

# GAME ARCHITECTURE MMO SHOOTER GAME

Allampalam Anuj  
Ardra Ayyappath  
Suyash Baranwal

# Objective

Provision backend architecture for an MMO Shooter Game





## Assumptions

- 40 million players at peak time
- Global provisioning
- 2 patch updates in a week
- Front end and game logic is handled by the client.
- No budget constraints



## Scope

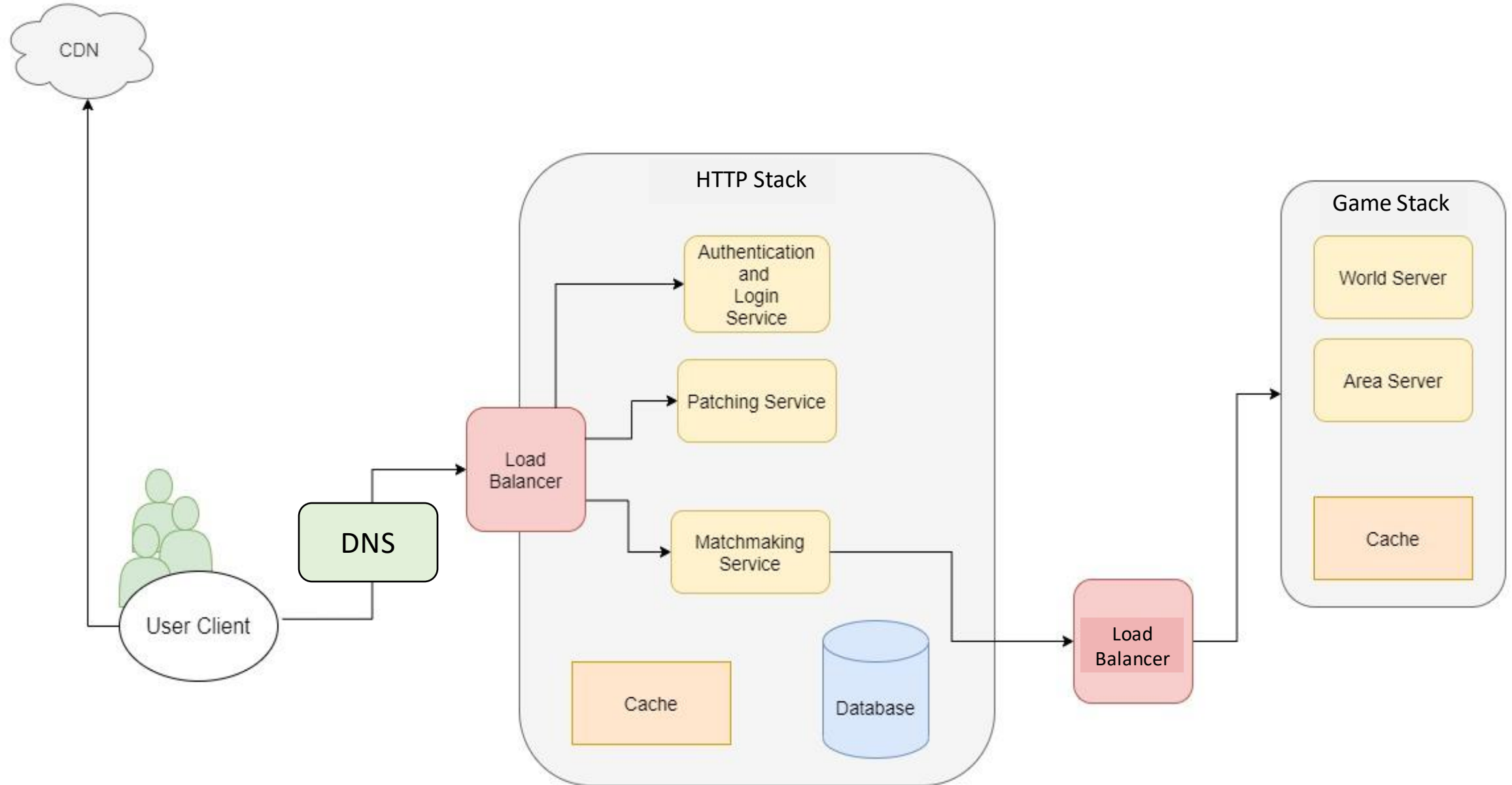
- Implementation of backend architecture ensuring low-latency game play.
- Cost estimation is out of scope
- Latency quantification is out of scope



