Solution Architecture

1 What Is The Time.com { stateless Web-App }

Stateless Web App: WhatIsTheTime.com

- 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
- Let's go through the Solutions Architect journey for this app

say more users are coming in since the app is getting popular, so we will have to improve the instance type

say t2 → m5

How required to upgrade is a downtime

Vertical scaling

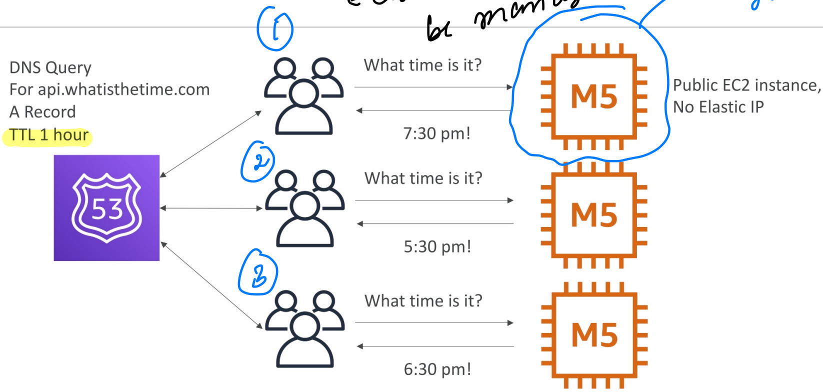
so no - one is able to access the application bad :c

when adding more instances (scaling horizontally) all of the new instances added will have elastic IPs

but the users needs to be aware of more & more IPs and we need to manage the infrastructure.

