

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

Del 3

- Bonusoppgaver
- Oppsummering og avslutning

Del 1: Introduksjon og case

Serverless

- Tredjepart (skyleverandør) håndterer oppsett, provisjonering og forvaltning av servere
- All funksjonalitet kjøres på “managed services”, høynivå tjenester der underliggende infrastruktur er abstrahert bort

Serverless

- 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

Function as a service

- 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



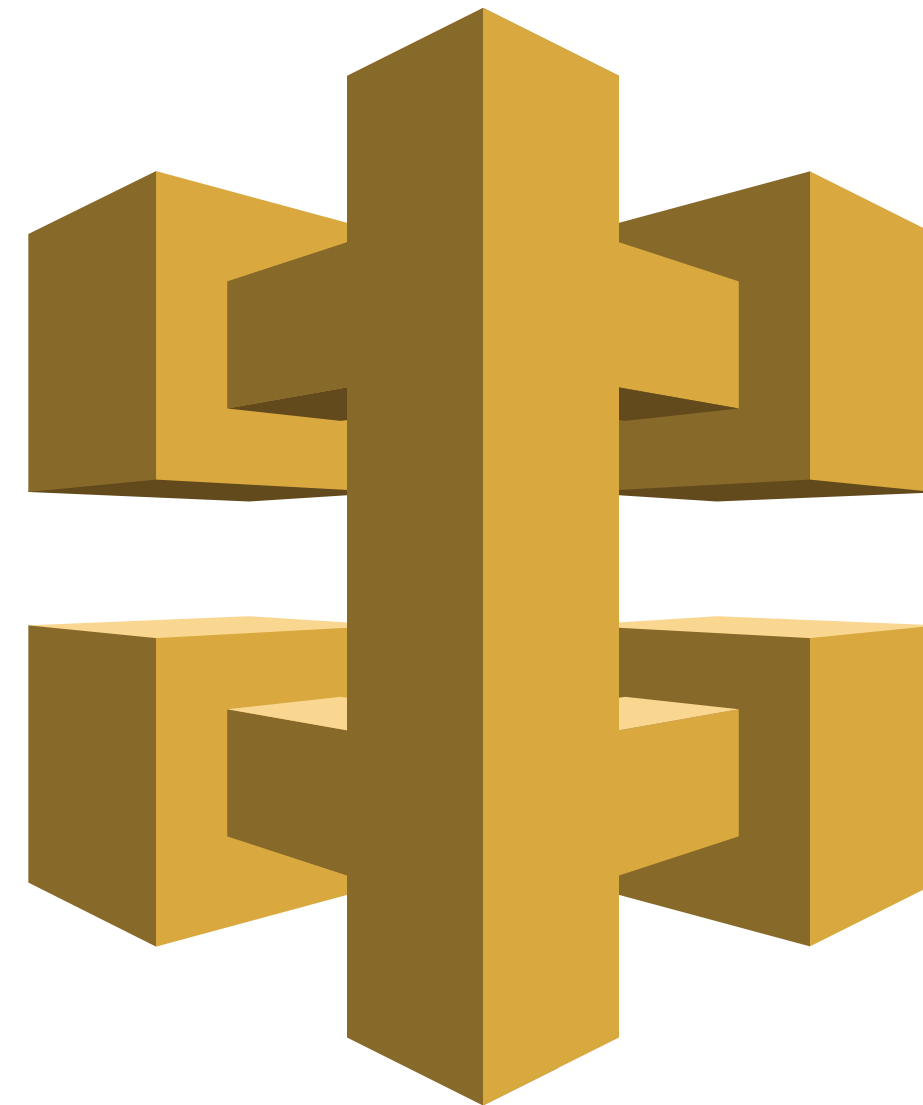
DynamoDB



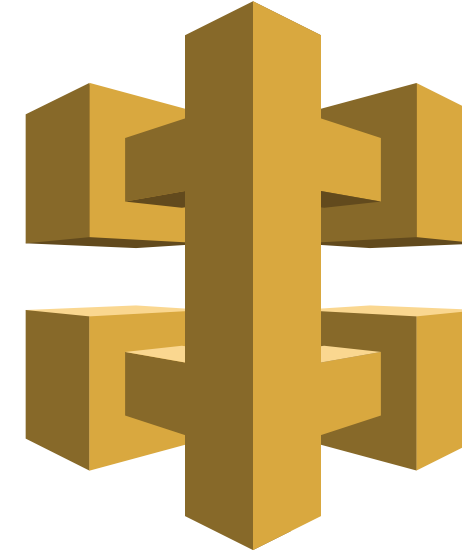
Lambda



DynamoDB



API Gateway



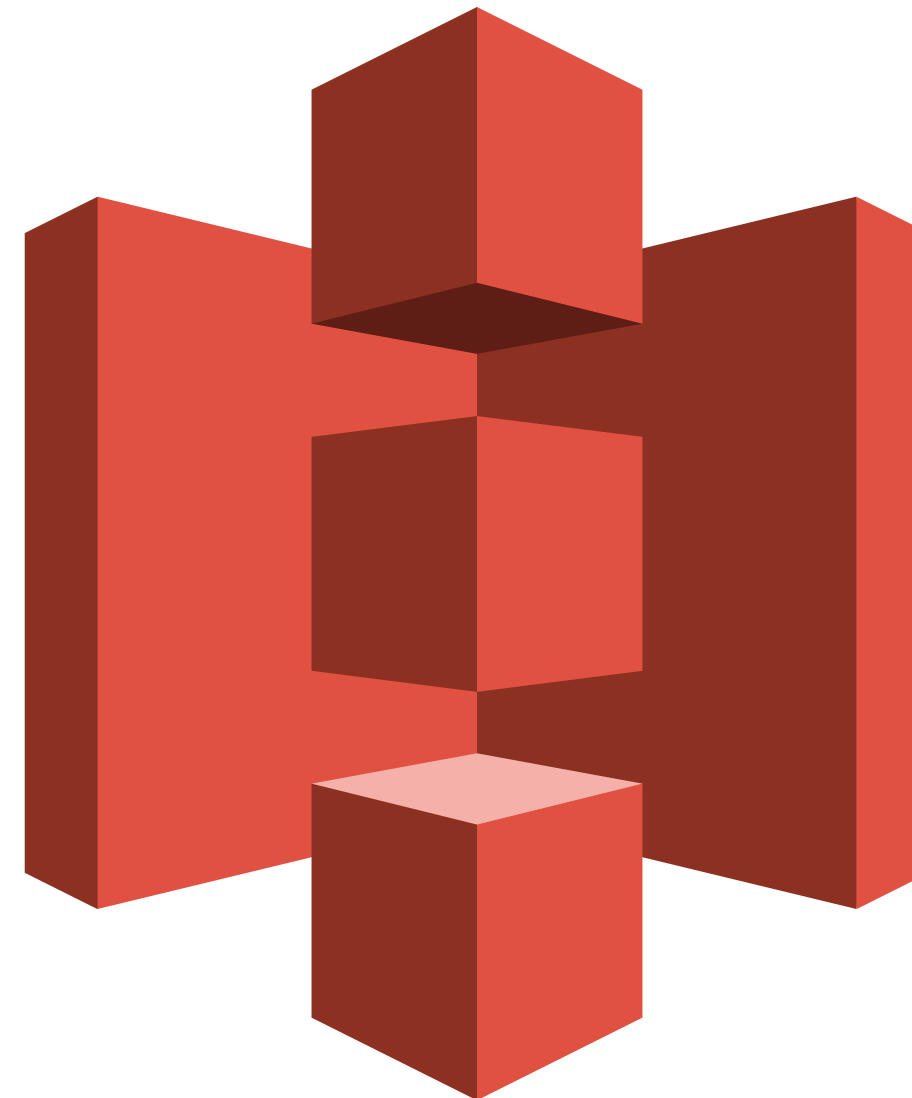
API Gateway



