# AWS Architecture concept

## Well Architected Framework 6 Pillars

They are not something to balance, or trade-offs, they’re a synergy

* Operational Excellence
* Security
* Reliability
* Performance Efficiency
* Cost Optimization
* Sustainability

## AWS Well-Architected Tool

* Free tool to review your architectures against the 6 pillars Well-Architected

Framework and adopt architectural best practices

* How does it work?
  + Select your workload and answer questions
  + Review your answers against the 6 pillars
  + Obtain advice: get videos and documentations, generate a report, see the results in a dashboard
* Let’s have a look: <https://console.aws.amazon.com/wellarchitected>

We’ve explored the most important architectural patterns:

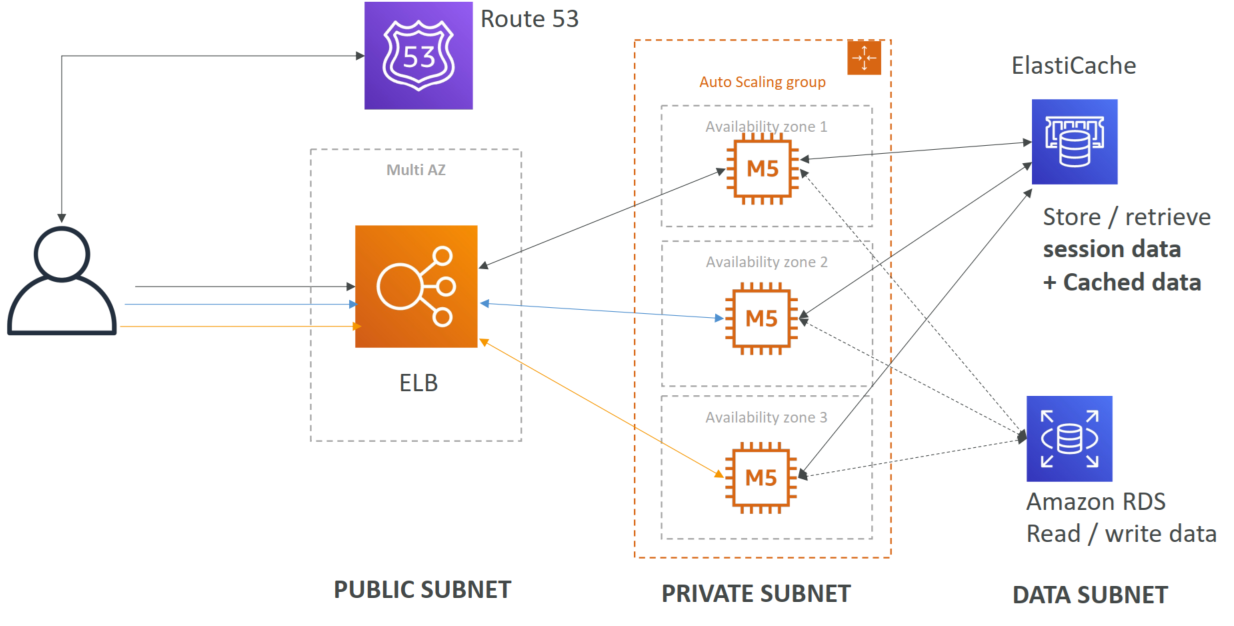
* Classic: EC2, ELB, RDS, ElastiCache, etc…
* Serverless: S3, Lambda, DynamoDB, CloudFront, API Gateway, etc…

See more AWS architectures:

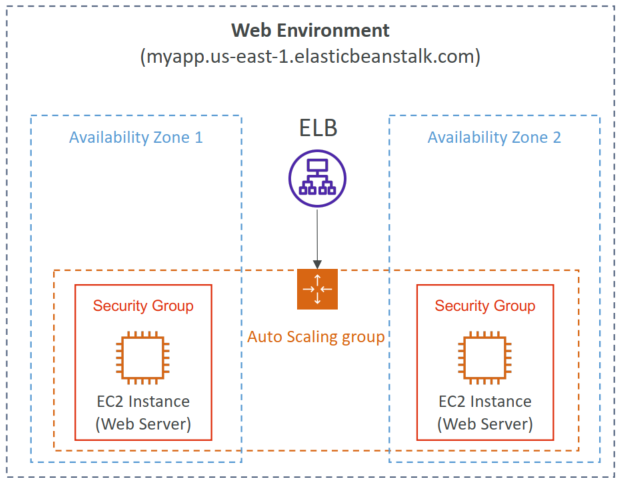
* <https://aws.amazon.com/architecture/>
* https://aws.amazon.com/solutions/