Instead! add route 53 so that no elastic IPs need to be managed

lets say this instance is gone

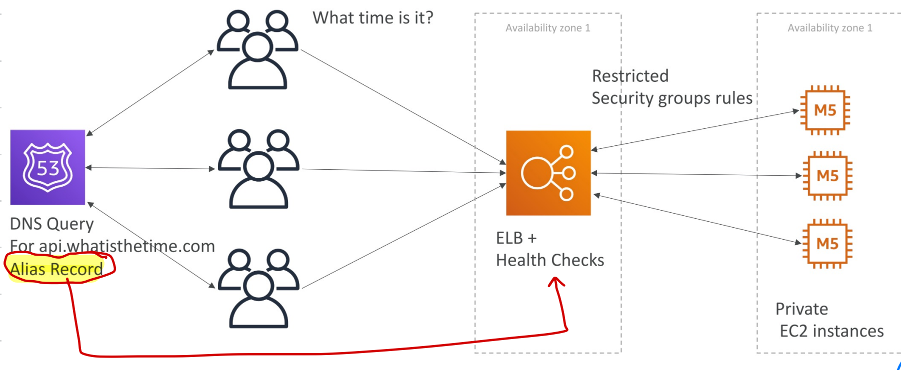


but since the TTL is 1 hr ∴ users 1 will still try to connect to M5 1

bad :(

Instead add load balancers to distribute

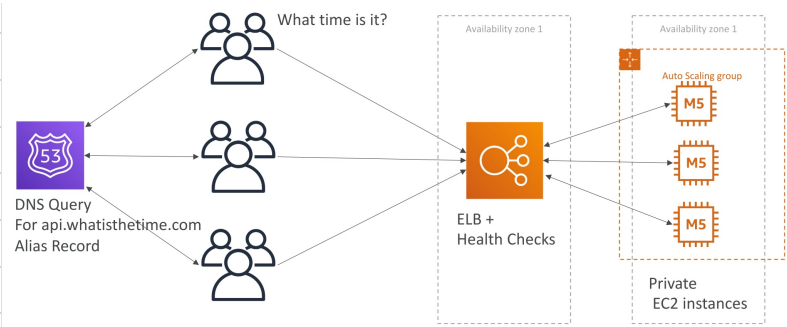
DNS name of ELB as a record in route 53



now we get proper health checks

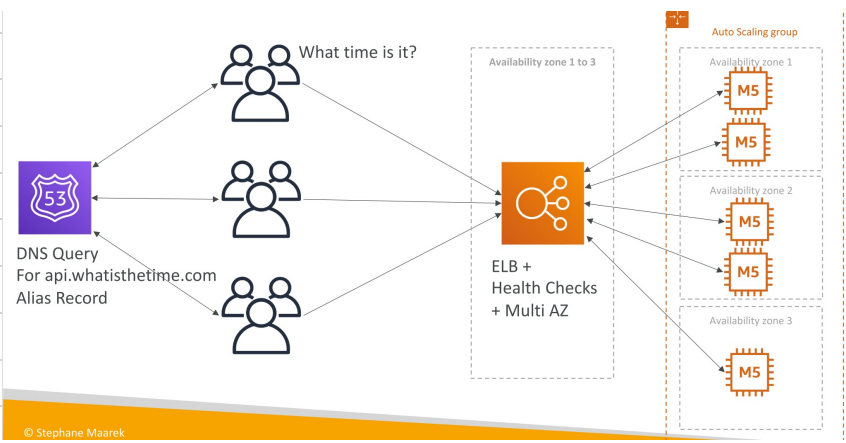
but adding & removing instances is hard

Add EC2's into autoscaling group!



But since AS is only 1 AZ hence any AZ disaster will bring the entire system down.

Need a multi A-Z setup!



resilient to failures.

→ we know that some instance will keep running so we can also increase capacity into them!

→ How how we considered the 5 pillars of AWS well architected framework?

1. Cost → scaling up & down [using ASG?, reserve instances
2. Performance → vertical scaling, ELB, ASG

(3) Reliability → Route 53 can be used to reliably route the traffic, Multi AZ ELB, Multi AZ ASGs

(4e) Security → Security groups for ELB ↔ EC2

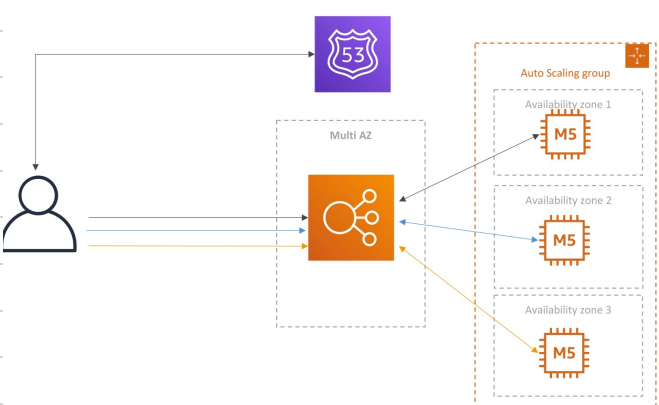
(5) Operational excellence → Manual process to automatic like ASG & etc.

(2) My Clothes.com { Stateful App }

Stateful Web App: MyClothes.com

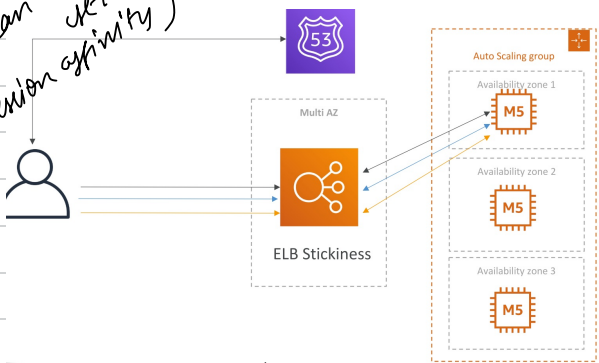
- MyClothes.com allows people to buy clothes online.
- There's a shopping cart → session data
- Our website is having hundreds of users at the same time
- We need to scale, maintain horizontal scalability and keep our web application as stateless as possible
- Users should not lose their shopping cart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

- Let's see how we can proceed!



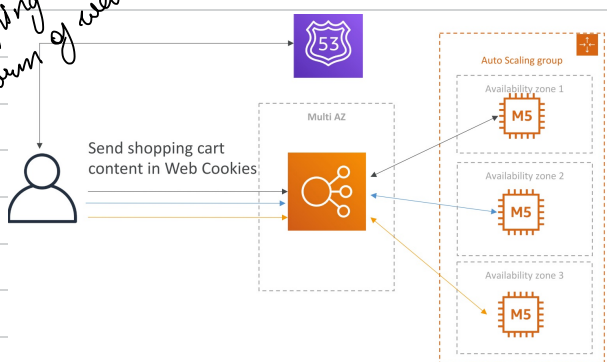
→ but if for the next session the same user re-routed to other EC2 instance will loose his shopping cart because its in a stateless form and the session data is not being stored.

Instead we can add stickiness (session affinity)



→ but if the instance fails the user will lose his shopping cart!