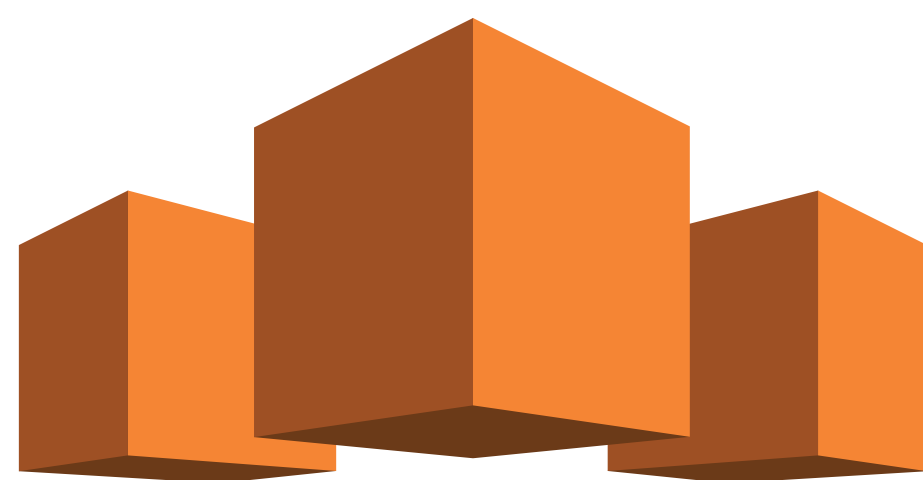
Lambda



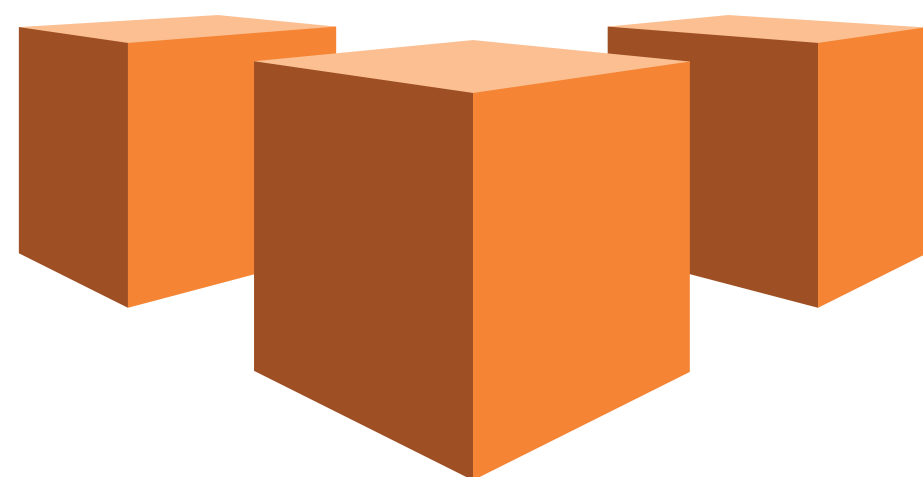
DynamoDB



S3



Cloudfront



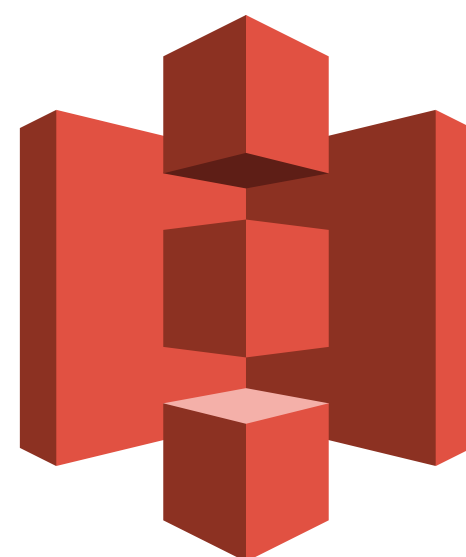
API Gateway



Lambda



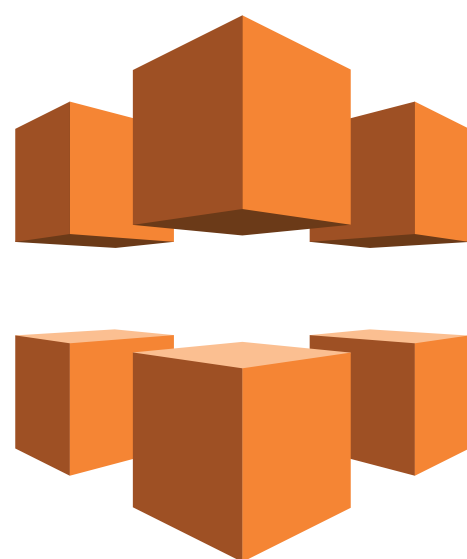
DynamoDB



S3



Klient



Cloudfront



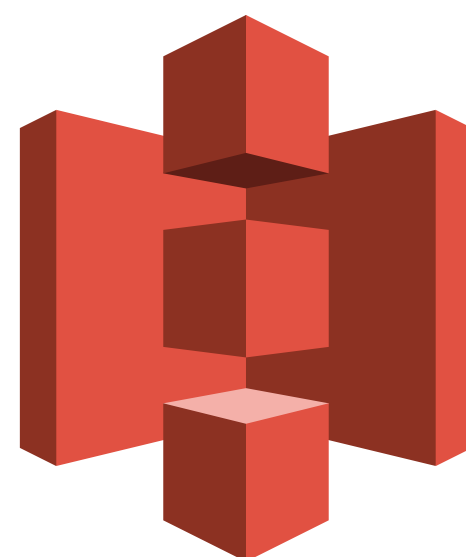
API Gateway



Lambda



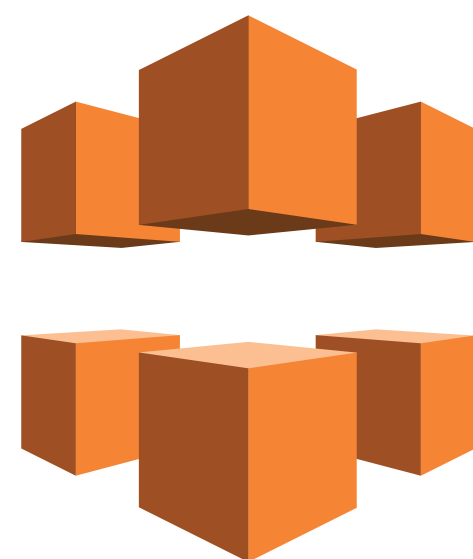
