som pay-as-you-go, der man bare betaler for tiden koden kjører, og ikke betaler noe mens den står stille. Lambda fakturerer en event-funksjon per forespørsel pluss kjøretiden den bruker, målt i gigabytesekunder. En tjeneste som har travelt én time i døgnet, betaler for den timen og ikke for en server som venter de andre 23.

Den andre grunnen er skalering i begge retninger. Lambda oppretter og fjerner kjøremiljøer etter hvert som trafikken stiger og faller, og whitepaperen beskriver funksjoner der antallet instanser faller til null når det ikke er arbeid å gjøre. Microsoft skriver at Container Apps skalerer på HTTP-trafikk, hendelser eller CPU- og minnebelastning, og at de fleste applikasjoner kan skalere helt ned til null.

Den tredje grunnen er arbeidet som forsvinner fra teamets bord. AWS nevner servervedlikehold, klargjøring av kapasitet, skalering og patching blant oppgavene Lambda tar seg av. For et lite team er det nettopp disse gjøremålene som kan fylle en hel uke uten at noe blir levert, og på en serverless-plattform blir de leverandørens ansvar.

Bruksområdene i whitepaperen viser hvor modellen passer best: å reagere på endringer i en database, behandle sensordata fra enheter, håndtere datastrømmer og korte planlagte oppgaver av den typen som ellers ville vært en cron-jobb. Microsofts liste for Container Apps legger til API-endepunkter, bakgrunnsjobber og mikrotjenester.

Den samme whitepaperen advarer også om at serverless kan se ut som det billigste alternativet i starten, og ber team se på hele kostnadsbildet før de bestemmer seg. Forbruksbasert fakturering lønner seg for arbeid som kommer i rykk og napp og står stille mesteparten av tiden. Arbeid som holder en maskin opptatt døgnet rundt, er et annet regnestykke, og feilsøking, som ifølge whitepaperen kan være vanskeligere enn på IaaS og PaaS, er også en kostnad.

## Slik ser det ut i praksis

AWS-dokumentasjonen for handlere i Python har et eksempel som tar imot en ordre, skriver en kvittering som tekst og lagrer den i en Amazon S3-bucket. En forkortet versjon ser slik ut:

```
import os

import boto3

# Kjører i Init-fasen, én gang per kjøremiljø.
s3_client = boto3.client("s3")


def lambda_handler(event, context):
    order_id = event["Order_id"]
    bucket_name = os.environ.get("RECEIPT_BUCKET")
    receipt_content = (
        f"OrderID: {order_id}\n"
        f"Amount: ${event['Amount']}\n"
        f"Item: {event['Item']}"
    )
    key = f"receipts/{order_id}.txt"
    s3_client.put_object(Bucket=bucket_name, Key=key, Body=receipt_content)
    return {"statusCode": 200, "message": "Receipt processed successfully"}
```

Handleren tar nøyaktig to argumenter. Det første inneholder dataene som skal behandles, og en hendelse i JSON kommer inn som en Python-dictionary, så denne funksjonen forventer input med denne formen:

```
{"Order_id": "12345", "Amount": 199.99, "Item": "Wireless Headphones"}
```

Det andre argumentet, kontekstobjektet, inneholder opplysninger om selve kallet. Heter filen `lambda_function.py`, får handleren navnet `lambda_function.lambda_handler`, som er standardnavnet Lambda-konsollen gir en

ny funksjon. Bucketnavnet kommer fra en miljøvariabel og ikke fra koden, slik AWS anbefaler å sende inn driftsinnstillinger, og koden inneholder ingen påloggingsopplysninger: Lambda gir dem til SDK-et gjennom sine egne miljøvariabler.

Hvor klienten opprettes, har betydning. For hendelsesfunksjoner med on-demand-kjøring uten SnapStart eller Managed Instances fører Lambda et kjøremiljø gjennom tre faser: Init, Invoke og Shutdown. Kode utenfor handleren kjører én gang under Init, og AWS begrenser det første Init-forsøket til 10 sekunder. Deretter kjører handleren for hver hendelse innenfor funksjonens tidsgrense. Når et miljø gjenbrukes til neste hendelse, det AWS kaller en warm start, finnes klienten fra Init fortsatt og bygges ikke på nytt.

Serverless trenger ikke å bety funksjoner. Porteføljeinnlegget om Azure-infrastruktur klargjort som kode nevner to tjenester, `api` og `www`, som kjører på Azure Container Apps, og der containerimagene ligger i prosjektets eget Azure Container Registry. Microsoft kaller Container Apps en serverless-plattform for containere, så det er de to tjenestene som utgjør den serverløse delen av oppsettet. Hvert miljø beskrives av ett sett Bicep-maler med parametere, og derfor kan innlegget behandle et nytt miljø som en pull request.

Den motsatte veien kan også være riktig. Oppføringen om å gjennomgå infrastrukturen og flytte arbeidslaster bort fra AWS tar utgangspunkt i et AWS-oppsett som var dimensjonert langt utover det virksomheten brukte, og fortsatt inneholdt tjenester ingen trengte. De ubrukte tjenestene ble fjernet, den gjenværende kapasiteten ble tilpasset, og arbeidslastene ble flyttet bort fra AWS. Serverless-fakturering er én måte å slutte å betale for ubrukt kapasitet på; å gjennomgå og fjerne ubrukt infrastruktur er en annen. En sammenligning av den reelle belastningen og kostnadene ville være nødvendig for å avgjøre hva som koster minst; oppføringen dokumenterer endringer, ikke et kostnadsforhold for sammenlignbare oppsett.