## Approach/methodology

- Game servers are located near players
- Everything else is handled by HTTP API
- Data replication is avoided, cache is used.
- No one really plays everywhere
- time critical (game play) and non-time critical (HTTP, pre-gameplay)
- Game is hosted on AWS

# Bird's eye view : Block Diagram



# Basic logic

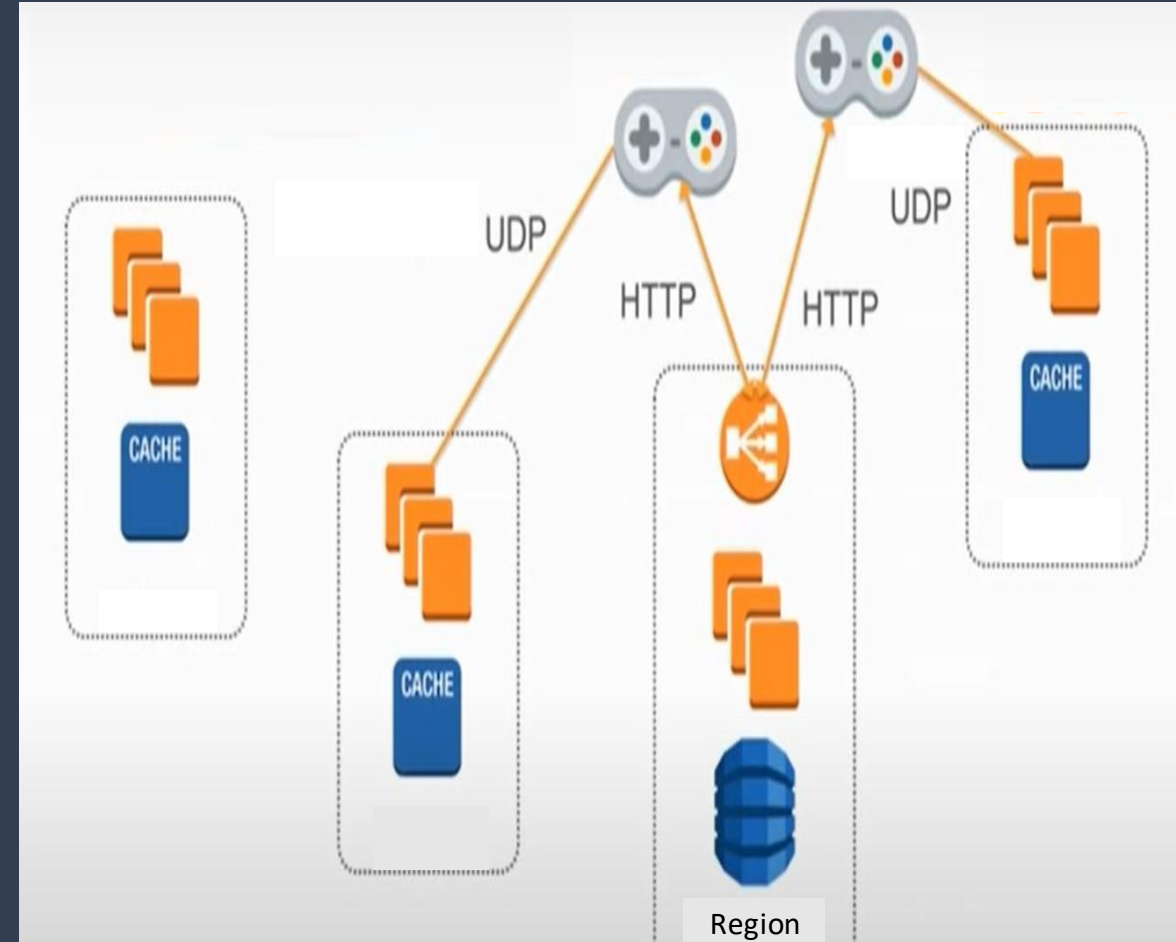
The loosely coupled stacks can communicate with each other, but are least dependent on each other.

HTTP stack handles pre-gameplay -

- Authentication
- Patching and updates
- Matchmaking

Game stack handles game play and all games stacks are connected to the HTTP stack.