# AWS Architechture Example

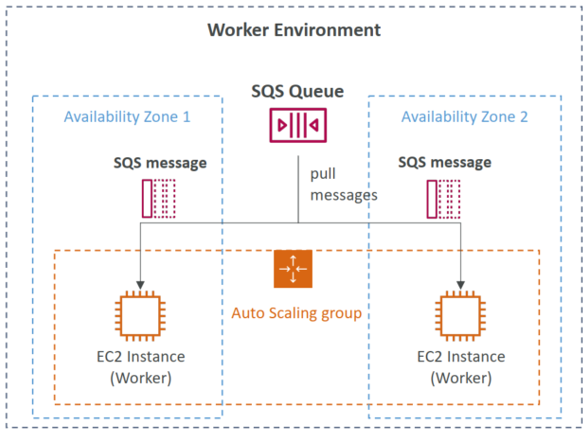
## Typical architecture: Web App 3-tier



### Web Server Tier



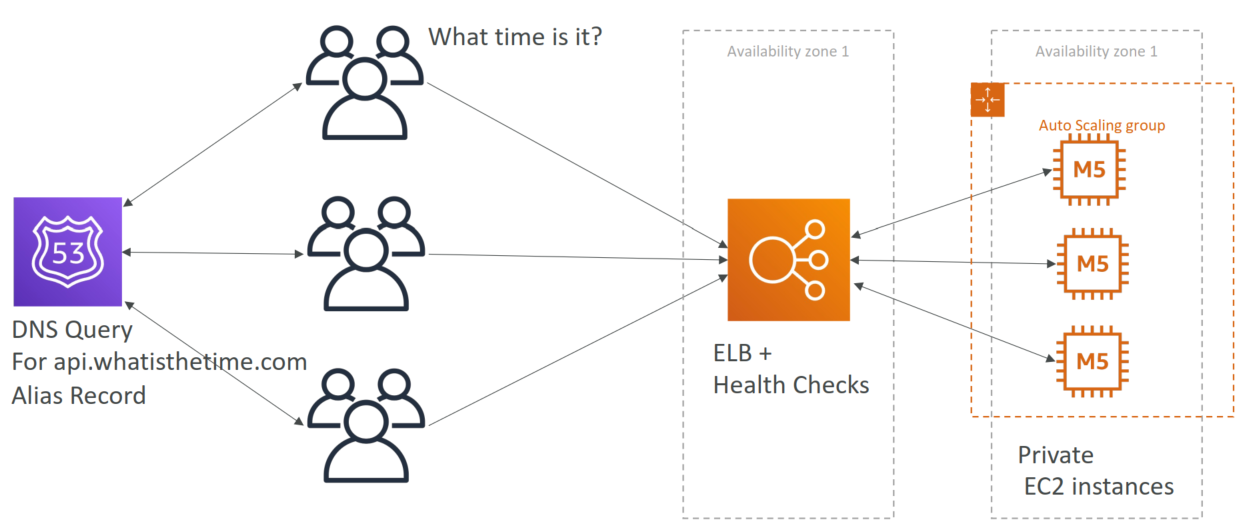
### Worker Tier



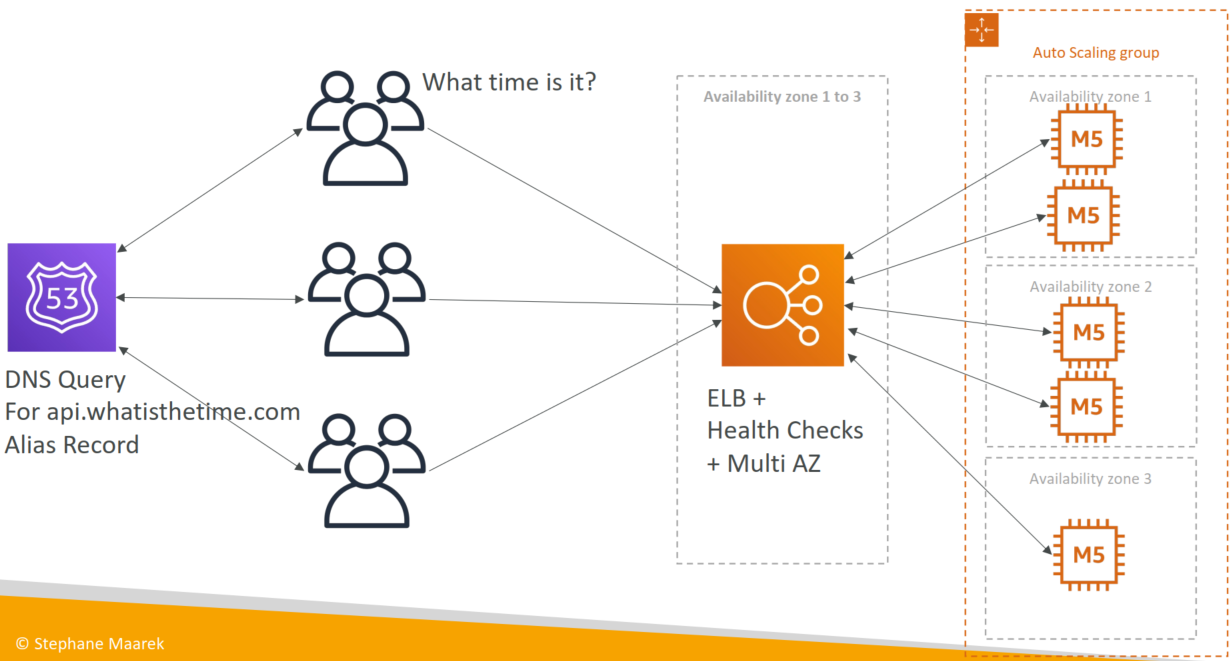
## Stateless web app: What time is it?

* WhatIsTheTime.com allows people to know what time it is
* We don’t need a database
* We want to start small and can accept downtime
* We want to fully scale vertically and horizontally, no downtime

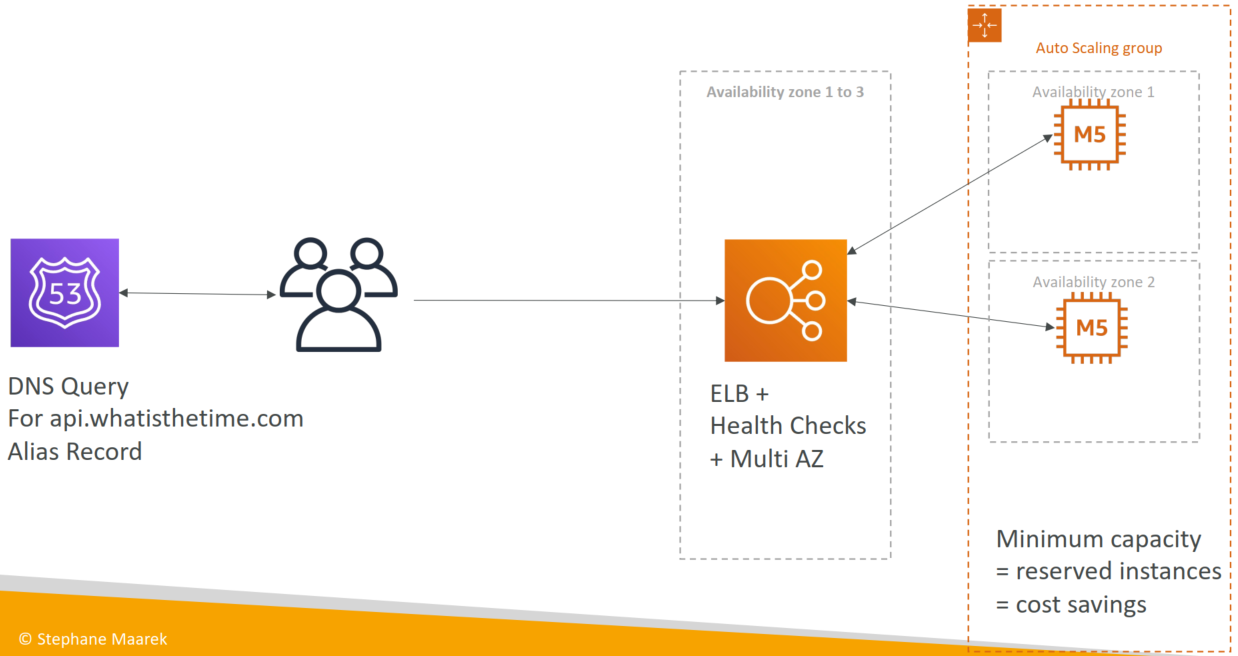
### Scaling horizontally, with an auto-scaling group



