

# Introduksjon til serverless

Henrik Wingerei, Fredrik Valdmanis, Harald Ringvold

# Agenda

## Del 1

- Introduksjon til serverless
- Hands-on
- Oppsummering

## Del 2

- Introduksjon til Serverless Framework
- Hands-on
- Oppsummering og avslutning

# Del 1: Introduksjon og case

# Serverless

- Tredjepart (skyleverandør) håndterer oppsett, provisjonering og forvaltning av servere
- All funksjonalitet kjøres på “managed services”, høynivåttjenester der underliggende infrastruktur er abstrahert bort
- Kan bruke enkelttjenester, eller sy de sammen til en fullstendig applikasjon
- Har i lang tid hatt managed services for database, serving av statiske filer, CDN, etc, men backend har typisk fortsatt kjørt på VM-er eller PaaS-løsninger

# Function as a service

- AWS Lambda og andre FaaS-varianter muliggjør nå full serverless computing
- Backendkode skrives som rene funksjoner
  - Input: Et event i skyplattformen
    - F.eks.: HTTP-request, ny fil lastet opp i S3, endring i databasetabell
  - Output: F.eks.: HTTP-response
- Funksjonens runtime starter/stopper automatisk for hver request
- Ingen egen provisjonering og drift av servere
- Faktureres for per kjøring av funksjonen

# Hvorfor serverless?

- Ingen håndtering av infrastruktur
- Automatisk skalering
- Individuell skalering av funksjoner
  - Billing by function
- Betal kun for bruk