DynamoDB



S3



Klient



Cloudfront



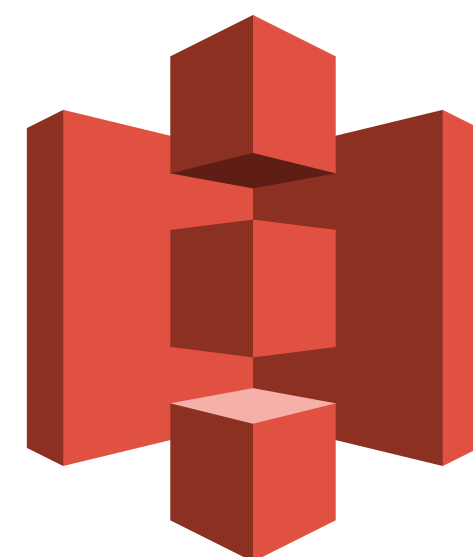
API Gateway



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

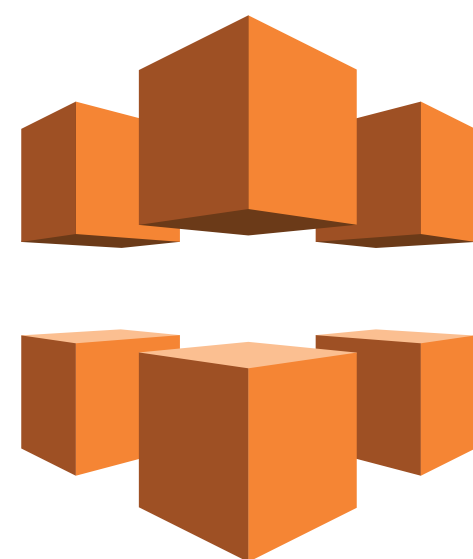
- Amazons filagringstjeneste
- Hoster statiske filer
- I vår arkitektur server den frontenden, som kun består av statiske filer: HTML, CSS, JS