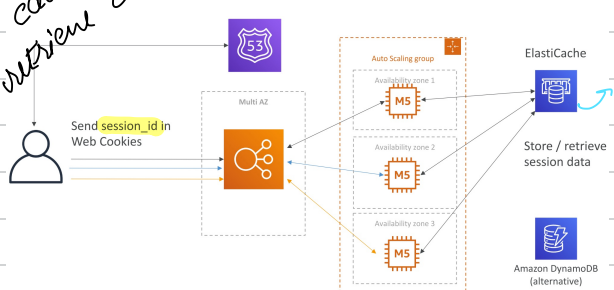
Instead the user stores the shopping cart in the form of web cookies.



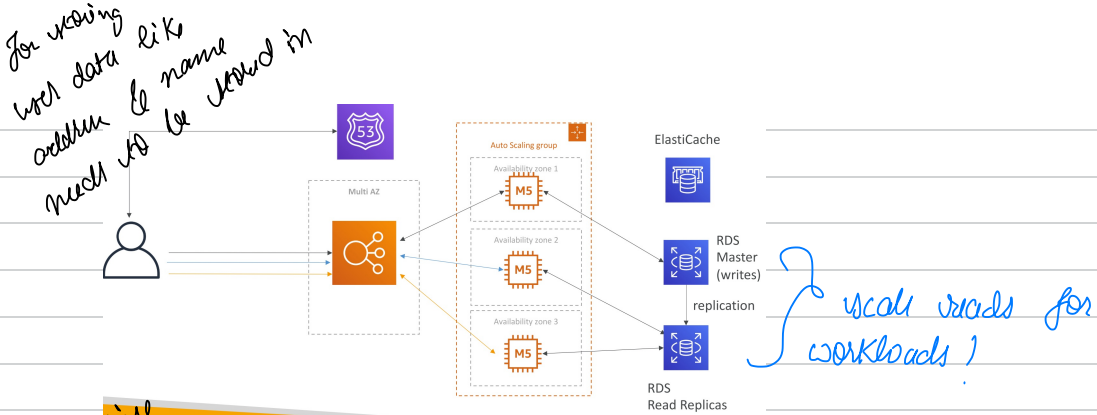
→ This approach makes our app

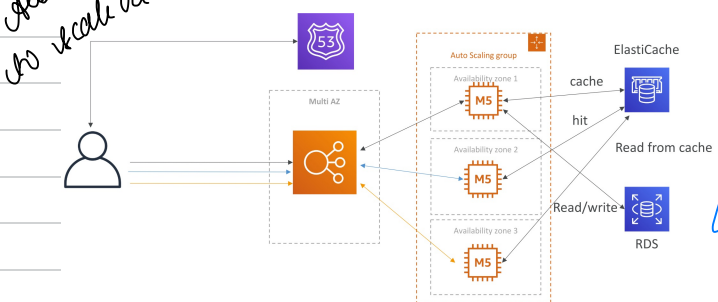
1. stateless
2. http requests are heavier because this data is being shared every time
3. cookies can be altered so cookie validation & size needs to be taken care of.

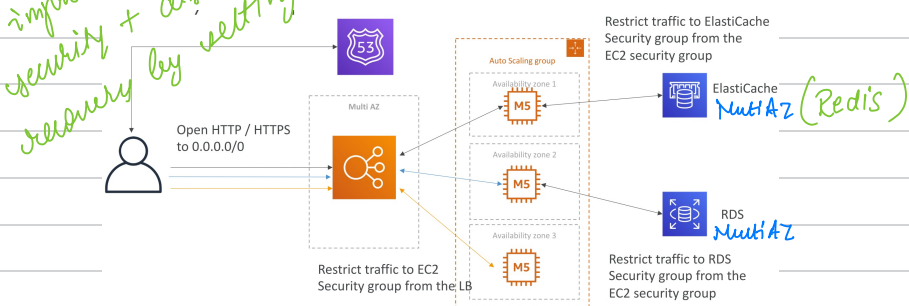
Instead we can use elastic cache to store & retrieve easily



sub-millisecond response







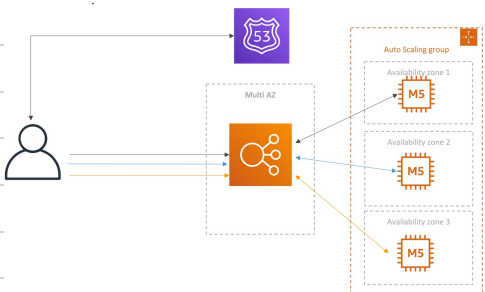
3-tier architectures for web applications