### Making our app multi-AZ



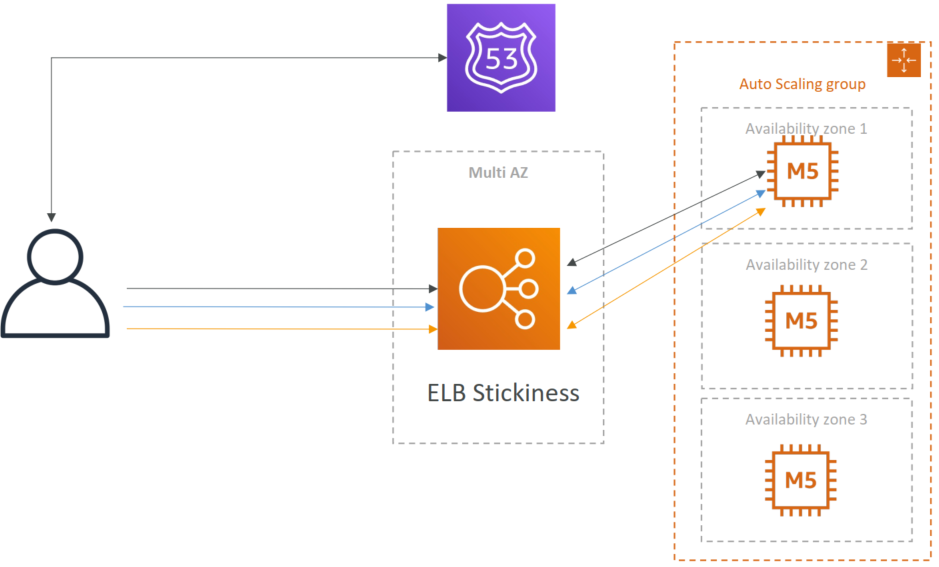
### Minimum 2 AZ => Let’s reserve capacity



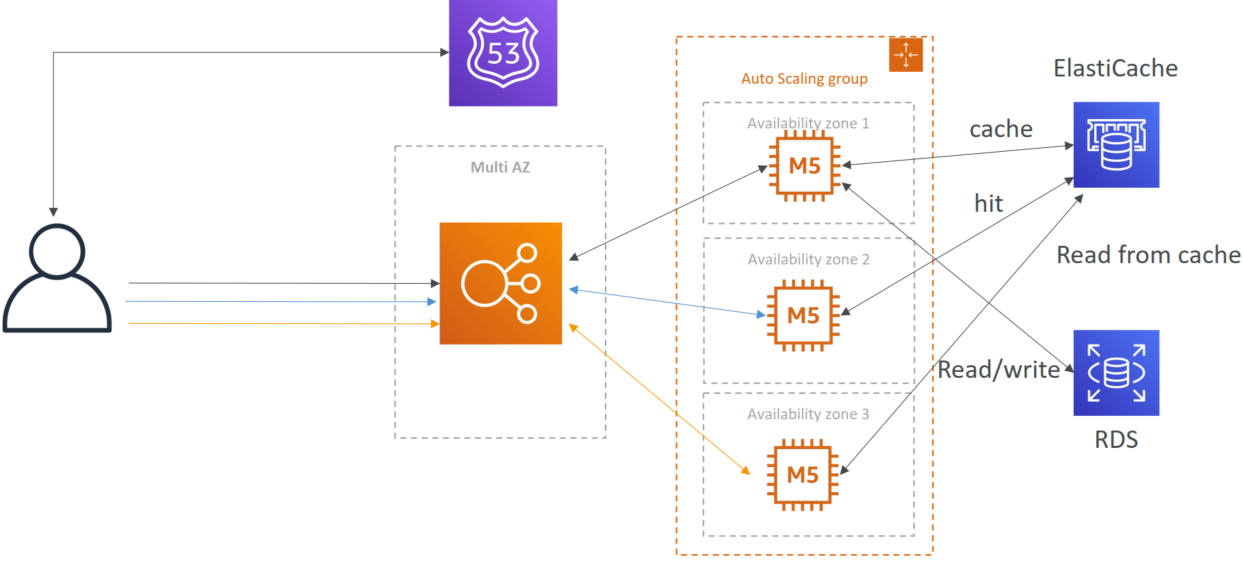
## Stateful Web App: MyClothes.com

* MyClothes.com allows people to buy clothes online.
* There’s a shopping cart
* Our website is having hundreds of users at the same time
* We need to scale, maintain horizontal scalability and keep our web application as stateless as possible
* Users should not lose their shopping cart
* Users should have their details (address, etc) in a database