Case



Lambda





Lambda



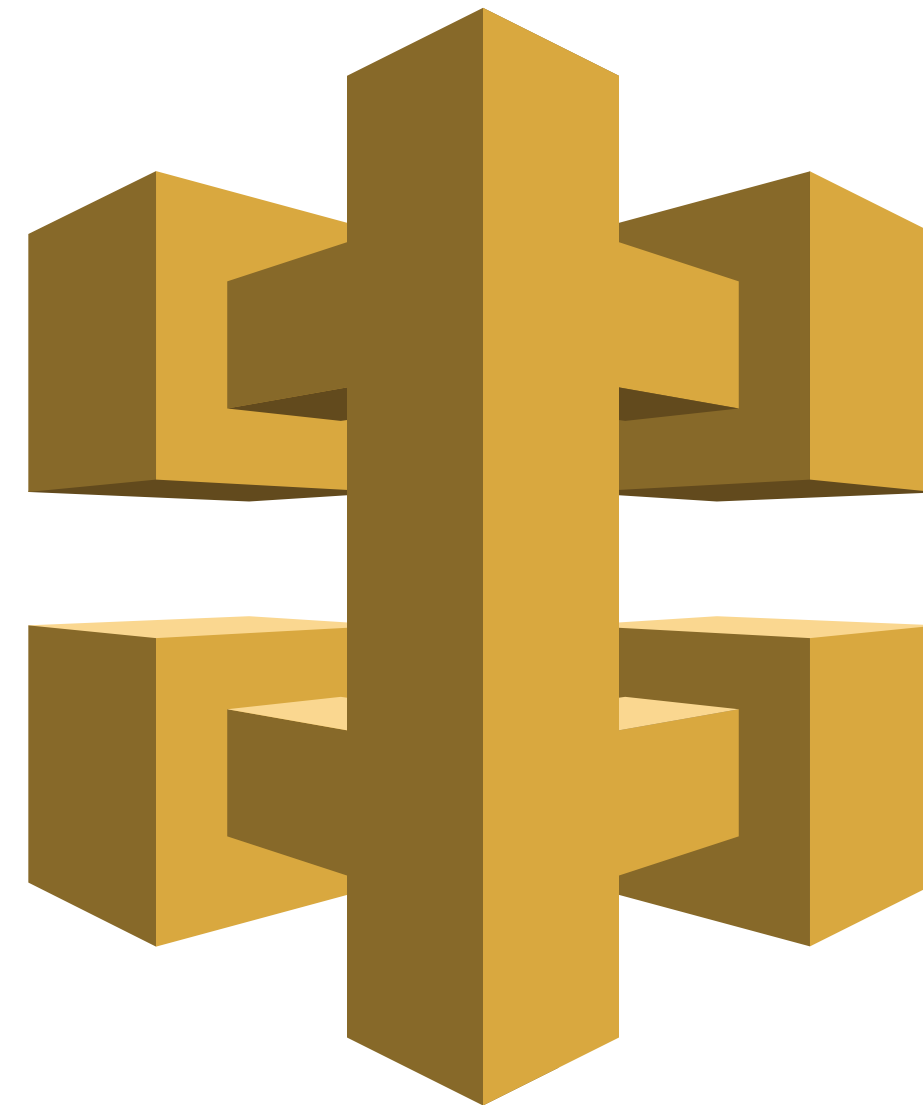
DynamoDB



Lambda



DynamoDB



API Gateway



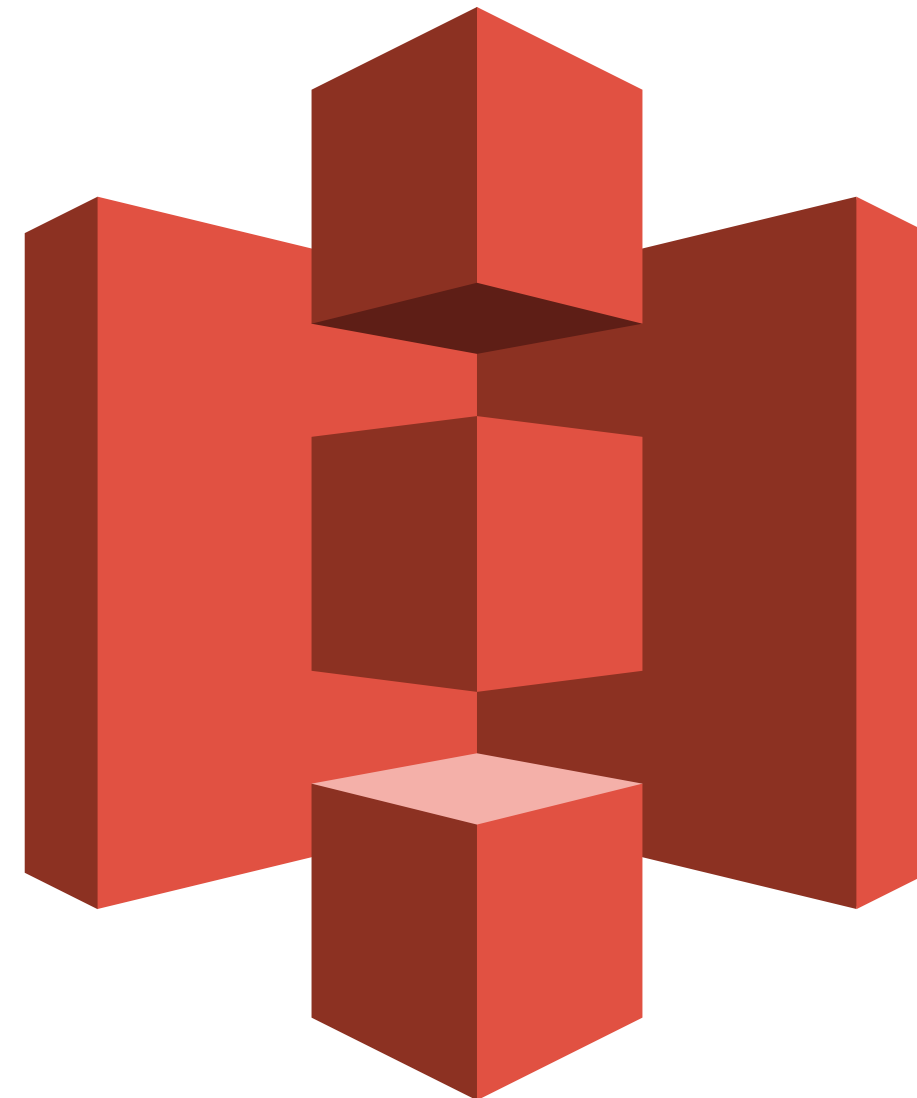
API Gateway



Lambda



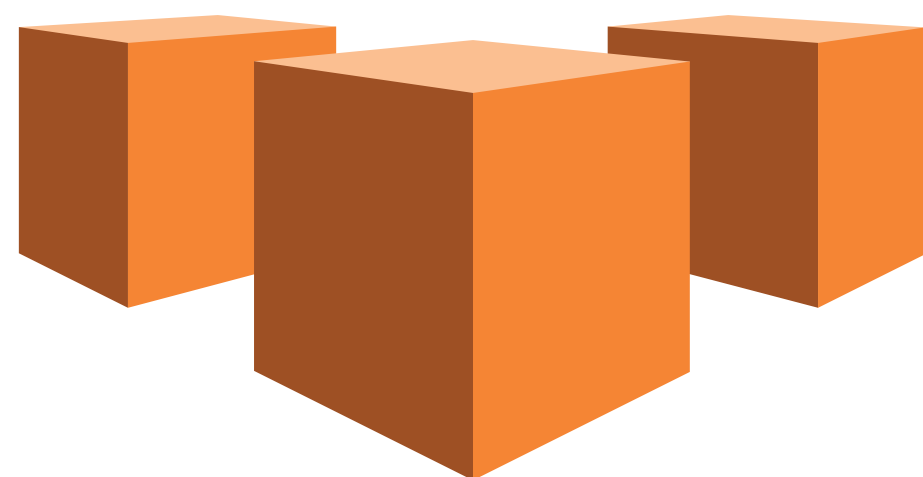
DynamoDB



S3



Cloudfront



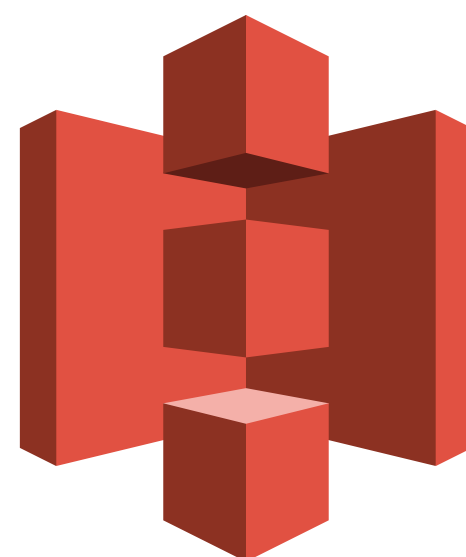
API Gateway



Lambda



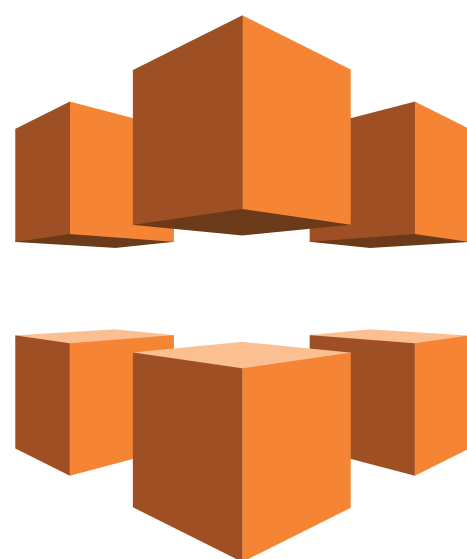
DynamoDB



S3



Klient



Cloudfront



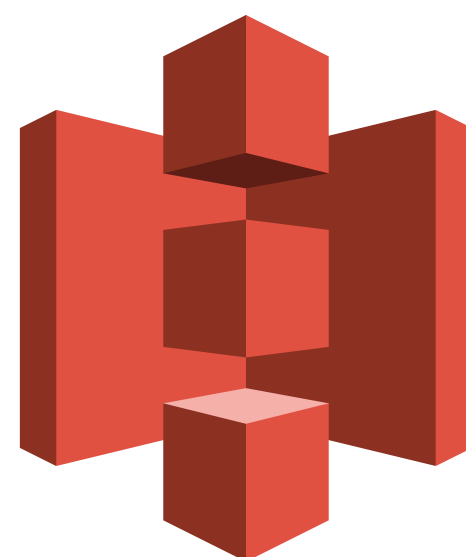