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



Klient



Cloudfront



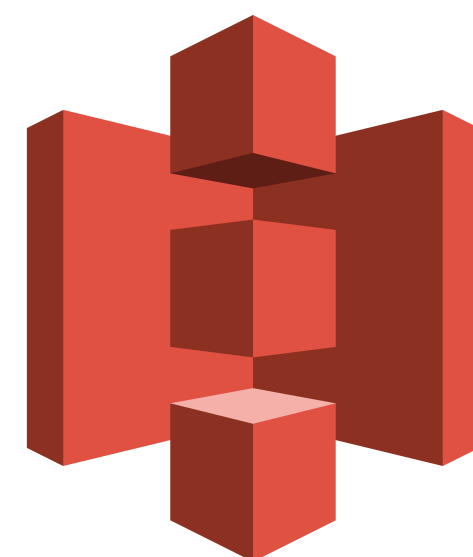
API Gateway



Lambda



DynamoDB



S3

Utfordringer fra case

- Manuelt oppsett
- Redigert kode i nettleseren
- AWS webkonsollet er lite brukervennlig
- Kan ikke teste lokalt

Del 2: Automatisert oppsett

Serverless framework

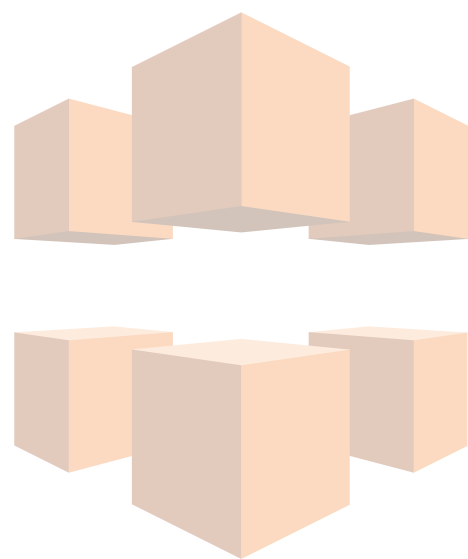
- Node.js CLI verktøy
- Et verktøy for å utvikle, teste og deploye serverless-applikasjoner
- Støtter flere skyleverandører

Serverless framework

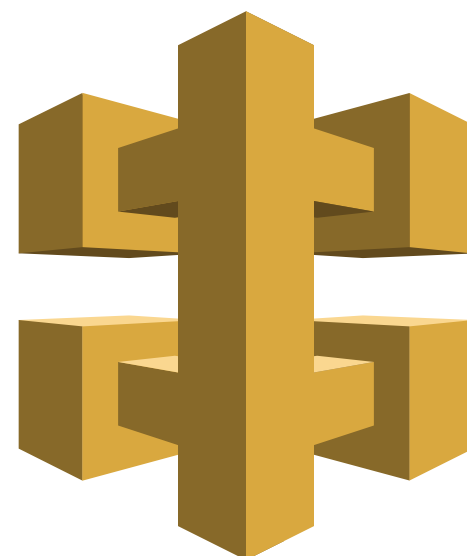
- Konfigurerer applikasjonen i yaml
- Serverless Framework (SF) bruker CloudFormation til å provisjonere opp alle tjenestene som applikasjonen din bruker