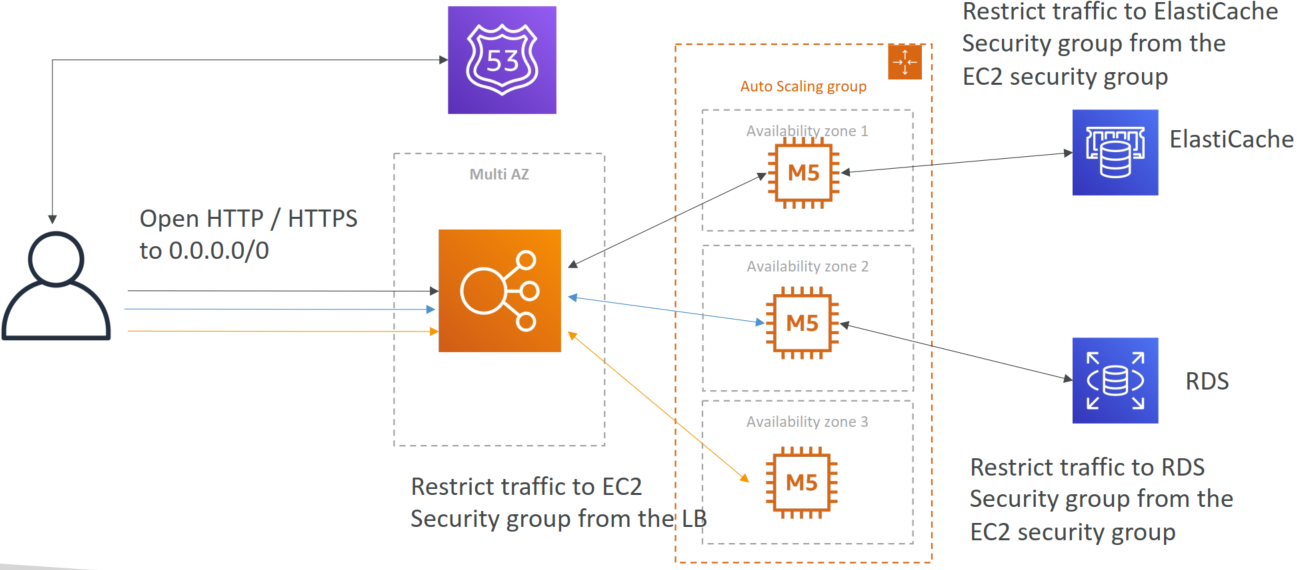
### Introduce Stickiness (Session Affinity)



### Scaling Reads (Alternative) – Lazy Loading



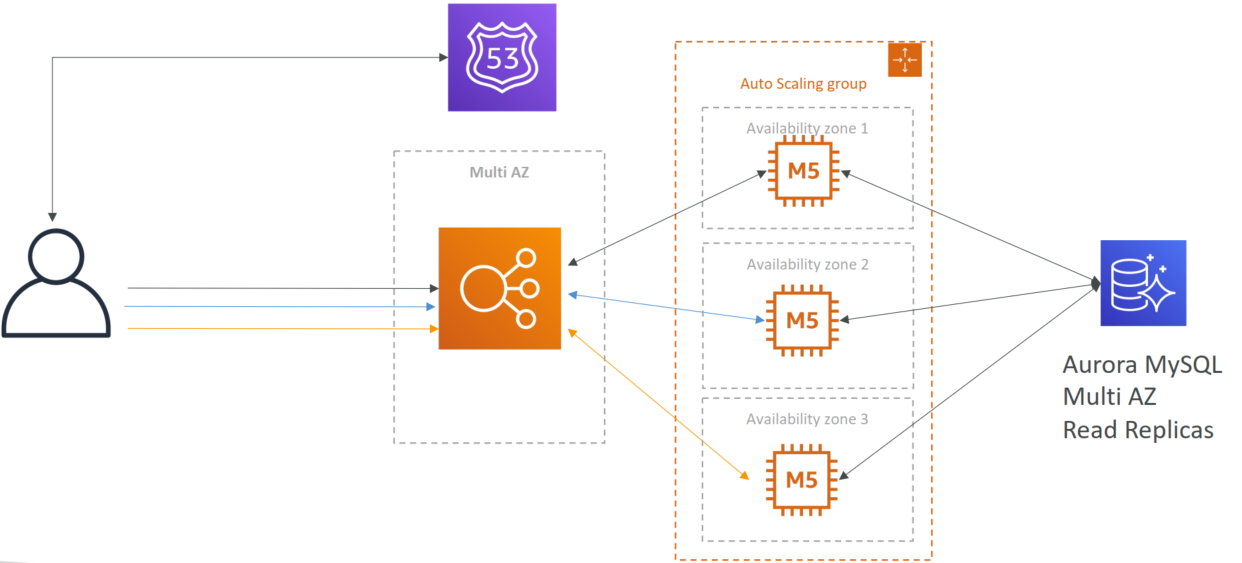
### Security Groups



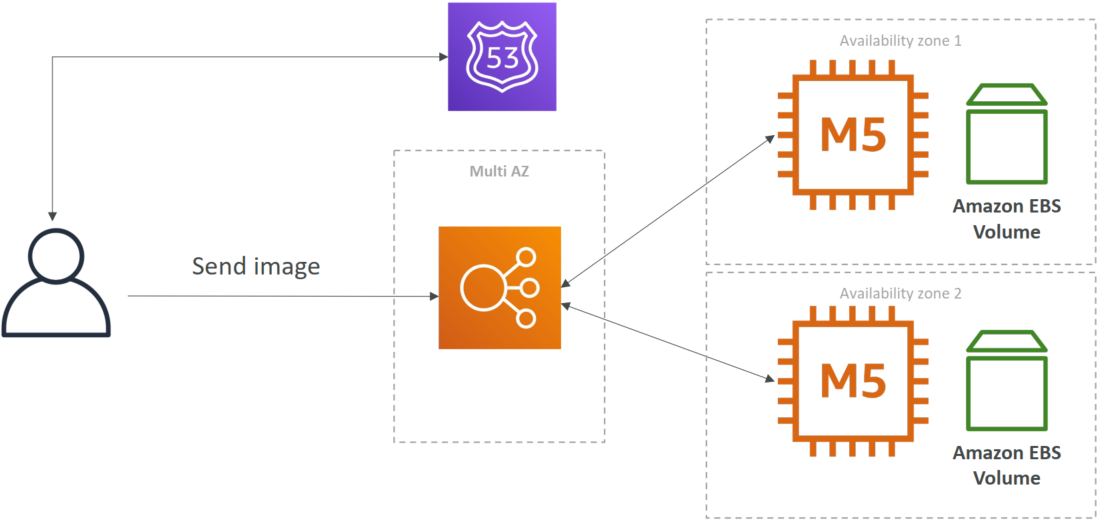
## Stateful Web App: MyWordPress.com

* + We are trying to create a fully scalable WordPress website
  + We want that website to access and correctly display picture uploads
  + Our user data, and the blog content should be stored in a MySQL database.