- The centralized HTTP stack has dedicated database with data of all game stacks.
- Pre-gameplay : HTTP API calls. Game Play : UDP & RUDP (faster than TCP)





# Game Flow



Log in via HTTP  
API



Download game  
assets



Matchmaking to  
server



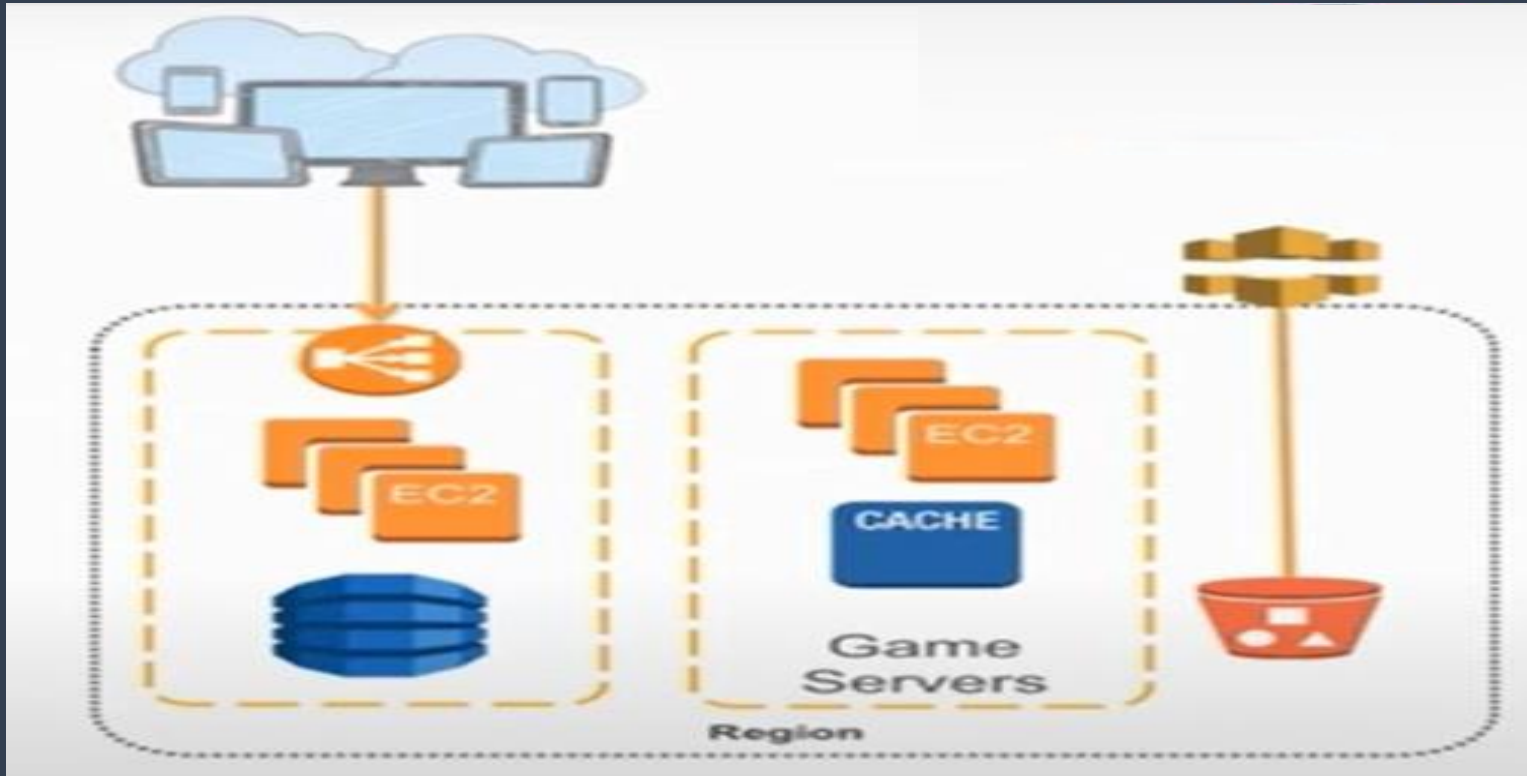
Connect to  
server



Game begins



Game ends



Write via HTTP

# HTTP Stack

## Authentication and Authorization

- Handled using AWS Cognito(IAM service)
  - Authentication – Integrate with social media logins
  - Authorization –different levels(User, Group, Role, Policy)

1

## Patching and Updating

- Handled by AWS S3(storage)+Cloudfront(CDN service)
- Update/Patch is pushed to S3 and distribution is handled by CloudFront(regional and EL Cache)

