Engineer

serverless

att köra kod utan att själv sköta servrarna under den

Vad det är

Serverless är ett sätt att köra kod där teamet skriver och driftsätter applikationen medan en leverantör driver, skalar och underhåller maskinerna under den. På svenska talar man också om serverlösa tjänster. Whitepapern från CNCF Serverless Working Group använder det längre namnet serverless computing och beskriver det som att bygga och köra applikationer utan att någon i teamet behöver hantera en server.

Namnet tar i, och whitepapern säger det rakt ut: det behövs fortfarande servrar för att köra en serverless-plattform. Servrarna finns kvar, och det gör driftarbetet också. Det som ändras är vem som utför det. Teamet väljer inte längre maskinstorlekar, installerar inte operativsystemet, patchar det inte och bestämmer inte hur många kopior som ska köras. Det lämnar över kod eller en containeravbildning och en beskrivning av när koden ska köras.

Whitepapern delar upp fältet i två delar. Functions-as-a-service, förkortat FaaS, är händelsedriven databehandling: små stycken logik som körs när något händer. Det som startar en funktion kallas en trigger, och den är oftast resultatet av en händelse, till exempel en inkommande HTTP-förfrågan eller en ändring i en databas. Backend-as-a-service, förkortat BaaS, är den andra delen: tredjepartstjänster som man når via ett API och som ersätter hela delar av en applikations funktionalitet. Funktionerna förväntas vara tillståndslösa och kortlivade, så det är på BaaS-sidan som tillståndet bor.

Två produkter visar spännvidden. AWS Lambda beskriver sig själv som en serverless beräkningstjänst som kör kod utan att man behöver förbereda eller hantera servrar, och dess ursprungliga form är event-funktionen: en handler plus en trigger. Azure Container Apps kallar sig en serverless-plattform för att köra

containerbaserade applikationer utan att hantera infrastrukturen under dem. Den första tar emot en funktion, den andra en hel container, och båda lämnar maskinerna åt leverantören.

Använd begreppet i ett beslut

Använd ”Vad är serverless?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Servrar dimensionerade efter vad du faktiskt använder, flyttade till molnet eller tillbaka ut ur det, med månadskostnaden uträknad först.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Drev 20 Docker-appar på Google Cloud och AWS — 20 containerbaserade applikationer, driftsatta och övervakade

Drev 20 Docker-appar på Google Cloud och AWS Hör av er →

Varför det spelar roll

Debiteringsmodellen är det första skälet till att team tittar på det. Whitepapern beskriver prissättningen som pay-as-you-go, där man bara betalar för tiden koden körs och inte betalar något medan den står stilla. Lambda debiterar en event-funktion per förfrågan plus den körtid den använder, mätt i gigabytesekunder. En tjänst som har fullt upp en timme om dygnet betalar för den timmen och inte för en server som väntar de övriga 23.

Det andra skälet är skalning åt båda hållen. Lambda skapar och tar bort körmiljöer i takt med att trafiken ökar och minskar, och whitepapern beskriver funktioner vars antal instanser sjunker till noll när det inte finns något arbete. Microsoft skriver att Container Apps skalar på HTTP-trafik, händelser eller belastning på processor och minne, och att de flesta applikationer kan skala hela vägen ned till noll.

Det tredje skälet är arbetet som försvinner från teamets bord. AWS räknar upp serverunderhåll, kapacitetsplanering, skalning och patchning bland uppgifterna som Lambda sköter. För ett litet team är det just de sysslorna som kan fylla en hel vecka utan att något levereras, och på en serverless-plattform blir de leverantörens ansvar.

Användningsfallen i whitepapern visar var modellen passar bäst: att reagera på ändringar i en databas, behandla sensordata från enheter, hantera dataströmmar och korta schemalagda uppgifter av det slag som annars hade varit ett cron-jobb. Microsofts lista för Container Apps lägger till API-slutpunkter, bakgrundsjobb och mikrotjänster.