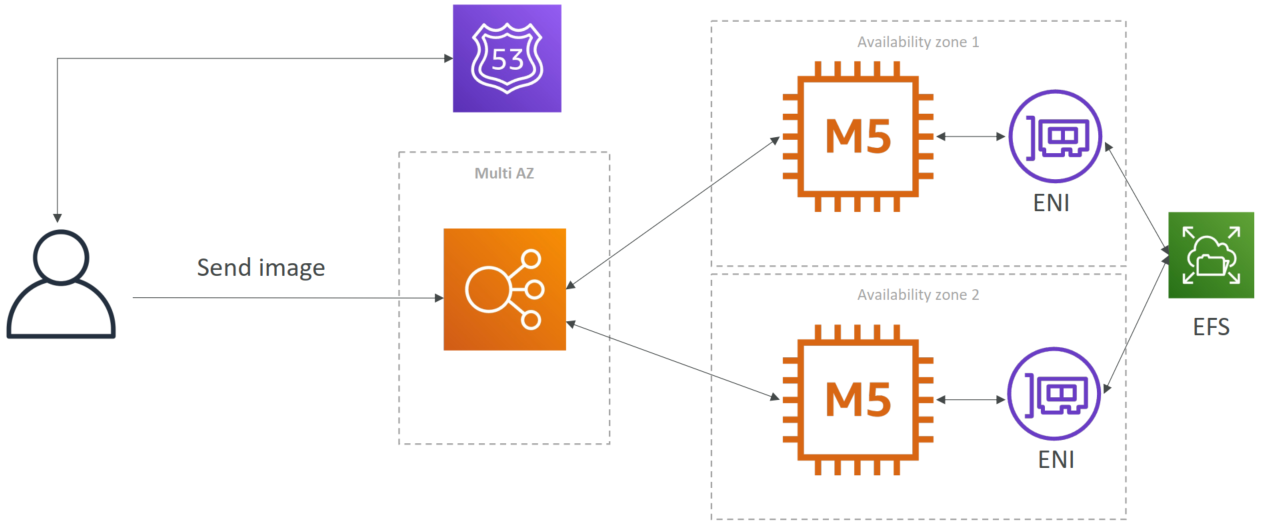
### Scaling with Aurora: Multi AZ & Read Replicas