2

## Matchmaking

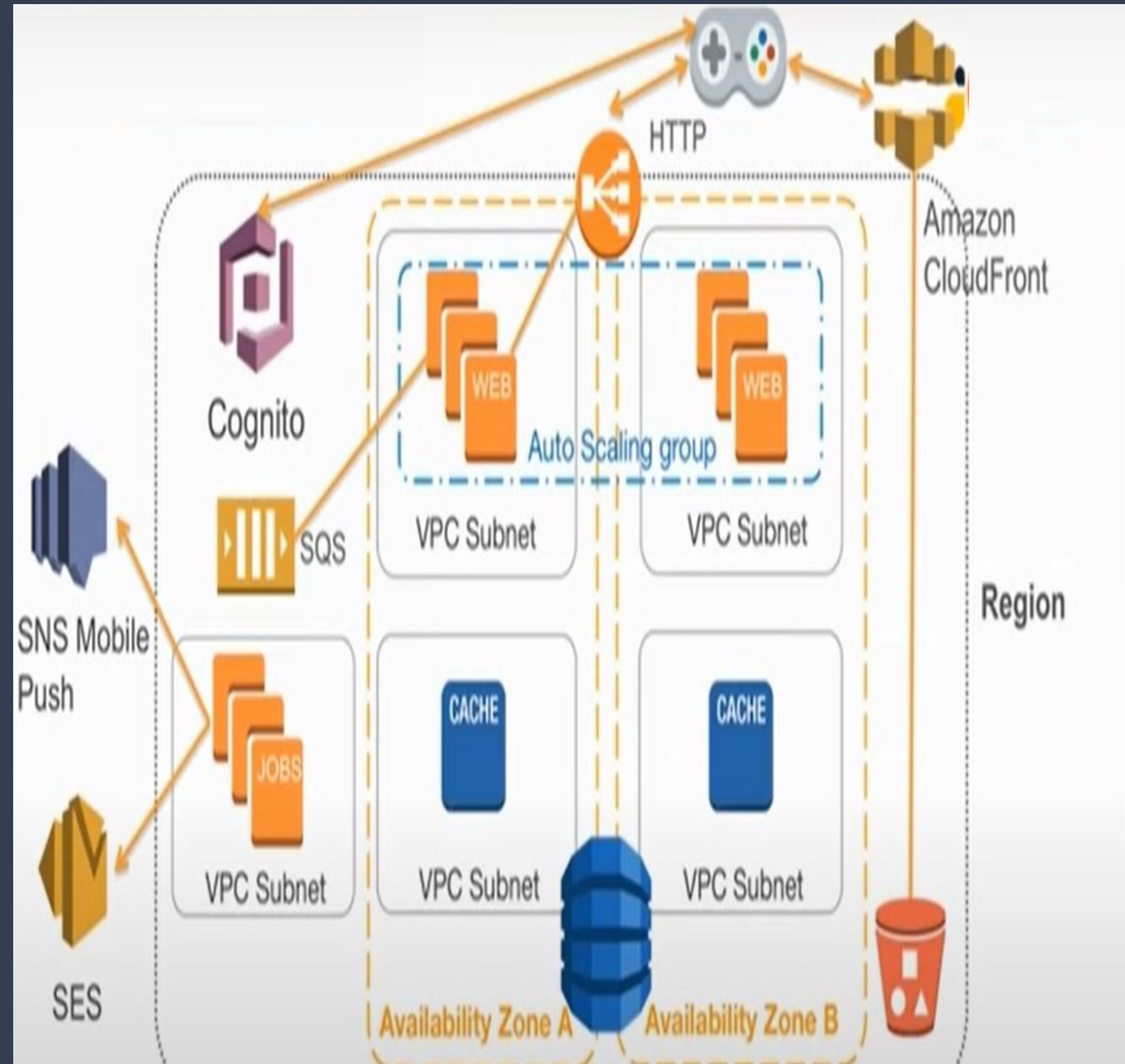
- Db has info about active game stacks in that regions
- ELB to decide which game stack is best for the player.

3

## Notifications and Emails

- Push notifications on the device
- For both end users and engineers(ASG, ELB Healthcheck etc)