Klient



Cloudfront



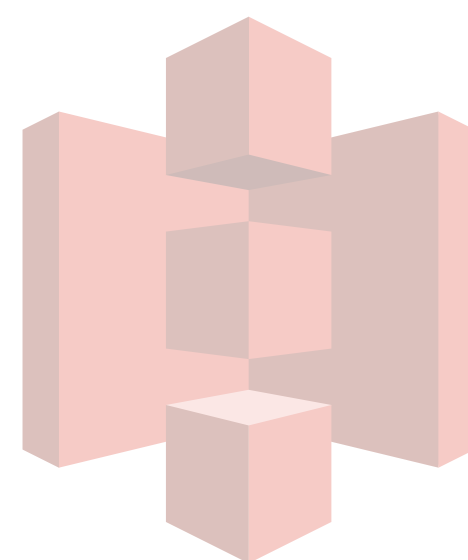
API Gateway



Lambda



DynamoDB

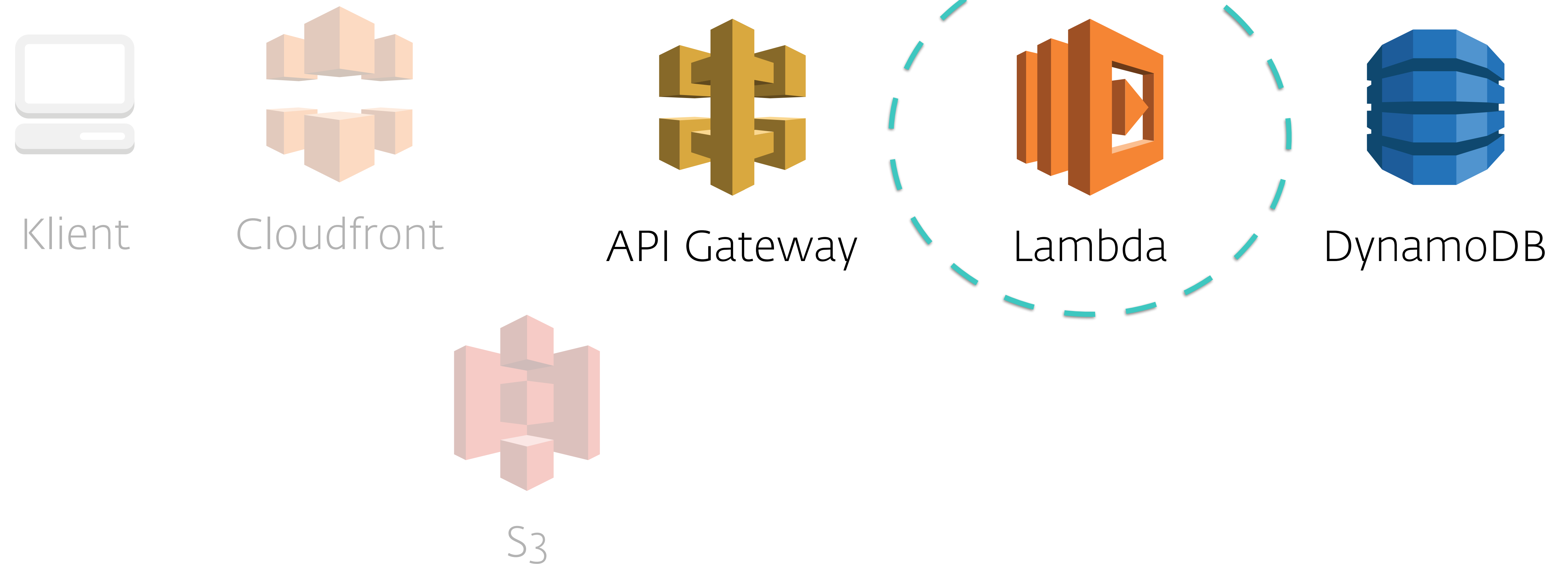


S3

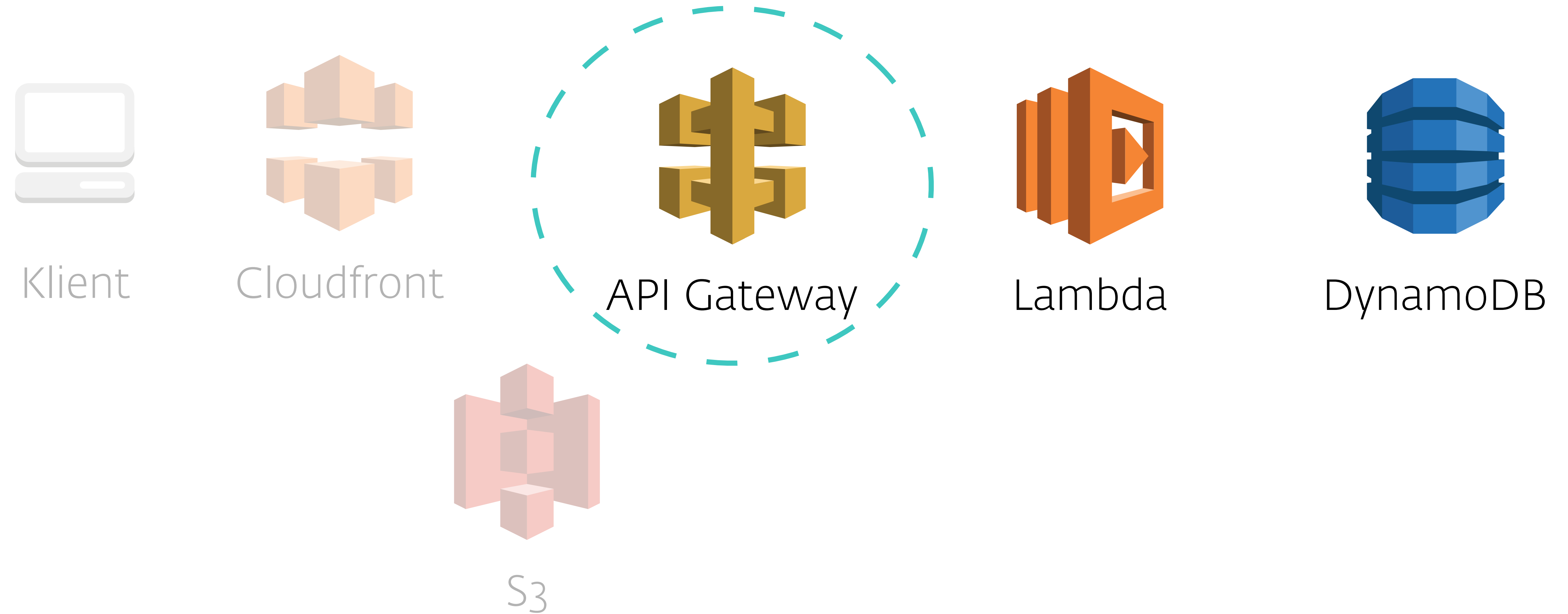
Service



Function



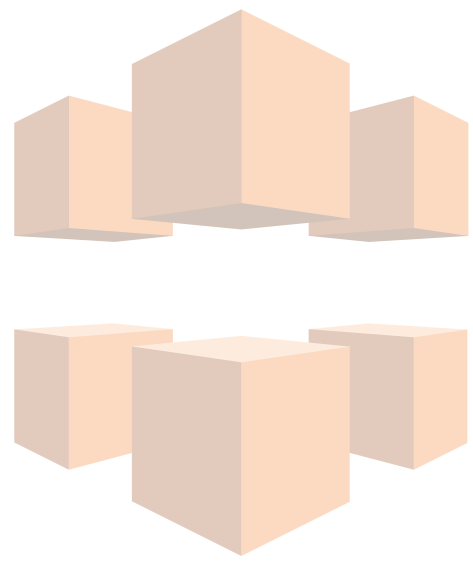
Events



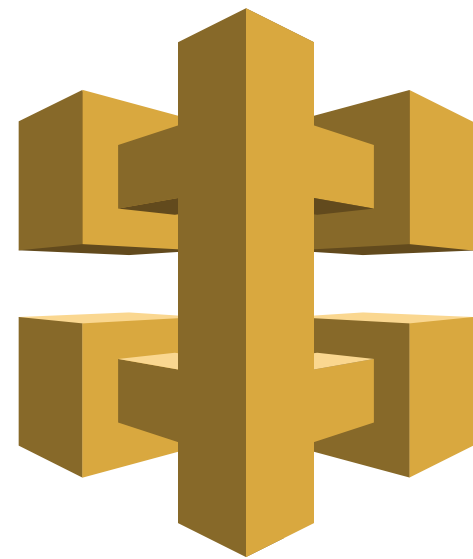
Resources



Klient



Cloudfront