- ELB sticky sessions
- Web clients for storing cookies and making our web app stateless
- ElastiCache
  - For storing sessions (alternative: DynamoDB)
  - For caching data from RDS
  - Multi AZ
- RDS
  - For storing user data
  - Read replicas for scaling reads
  - Multi AZ for disaster recovery
- Tight Security with security groups referencing each other

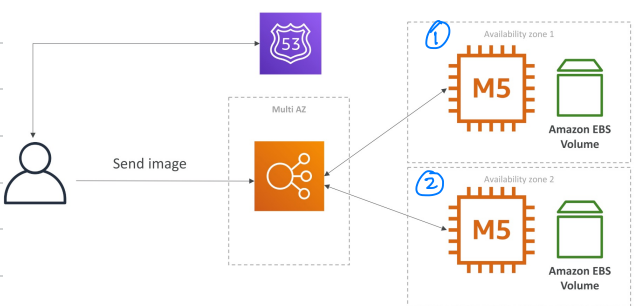
(3) Wordpress application ⟨ stateful ?

Stateful Web App: MyWordPress.com

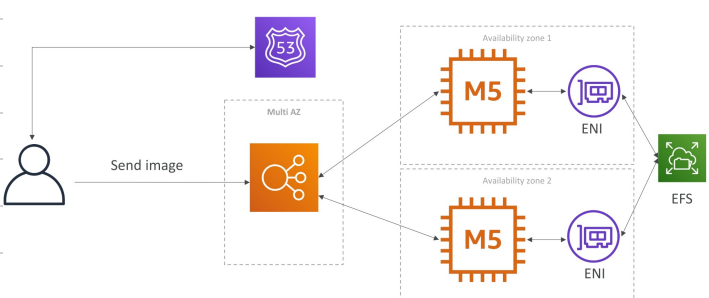
- We are trying to create a fully scalable WordPress website
- We want that website to access and correctly display picture uploads
- Our user data, and the blog content should be stored in a MySQL database.



→ Atomic 3 tier archi. for storing user data
→ we can use Aurora for Multi AZ & Read replicas & global DBs



For storing & serving images EBS will work perfectly & cheaply if only one instance need to be operated but in production if we want to scale then (1) से store होगा EBS (2) से नहीं होगा !

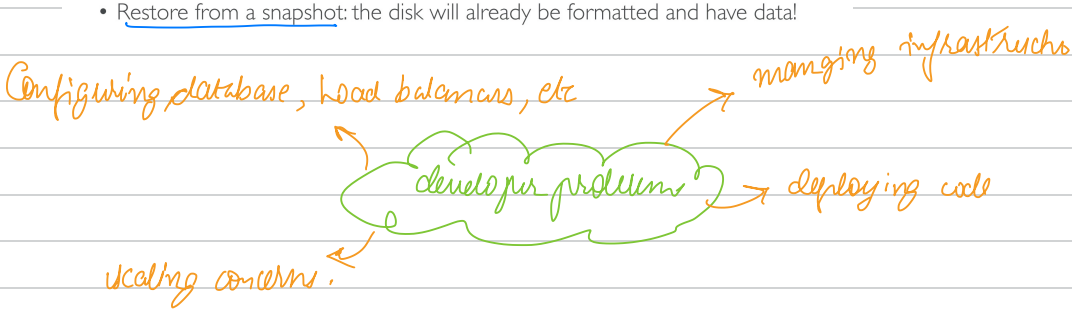


For that we can use a EFS which can be attached to multiple EC2 instances

To launch any application we need to provision EC2, RDS, ELB, ASG adding security groups etc. Hence all these configurations are needed to be done with proper cloud practices