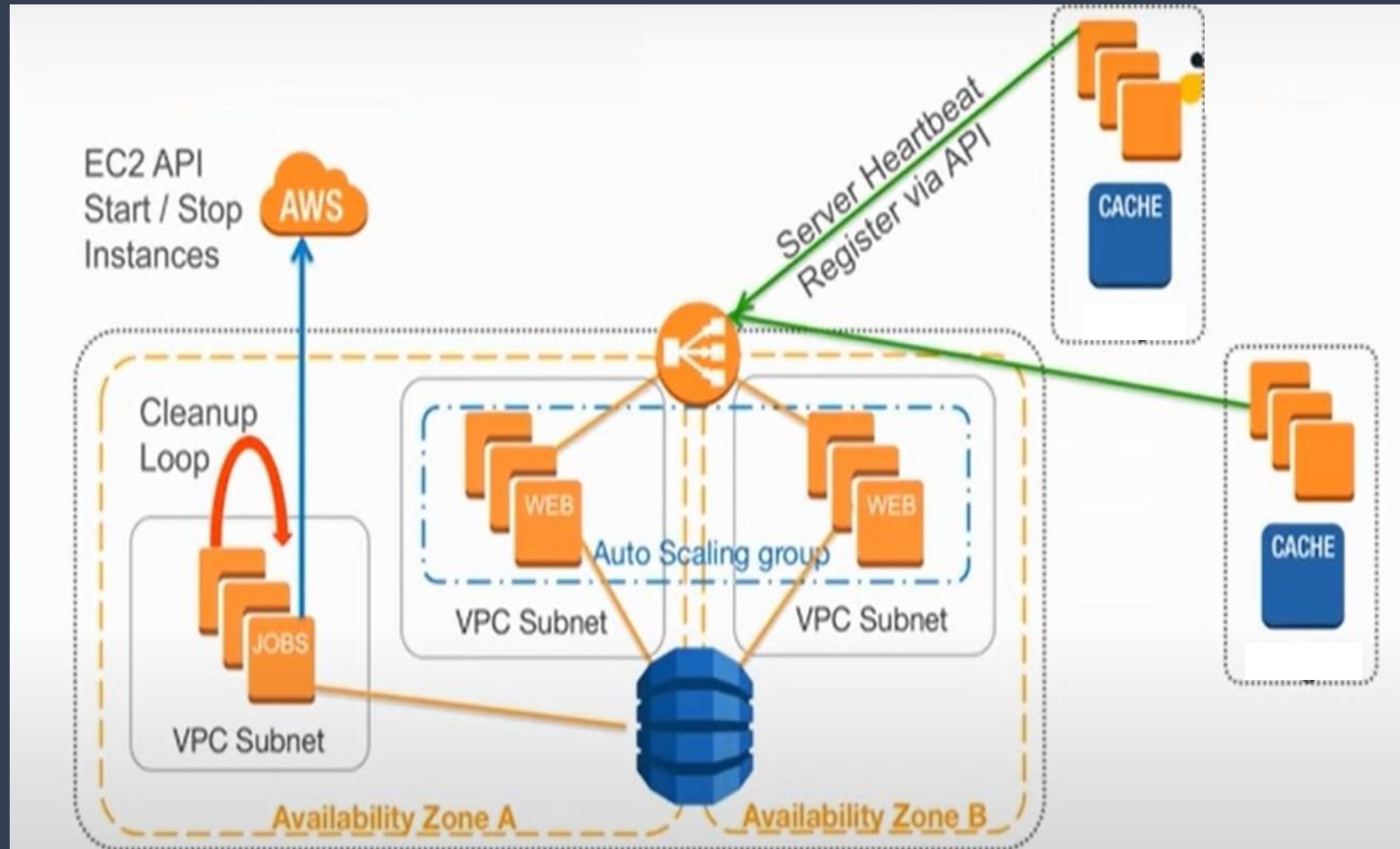
4



# Game Stack Registration

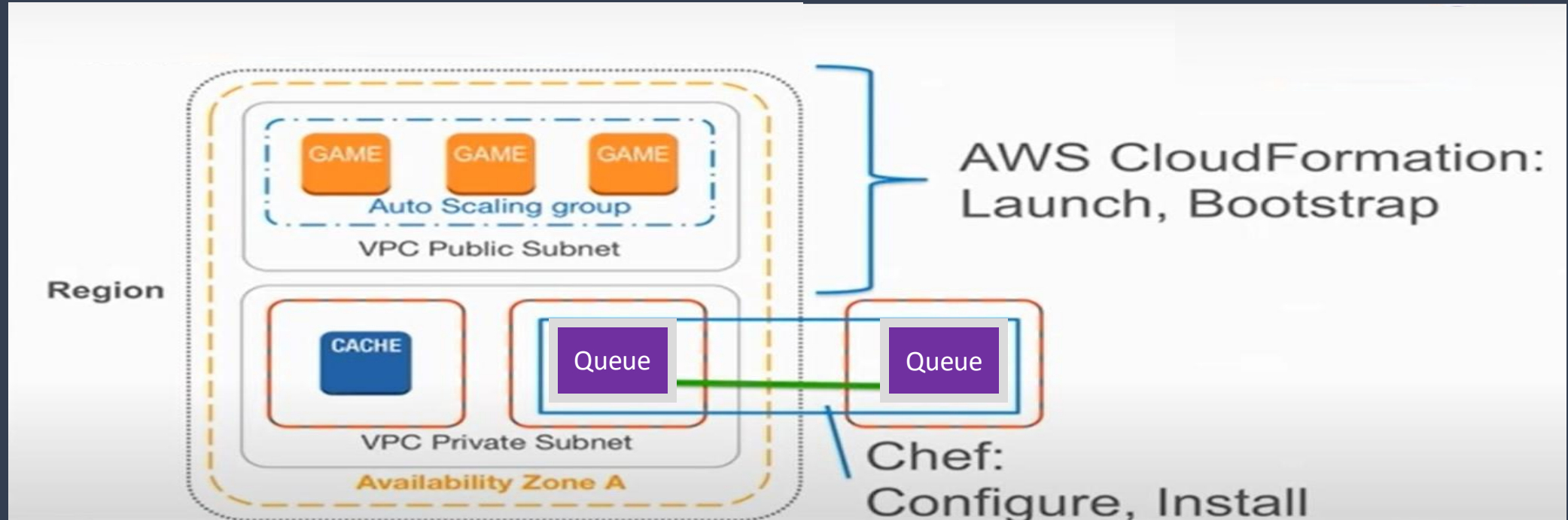
Cleanup loop checks DB at intervals and removes data related to game stacks whose last update is outdated.

Availability handled using heartbeat between Game Stack and HTTP Server





# Cloud Formation and Chef



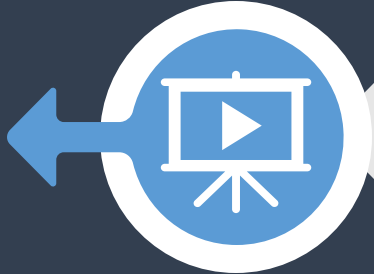
Set of **Infrastructure as Code(IAC)** and automation tool used to configure the entire system of servers

Cloud Formation : describes EC2 instances and the AWS resources and creates a **JSON template** out of them