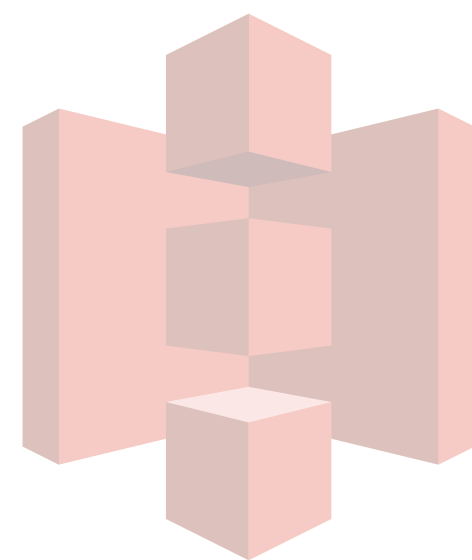
API Gateway



Lambda

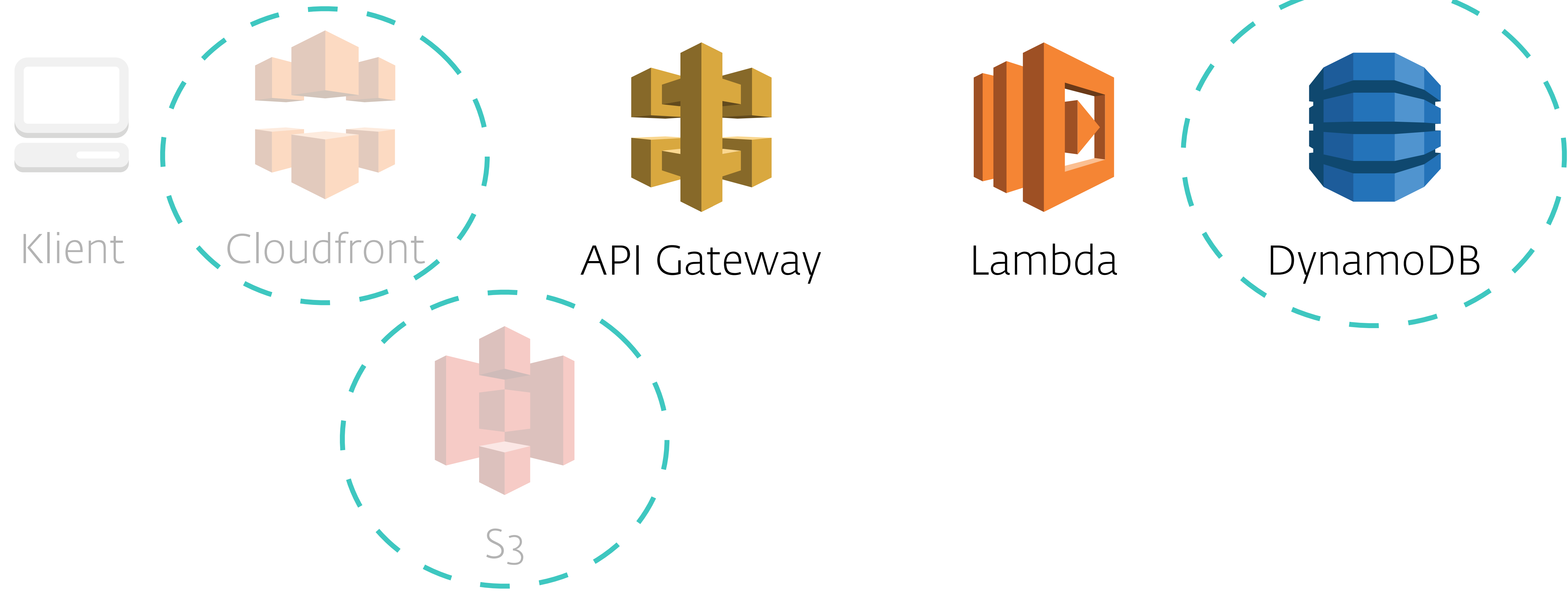


DynamoDB



S3

Resources



Serverless framework

- Deployer hele servicen inkl. funksjoner og alle ressurser
- Kan deploye kun lambda-funksjonene og invokere disse
- Mulighet for å invokere lambda-funksjonene lokalt
- Støtte for å lese logger, hente ut metrikker ++

Case – Serverless framework

Oppsummering Serverless framework

Del 3: Bonusoppgaver

Avslutning

Alternativer for automatisert oppsett

- AWS CLI
- SAM (AWS Serverless Application Model)
- Terraform
- Cloudformation
- 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.

Takk for deltakelsen,
og takk for oss!

Henrik Wingerei
Fredrik V. Mørken
Harald Ringvold