### Storing images in EBS (single instance application)

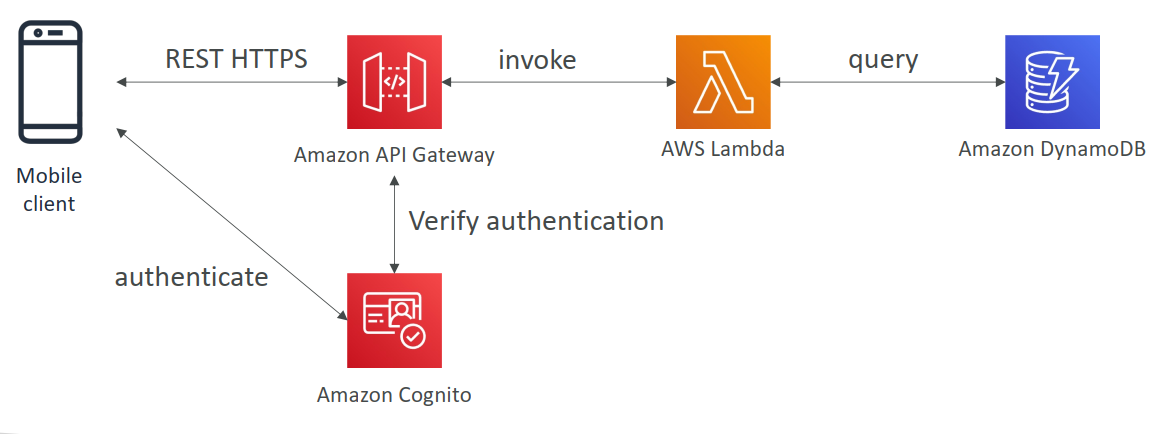


### Storing images in EBS (distributed application)

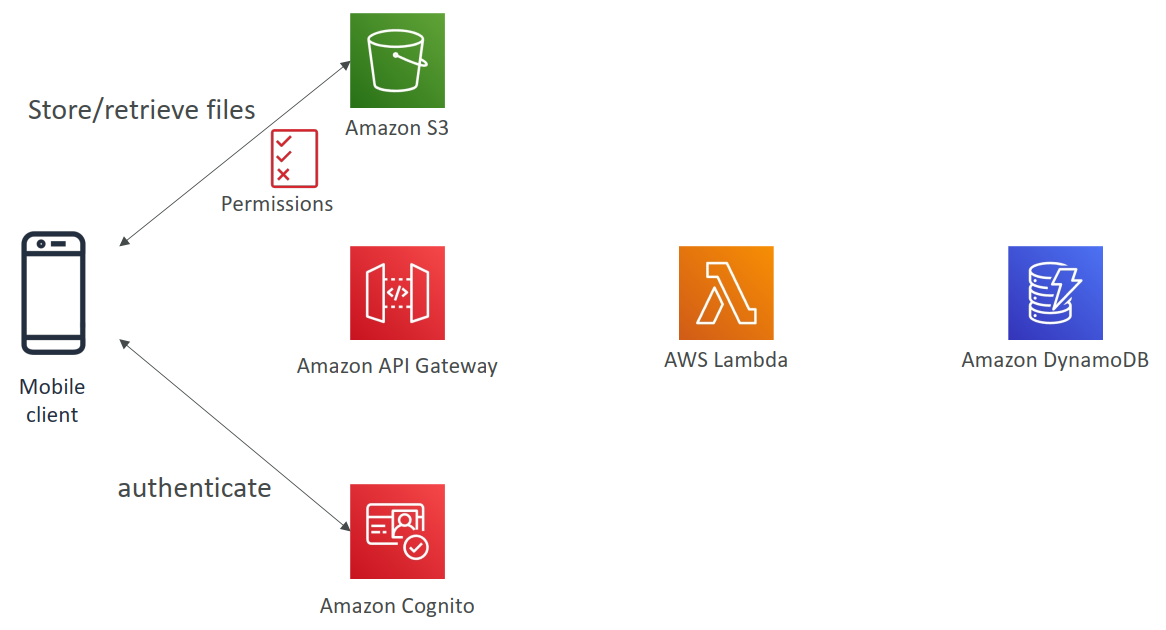


## Mobile app: Server less

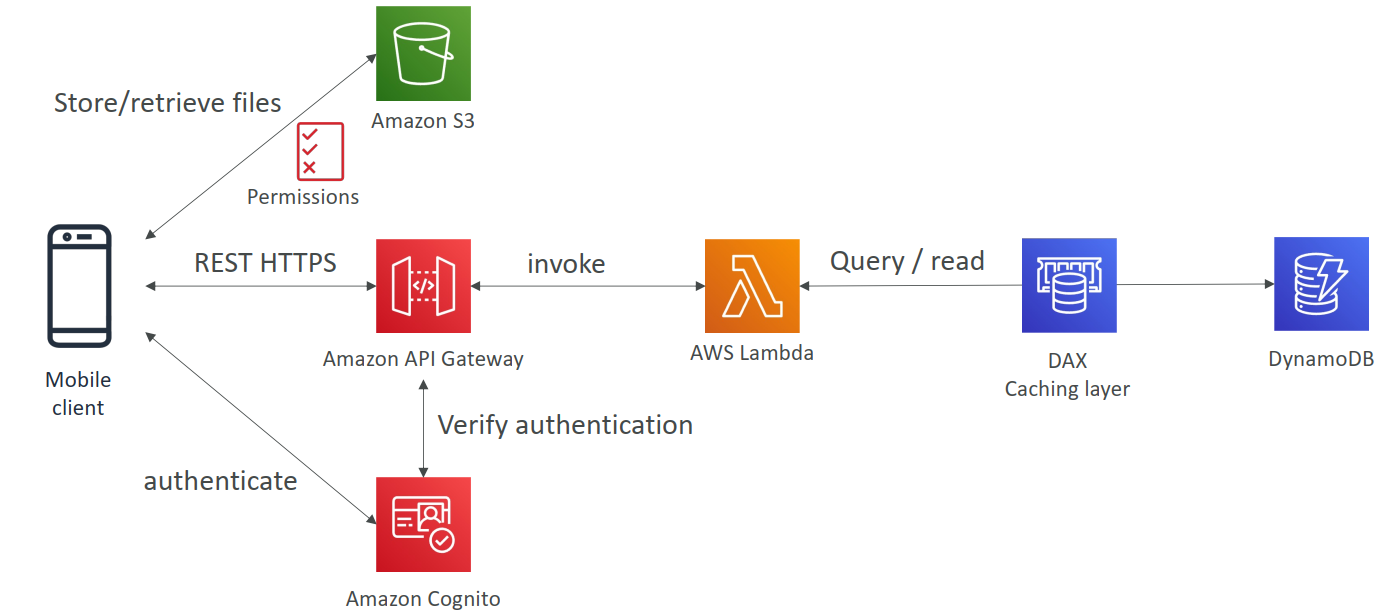
### Mobile app: REST API layer



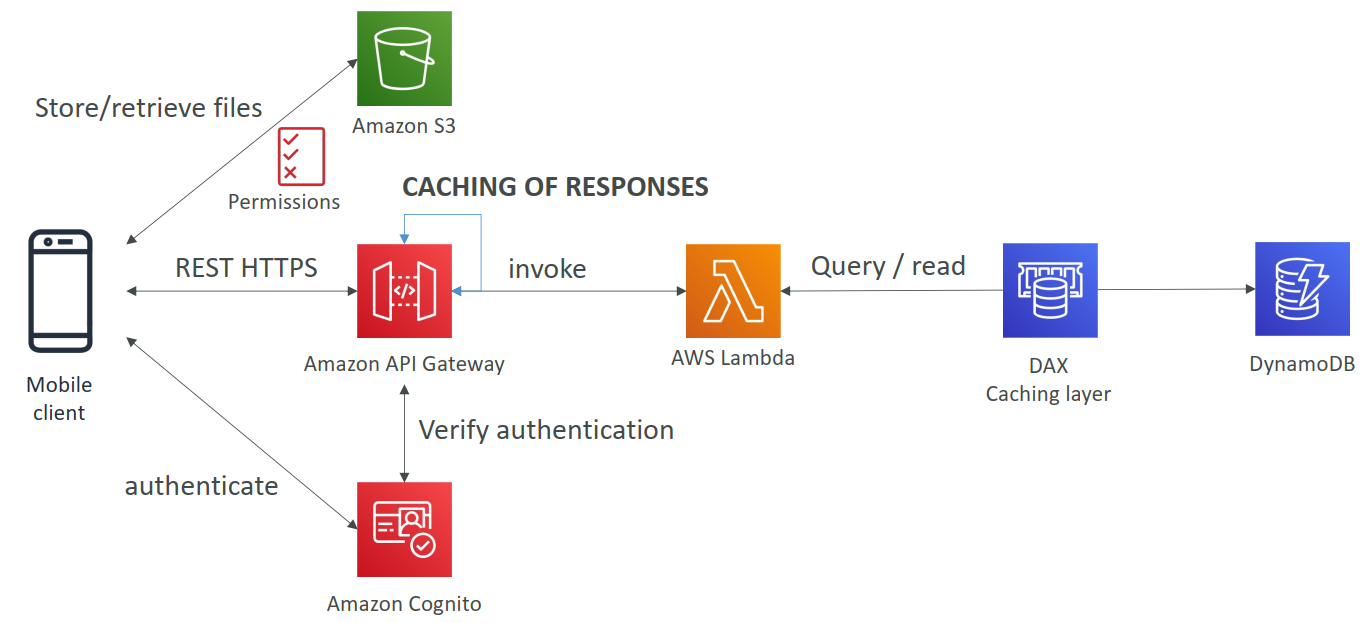
### Mobile app: Giving users access to S3