These templates can then be used to **replicate the resources** in other regions or even same region

Chef runs a bunch of scripts having instructions of software and its configuration/installation specification

# Logs and Analytics



Kinesis Data Firehose sends data to S3 , Amazon Redshift, and enables real-time data analysis on the streaming data.



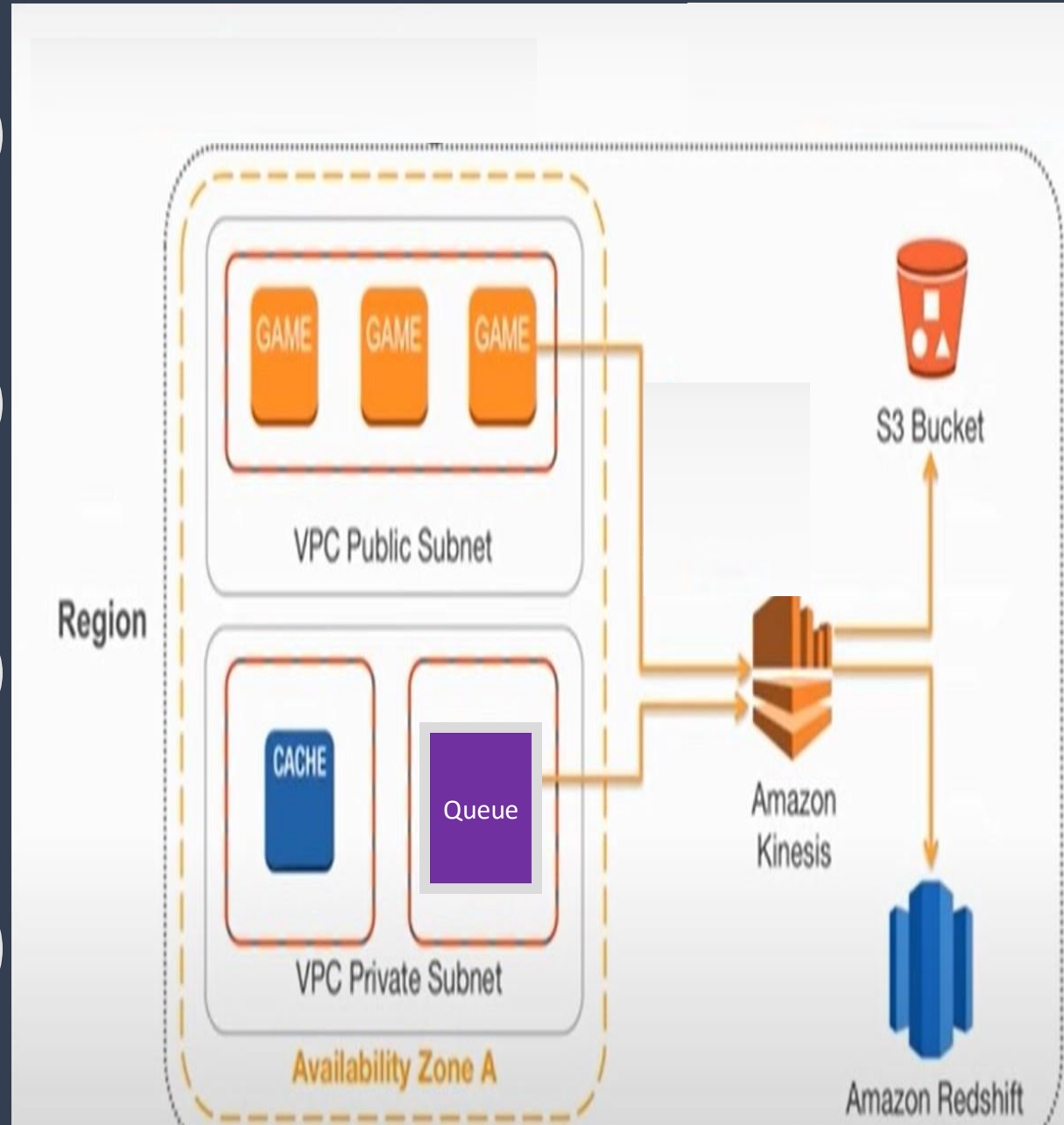
APIs can be used to auto-scale the capacity. Data can also be transferred from DynamoDB for analysis.