Samma whitepaper varnar också för att serverless kan se ut som det billigaste alternativet i början, och ber team att se på hela kostnadsbilden innan de bestämmer sig. Förbrukningsbaserad debitering lönar sig för arbete som kommer i skurar och står stilla större delen av tiden. Arbete som håller en maskin sysselsatt dygnet runt är en annan kalkyl, och felsökning, som enligt whitepapern kan vara svårare än på IaaS och PaaS, är också en kostnad.

Så ser det ut i praktiken

AWS dokumentation för handlers i Python har ett exempel som tar emot en order, skriver ett kvitto som text och sparar det i en Amazon S3-bucket. En förkortad version ser ut så här:

```
import os

import boto3

# Körs i Init-fasen, en gång per körmiljö.
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"}
```

Handlern tar exakt två argument. Det första innehåller de data som ska behandlas, och en händelse i JSON kommer in som en Python-dictionary, så den här funktionen väntar sig indata med följande form:

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

Det andra argumentet, kontextobjektet, innehåller uppgifter om själva anropet. Heter filen `lambda_function.py` får handlern namnet `lambda_function.lambda_handler`, vilket är standardnamnet som Lambda-konsolen

ger en ny funktion. Bucketnamnet kommer från en miljövariabel och inte från koden, så som AWS rekommenderar att driftinställningar skickas in, och koden innehåller inga inloggningsuppgifter: Lambda ger dem till SDK:t genom sina egna miljövariabler.

Var klienten skapas spelar roll. För en händelsefunktion som körs på begäran utan SnapStart eller Managed Instances går körmiljön i Lambda igenom tre faser: Init, Invoke och Shutdown. Kod utanför handlern körs en gång under Init, och AWS begränsar det första Init-försöket till 10 sekunder. Därefter körs handlern för varje händelse inom funktionens tidsgräns. När en miljö återanvänds för nästa händelse, det AWS kallar en warm start, finns klienten från Init kvar och byggs inte om.

Serverless behöver inte betyda funktioner. Portföljinlägget om Azure-infrastruktur provisionerad som kod nämner två tjänster, `api` och `www`, som körs på Azure Container Apps och vars containeravbildningar ligger i projektets eget Azure Container Registry. Microsoft kallar Container Apps en serverless-plattform för containrar, så det är de två tjänsterna som utgör serverless-delen av helheten. Varje miljö beskrivs av en enda uppsättning Bicep-mallar med parametrar, och därför kan inlägget behandla en ny miljö som en pull request.

Den motsatta vägen kan också vara den rätta. Inlägget om att granska infrastrukturen och flytta arbetslaster bort från AWS börjar med en AWS-miljö som var långt större än verksamheten använde och som fortfarande bar på tjänster som ingen behövde. De oanvända tjänsterna togs bort, återstående kapacitet anpassades och arbetslasterna flyttades bort från AWS. Serverless-debitering är ett sätt att slippa betala för ledig kapacitet; att granska och ta bort oanvänd infrastruktur är ett annat. En jämförelse av den verkliga belastningen och kostnaderna skulle behövas för att avgöra vilket som kostar minst; inlägget redovisar ändringar snarare än ett kostnadsförhållande på jämförbara grunder.

Var det går fel

De flesta fel med serverless kommer av att man behandlar en funktion som en liten server som alltid är igång, vilket den inte är.

Kallstarter fördröjer en del förfrågningar. När ingen miljö står redo måste Lambda hämta koden, sätta upp en miljö och köra Init-koden innan handlern ser händelsen. Enligt AWS drabbar kallstarter vanligtvis under 1 % av anropen och varar från under 100 millisekunder till över en sekund; de är vanligare i

utvecklings- och testfunktioner, och Init-tiden debiteras. Lösningen är att hålla Init liten, bara importera det som funktionen behöver och läsa in sällan använda objekt först när de behövs. Där fördröjningen spelar roll håller provisioned concurrency ett valt antal miljöer initierade i förväg.