### Mobile app: High read throughput, static data



### Mobile app: Caching at the API Gateway



## AWS Hosted Website Summary

• We’ve seen static content being distributed using CloudFront with S3

• The REST API was serverless, didn’t need Cognito because public

• We leveraged a Global DynamoDB table to serve the data globally

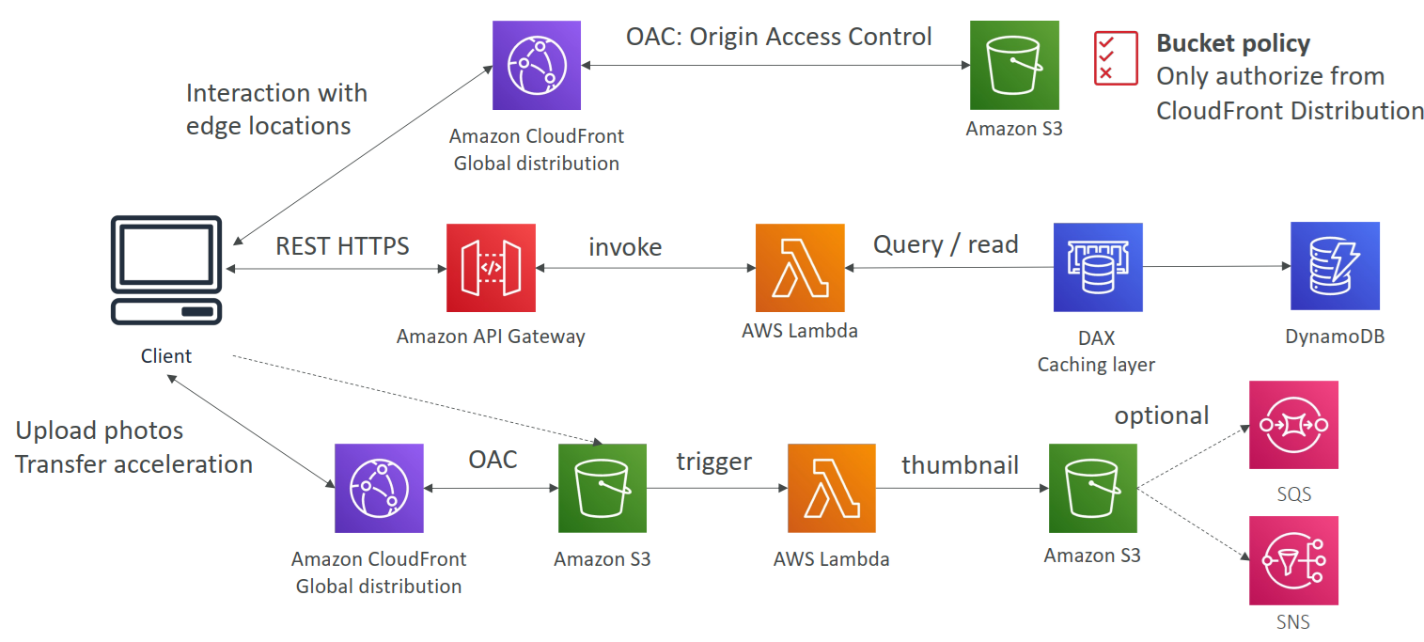
• (we could have used Aurora Global Database)

• We enabled DynamoDB streams to trigger a Lambda function

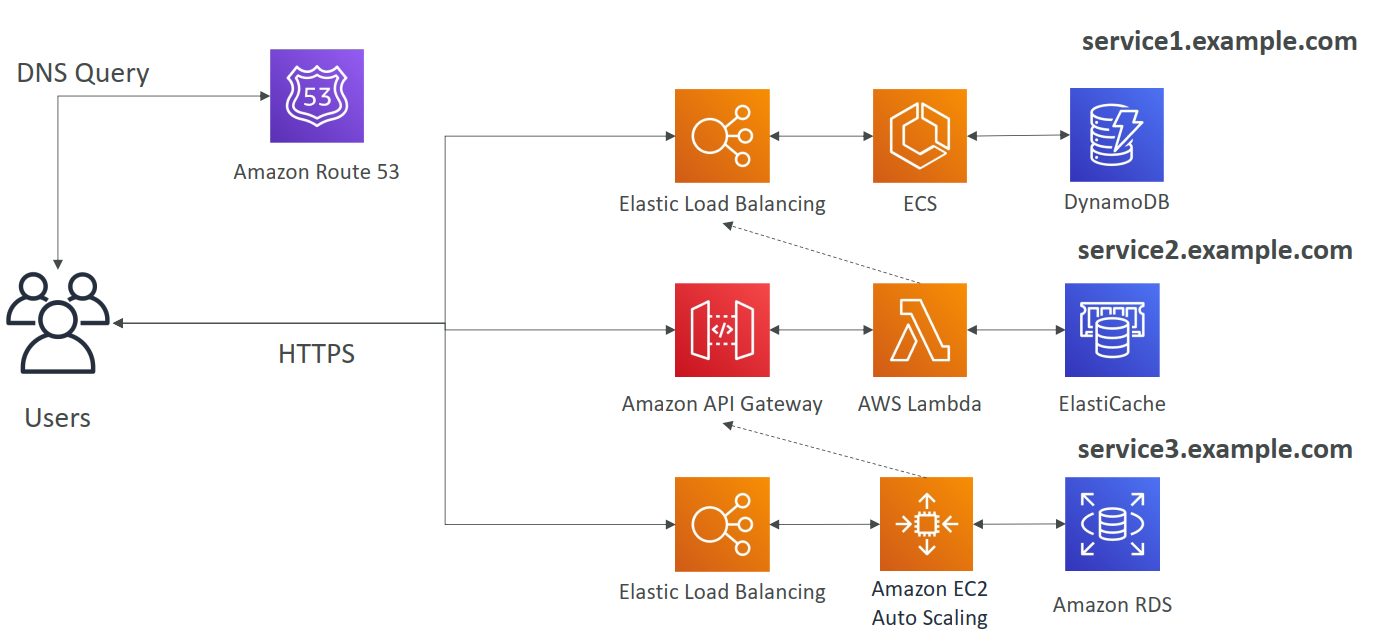
• The lambda function had an IAM role which could use SES