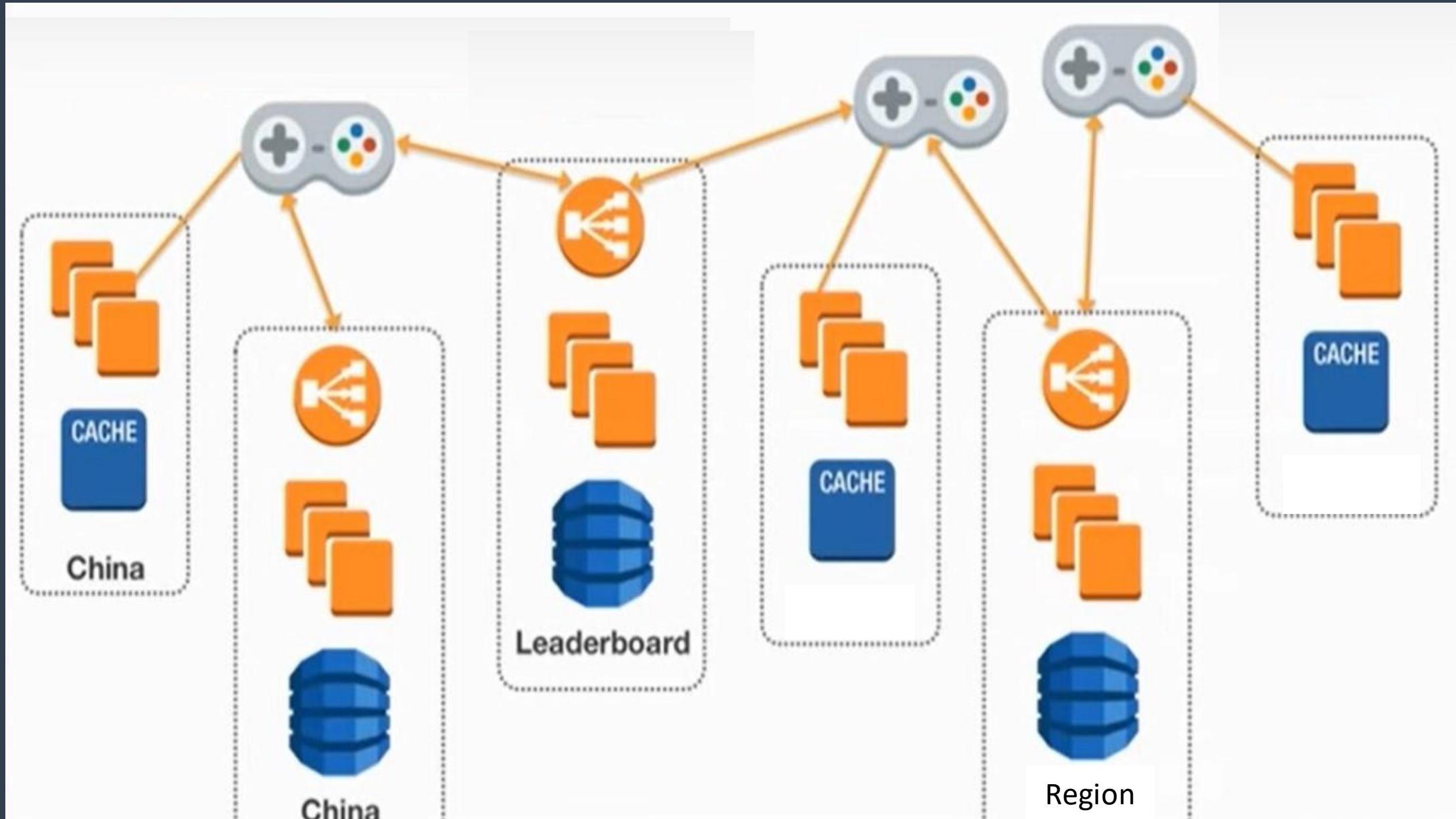
Amazon redshift is optimized for data sets ranging from a few hundred gigabytes to a petabyte