Tillstånd i miljön läcker eller försvinner. Varje anrop körs oberoende, och en miljö kanske återanvänds, kanske inte. AWS byter dessutom ut miljöer med några timmars mellanrum för att uppdatera runtime, även vid jämn trafik, och en återställning efter en krasch eller en timeout tömmer inte katalogen `/tmp`. Data från en händelse kan därför dyka upp under en annan eller försvinna utan förvarning. Lösningen är att hålla användardata och händelser helt utanför miljön och lägga allt som ska bevaras i en databas eller en lagringstjänst.

Samma händelse behandlas två gånger. En funktion kan ta emot dubblerade händelser, och AWS lista över god praxis ber om idempotent kod som validerar varje händelse och hanterar en dubblett på samma sätt som originalet. I exemplet ovan härleds kvittots nyckel från ordernumret, så en ny körning skriver över samma objekt i stället för att skapa ännu ett kvitto. Varje sidoeffekt som inte tål att upprepas, som att skicka ett e-postmeddelande eller dra pengar från ett kort, behöver en egen registrering av vad som redan är gjort.

Hårda gränser stoppar arbetet halvvägs. För händelsefunktioner utanför Lambda Managed Instances kan ett anrop köras i högst 900 sekunder, alltså 15 minuter. En synkron förfrågan och ett buffrat synkront svar är vardera begränsade till 6 MB, en asynkron förfrågan till 1 MB och miljövariabler till 4 KB sammanlagt. Att överskrida en gräns ger ett fel, inte en inbromsning. Lösningen är att utforma systemet efter gränserna från början: dela upp långa jobb i steg, lägga stora datamängder i en lagringstjänst och skicka vidare en referens, och flytta arbete som inte ryms inom gränserna till en containerplattform eller en virtuell maskin.

Samtidigheten tar slut före trafiken. Lambdas standardkvot är 1 000 samtidiga körningar per region, medan API Gateway som standard tillåter 10 000 förfrågningar per sekund, så ingången kan ta emot fler förfrågningar än funktionerna bakom den kan köra. Nya konton börjar med sänkta kvoter för samtidighet och minne, som AWS höjer automatiskt utifrån användningen. Lösningen är att begära en höjning före lanseringen och att lasttesta vid den väntade toppbelastningen.

En funktion som startar sig själv får räkningen att skena. En funktion som anropar sig själv, eller som sätter i gång en process som anropar den igen, kan utlösa en flod av anrop, och vart och ett debiteras. AWS råd är att undvika rekursiva anrop och, om en oväntad volym plötsligt dyker upp, genast sätta funktionens reserved concurrency till `0`, vilket stoppar alla anrop tills koden är rättad.

Programmeringsmodellen binder dig till en plattform. Whitepapern pekar ut plattformens programmeringsmodell, dess gränssnitt för händelser och meddelanden och dess BaaS-erbjudanden som källor till inlåsning. Lösningen är att hålla affärslogiken i vanliga funktioner som handlern bara anropar, och att föredra plattformar som Container Apps, som kör en vanlig containeravbildning från valfritt register, när en senare flytt är en verklig möjlighet.

Läs mer

- CNCF Serverless Whitepaper v1.0 — arbetsgruppens definitioner av serverless, FaaS och BaaS, och nackdelarna.
- What is AWS Lambda? — vad Lambda tar hand om, dess funktionstyper och dess debiteringsmodell.
- Define Lambda Event Function handler in Python — hela kvittoexemplet, namngivning av handlers och god praxis för koden.
- Understanding the Lambda execution environment lifecycle — faserna Init, Invoke och Shutdown, kallstarter och återanvändning av miljöer.
- Lambda quotas — gränser för samtidighet, minne, tidsgräns och payload, och hur de höjs.
- Azure Container Apps overview — Microsofts serverless-plattform för containrar, som kan skala till noll.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är serverless?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/serverless/