## Hvor det går galt

De fleste feil med serverless kommer av at man behandler en funksjon som en liten server som alltid står på, noe den ikke gjør.

**Kaldstarter forsinker noen forespørsler.** Når det ikke står et miljø klart, må Lambda hente koden, sette opp et miljø og kjøre Init-koden før handleren ser hendelsen. Ifølge AWS rammer kaldstarter vanligvis under 1 % av kallene og

varer fra under 100 millisekunder til over ett sekund; de er hyppigere i utviklings- og testfunksjoner, og Init-tiden faktureres. Løsningen er å holde Init liten, bare importere det funksjonen trenger, og først laste inn sjelden brukte objekter når de trengs. Der forsinkelsen betyr noe, holder provisioned concurrency et valgt antall miljøer initialisert på forhånd.

**Tilstand i miljøet lekker eller forsvinner.** Hvert kall kjører uavhengig, og et miljø blir kanskje gjenbrukt, kanskje ikke. AWS bytter dessuten ut miljøer med noen timers mellomrom for å oppdatere runtime, selv ved jevn trafikk, og en tilbakestilling etter et krasj eller et tidsavbrudd tømmer ikke katalogen `/tmp`. Data fra én hendelse kan derfor dukke opp under en annen eller forsvinne uten forvarsel. Løsningen er å holde brukerdata og hendelser helt utenfor miljøet og legge alt som skal bevares, i en database eller en lagringstjeneste.

**Den samme hendelsen behandles to ganger.** En funksjon kan motta duplikate hendelser, og AWS' liste over god praksis ber om idempotent kode som validerer hver hendelse og behandler et duplikat på samme måte som originalen. I eksemplet over er kvitteringens nøkkel avledet av ordrenummeret, så en ny kjøring overskriver det samme objektet i stedet for å lage enda en kvittering. Enhver sideeffekt som ikke tåler å bli gjentatt, som å sende en e-post eller belaste et kort, trenger sin egen registrering av hva som allerede er gjort.

**Harde grenser stopper arbeidet halvveis.** For hendelsesfunksjoner uten Lambda Managed Instances kan et kall kjøre i høyst 900 sekunder, altså 15 minutter. En synkron forespørsel og et bufret synkront svar er hver begrenset til 6 MB, en asynkron forespørsel til 1 MB og miljøvariabler til 4 KB til sammen. Å overskride en grense gir en feil, ikke en nedbremsing. Løsningen er å designe etter grensene fra starten av: dele lange jobber opp i trinn, lagre store datamengder i et lager og sende en referanse videre, og flytte arbeid som grensene ikke rommer, til en containerplattform eller en virtuell maskin.

**Samtidigheten tar slutt før trafikken gjør det.** Lambdas standardkvote er 1 000 samtidige kjøringer per region, mens API Gateway som standard tillater 10 000 forespørsler i sekundet, så inngangen kan ta imot flere forespørsler enn funksjonene bak den kan kjøre. Nye kontoer starter med reduserte kvoter for samtidighet og minne, som AWS hever automatisk basert på bruken. Løsningen er å be om en økning før lansering og å lastteste ved forventet toppbelastning.

**En funksjon som starter seg selv, får regningen til å løpe løpsk.** En funksjon som kaller seg selv, eller som setter i gang en prosess som kaller den igjen, kan utløse en flom av kall, og hvert eneste ett faktureres. AWS råder til å unngå rekursive kall og, hvis det plutselig kommer et uventet volum, straks å sette funksjonens reserved concurrency til `0`, noe som stopper alle kall til koden er rettet.

**Programmeringsmodellen binder deg til én plattform.** Whitepaperen peker på plattformens programmeringsmodell, grensesnittet for hendelser og meldinger og BaaS-tilbudet som kilder til innlåsing. Løsningen er å holde forretningslogikken i vanlige funksjoner som handleren bare kaller, og å foretrekke plattformer som Container Apps, som kjører et standard container-image fra et hvilket som helst registry, når en senere flytting er en reell mulighet.

## Les mer

- CNCF Serverless Whitepaper v1.0 — arbeidsgruppens definisjoner av serverless, FaaS og BaaS, og ulempene.
- What is AWS Lambda? — hva Lambda tar seg av, funksjonstypene og faktureringsmodellen.
- Define Lambda Event Function handler in Python — hele kvitteringseksemplet, navngivning av handlere og god praksis for koden.
- Understanding the Lambda execution environment lifecycle — fasene Init, Invoke og Shutdown, kaldstarter og gjenbruk av miljøer.
- Lambda quotas — grenser for samtidighet, minne, tidsavbrudd og payload, og hvordan de økes.
- Azure Container Apps overview — Microsofts serverless-plattform for containere, som kan skalere til null.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

**Se hva det koster** Ta kontakt →

# Gjør begrepene praktiske

Spør hvordan «Hva er serverless?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/serverless/