API Gateway



Lambda



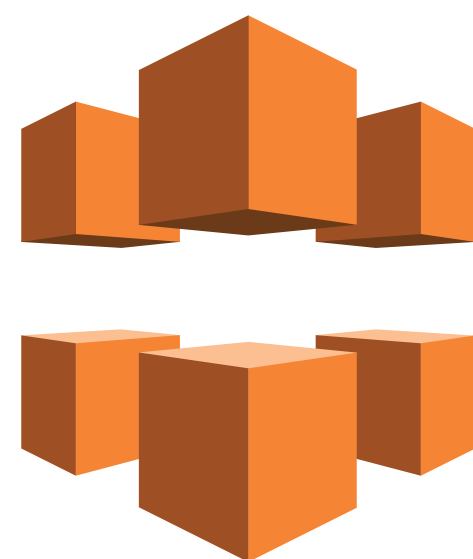
DynamoDB



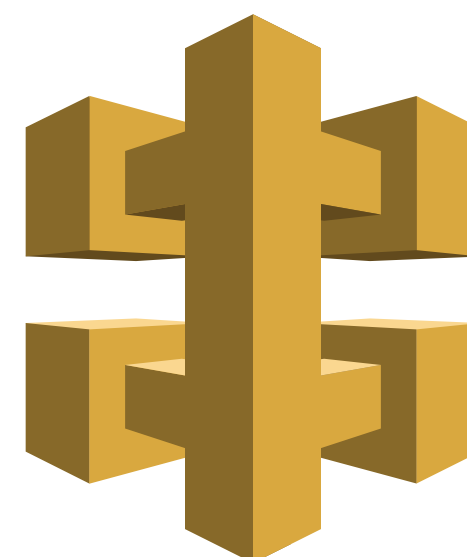
S3



Klient



Cloudfront



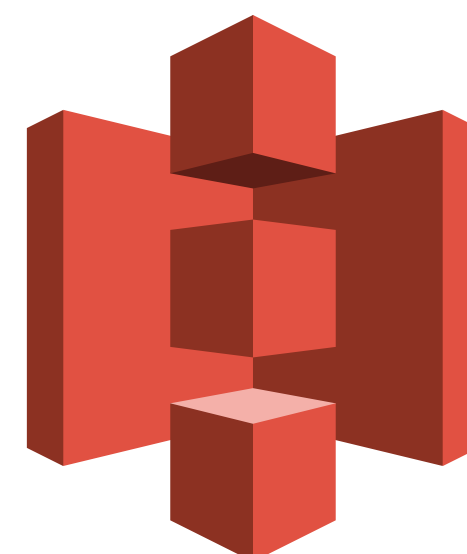
API Gateway



Lambda



DynamoDB



S3

<https://henriwi.gitbooks.io/serverless-workshop/>

<https://github.com/henriwi/serverless-workshop>

# DynamoDB

- “Fully managed” NoSQL-database (key/value store)
- Tabellene provisjonerer med en angitt “throughput”-kapasitet
- Ingen skikkelig autoscaling – løsninger finnes, men de må bygges over plattformen og er ikke-trivielle
- Ressurser:
- <https://www.trek10.com/blog/dynamodb-autoscaling-serverless-way/>



# Lambda

- Function-as-a-Service
- Betaler per request i enheter à 100 ms kjøretid
- Maks eksekveringstid: 300s
- Automatisk skalering
- JavaScript, Java, Python, C#, Go
- Kan trigges fra bl. a.
  - API Gateway
  - S3
  - DynamoDB
  - Cloudwatch
  - Kinesis