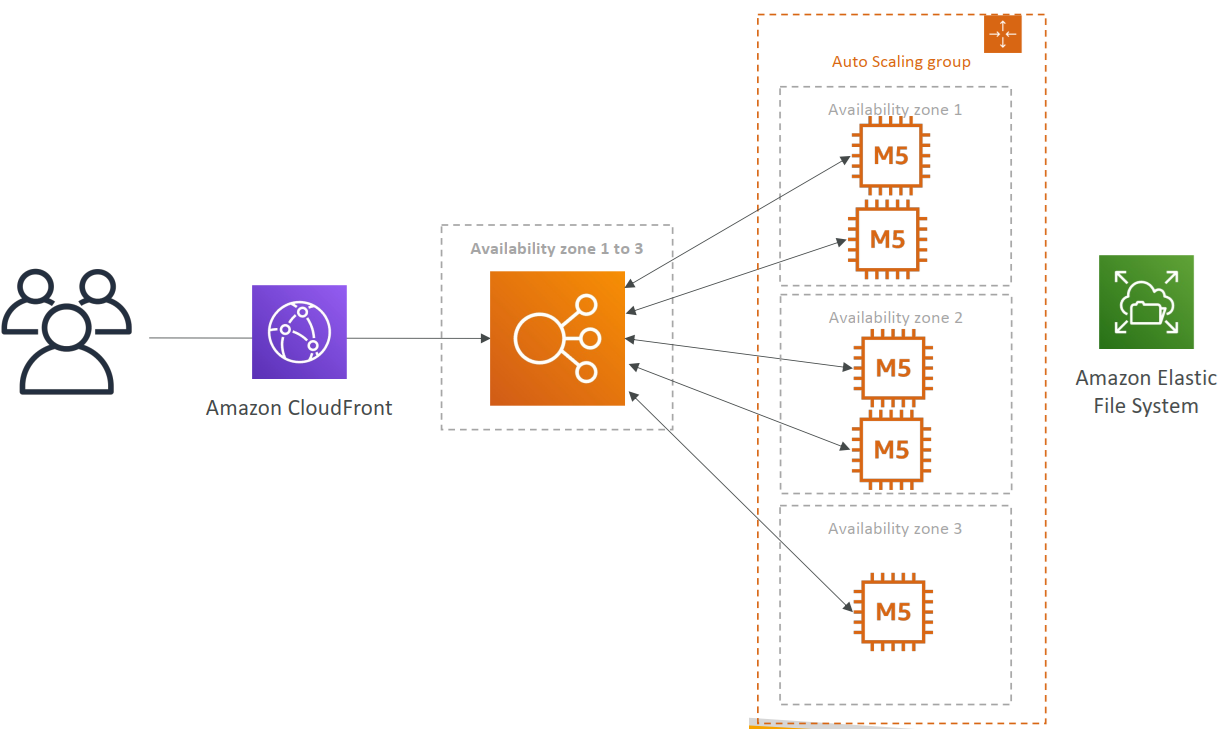
• SES (Simple Email Service) was used to send emails in a serverless way

• S3 can trigger SQS / SNS / Lambda to notify of events

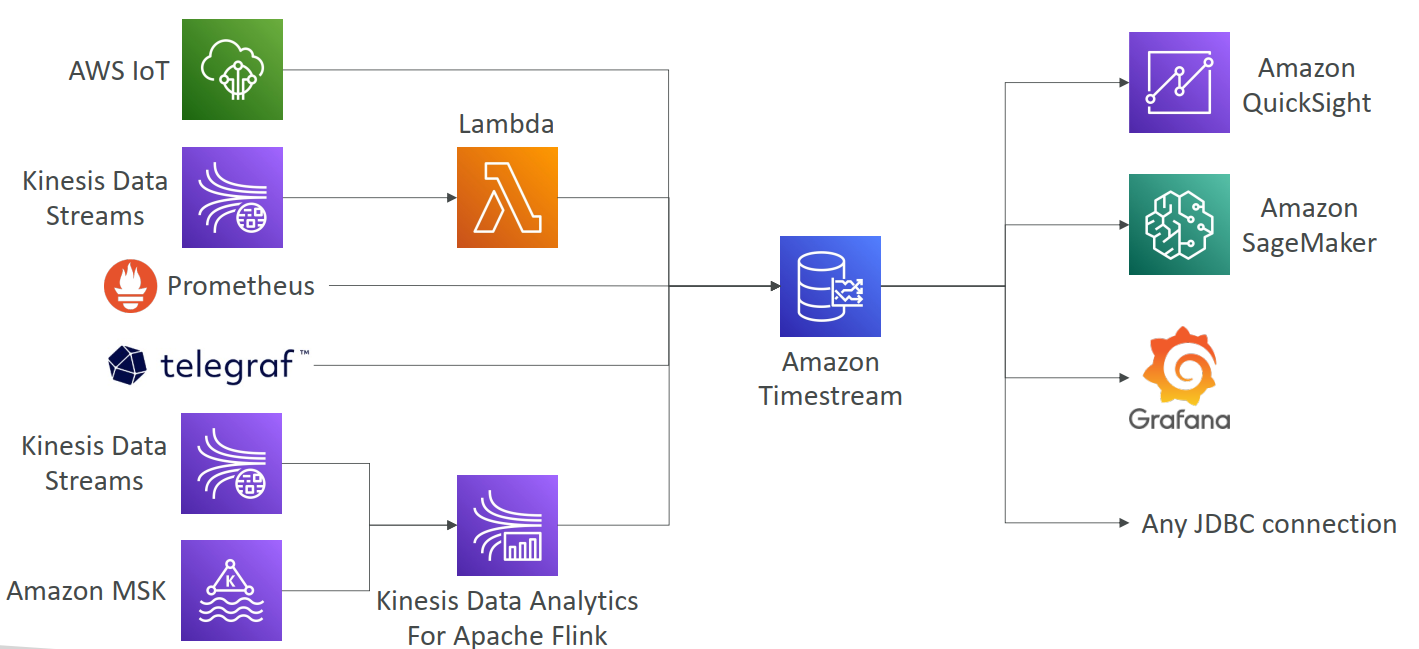


## Micro Services Environment





## Amazon Timestream



## Big Data Ingestion Pipeline