(1) Manual configuration

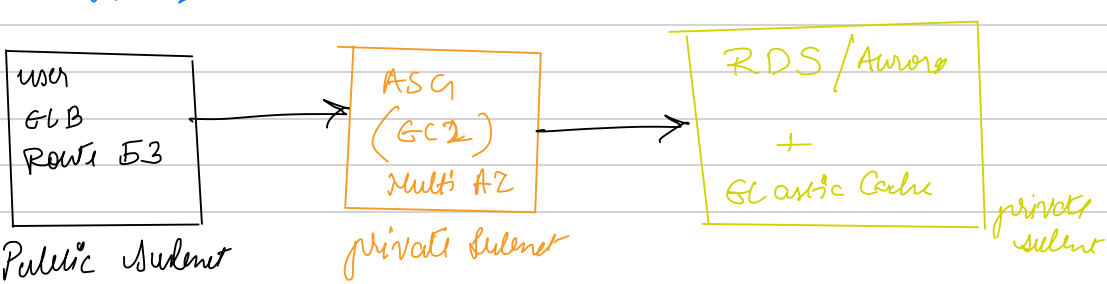
- EC2 Instances:
  - **Use a Golden AMI:** Install your applications, OS dependencies etc.. beforehand and launch your EC2 instance from the Golden AMI
  - **Bootstrap using User Data:** For dynamic configuration, use User Data scripts
  - **Hybrid:** mix Golden AMI and User Data (Elastic Beanstalk)
- RDS Databases:
  - Restore from a snapshot: the database will have schemas and data ready! { insert करते करते परेशान }
- EBS Volumes:
  - Restore from a snapshot: the disk will already be formatted and have data!



(2) Automatic configuration

By using beanstalk, developer will focus only on code & its deployement and managing all these will be on the fly.

typically most of the web application are 3-tier



Beanstalk

Platform as a Service (PaaS)

Elastic Beanstalk – Overview

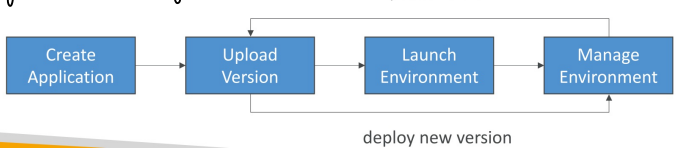
- Elastic Beanstalk is a developer centric view of deploying an application on AWS
- It uses all the component's we've seen before: EC2, ASG, ELB, RDS, …
- Managed service
  - Automatically handles capacity provisioning, load balancing, scaling, application health monitoring, instance configuration, …
  - Just the application code is the responsibility of the developer
- We still have full control over the configuration
- Beanstalk is free but you pay for the underlying instances

also provides health monitoring for our app.

Component of Beanstalk.

- **Application**: collection of Elastic Beanstalk components (environments, versions, configurations, …)
- **Application Version**: an iteration of your application code
- **Environment**
  - Collection of AWS resources running an application version (only one application version at a time)
  - **Tiers**: Web Server Environment Tier & Worker Environment Tier
  - You can create multiple environments (dev, test, prod, …)

- Platforms → A predefined runtime environment.

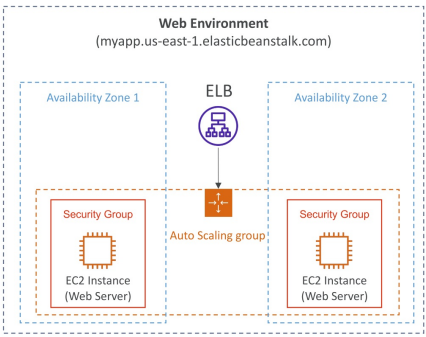


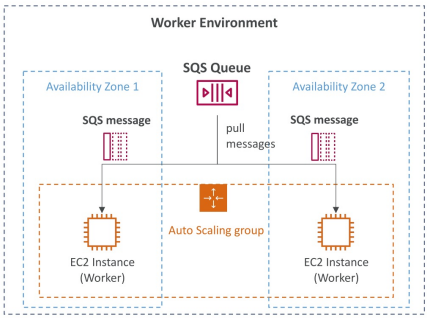
- Go
- Java SE
- Java with Tomcat
- .NET Core on Linux
- .NET on Windows Server
- Node.js
- PHP
- Python
- Ruby
- Packer Builder
- Single Container Docker
- Multi-container Docker
- Preconfigured Docker

- Also u can write ur custom platform code :)

Supported languages & platforms.

Web Server Tier vs. Worker Tier

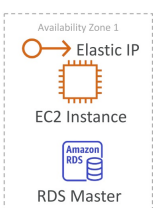




- Scale based on the number of SQS messages
- Can push messages to SQS queue from another Web Server Tier

Elastic Beanstalk Deployment Modes

Single Instance
Great for dev



High Availability with Load Balancer
Great for prod