# API Gateway

- Tjeneste for å sette opp API-er mot andre tjenester hos AWS
- Er nødvendig for å få trigget Lambda-funksjoner over HTTP(S)
- Tar seg av autorisering, API-versjonering, etc
- Skalerer automatisk

# S3

- Amazons filagringstjeneste
- Hoster statiske filer
- I vår arkitektur server den frontenden, som kun består av statiske filer: HTML, CSS, JS

# Cloudfront

- Content delivery network (CDN)
- Kan brukes for å samle innhold fra ulike origins under samme URL
- Angir “behaviors” som ruter trafikk på ulike pather under domenet ditt til ulike origins
- Kan konfigurere egne caching-regler for hvert origin

# Oppsummering case

- Satt opp applikasjon med database
- Uten mye kunnskap om servere, nettverk ol.
- Skalerer bra ut av boksen\*
- Kort tid

# Utfordringer fra case

- Manuelt oppsett
- Redigert kode i nettleseren
- AWS administrasjonspanel ikke brukervennlig
- Kan ikke teste lokalt

# Del 2: “Avanserte løsninger”

# Verktøy og automatisering

- Terraform
- AWS CLI
- Lambda-local
- Byggverktøy (Gulp)
- CloudFormation



# Serverless framework

- Node.js CLI verktøy
- Konfigurerer opp applikasjonen i yaml
- SF laster opp konfigurasjonen til CloudFormation og provisjonerer opp alle AWS-tjenestene

# Serverless framework

- Services
  - En applikasjon/tjeneste
- Functions
  - Lambdafunksjoner
- Events
  - Ulike eventer som trigger en funksjon
  - API Gateway, S3, DynamoDB, Kineses++
- Resources
  - Ressurser som servicen din er avhengig av som f.eks. En DynamoDB-tabell, S3-bucket el.l.

# Serverless framework

- Deployer hele servicen inkl. funksjoner og alle ressurser
- Kan deploye kun lambda-funksjonene og invokere disse
  - Kan også invokere lambdafunksjonene lokalt i et “mock” AWS-miljø
- Lese logger, hente ut metrikker ++
- Støtte for AWS, Azure og OpenWhisk (IBM)

Case – Serverless framework

Avslutning

# Oppsummering

- Full-stack applikasjon med managed services og Lambda på AWS
- Automatisert oppsett og utrulling med Serverless Framework og CloudFormation

# Alternativer til Serverless framework

- Apex (<http://apex.run/>)
- Zappa, for Python på AWS (<https://www.zappa.io/>)
- Claudia.js (<https://claudiajs.com/>)

# Fordeler med serverless og FaaS

- Ingen håndtering av infrastruktur
- Automatisk skalering
- Individuell skalering av funksjoner
  - Billing by function
- Betal kun for bruk



# Utfordringer

## Nytt computing-paradigme

- Kompetanse, verktøy, dokumentasjon
- Test, deploy, konfigurasjon, miljøer, versjonering

# Utfordringer

Vendor-spesifikk kunnskap (gjelder alt av cloud, men spesielt her)

# Utfordringer

Vanskelig å sammenligne pris vs. tradisjonell “serverfull” computing

# Utfordringer

Black box – begrenset innsikt i hvordan plattformen fungerer

- Ingen konfigurasjon og tuning av runtime

# Utfordringer

Utforende å feilsøke

# Utfordringer

Lang oppstartstid på “kalde” lambdaer

# Spådom

Serverless blir den dominerende modellen i  
framtiden

# Spådom

Bør vurdere å ta i bruk på områder der det passer spesielt godt

- Batcher
- Beregninger
- Funksjonalitet som brukes periodisk eller sjeldent
- Støttefunksjonalitet (e-post-utsending, logging)
- Hendelsesdrevne systemer (Kinesis)



# Spådom

På tradisjonelle webapper (CRUD) går kanskje ikke kost-nytte-regnskapet opp helt ennå

- Større kostnader pga kompetanse, plattformenes modenhet, etc
- Mindre nytteverdi for “gjennomsnittlige” apper med jevn trafikk, mindre behov for individuell skalering, etc.