Amazon Redshift delivers fast query and I/O performance for virtually any size dataset by using columnar storage technology while parallelizing and distributing queries across multiple nodes



# Special Cases





# Possible HTTP Requests

Client			Server
Authentication	Title-Wide Data Management	Shared Group Data	Authentication
AddUsernamePassword	GetCatalogItems	AddSharedGroupMembers	AuthenticateSessionTicket
LoginWithAndroidDeviceID	GetStoreItems	CreateSharedGroup	Account Management
LoginWithFacebook	GetTitleData	GetSharedGroupData	GetUserAccountInfo
LoginWithGameCenter	GetTitleNews	RemoveSharedGroupMembers	SendPushNotification
LoginWithGoogleAccount	AddUserVirtualCurrency	UpdateSharedGroupData	Player Data Management
LoginWithIOSDeviceID	Player Item Management	Server-Side Game Logic	GetLeaderboard
LoginWithPlayFab	ConsumeItem	GetLogicServerUrl	GetLeaderboardAroundUser
LoginWithSteam	GetUserInventory		GetUserData
RegisterPlayFabUser	RedeemCoupon		GetUserInternalData
SendAccountRecoveryEmail	SubtractUserVirtualCurrency		GetUserReadOnlyData
Account Management	UnlockContainerItem		GetUserStatistics
GetAccountInfo	StartPurchase		UpdateUserData
GetPlayFabIDsFromFacebookIDs	PayForPurchase		UpdateUserInternalData
GetUserCombinedInfo	ConfirmPurchase		UpdateUserReadOnlyData
LinkFacebookAccount	PurchaseItem		UpdateUserStatistics
LinkGameCenterAccount	Friend List Management		Title-Wide Data Management
LinkSteamAccount	AddFriend		GetCatalogItems
UnlinkFacebookAccount	GetFriendsList		GetTitleData
UnlinkGameCenterAccount	RemoveFriend		SetTitleData
UnlinkSteamAccount	SetFriendTags		Player Item Management
UpdateEmailAddress	IOS-Specific APIs		AddUserVirtualCurrency
UpdatePassword	RegisterForIOSPushNotification		GetUserInventory
UpdateUserTitleDisplayName	ValidateIOSReceipt		GrantItemsToUsers
Player Data Management	Matchmaking APIs		SubtractUserVirtualCurrency
GetFriendLeaderboard	GetCurrentGames		Matchmaking APIs
GetLeaderboard	GetGameServerRegions		NotifyMatchmakerPlayerLeft
GetLeaderboardAroundCurrentUser	Matchmake		RedeemMatchmakerTicket
GetUserData	StartGame		Steam-Specific APIs
GetUserReadOnlyData	Android-Specific APIs		AwardSteamAchievement
GetUserStatistics	AndroidDevicePushNotificationRegistration		
UpdateUserData	ValidateGooglePlayPurchase		
UpdateUserStatistics	Analytics		

# Example of Web API Call



## API Request:

```
POST https://3.playfabapi.com/Client/Matchmake
Content-Type: application/json;
X-Authentication: 7BC920BC255F7E60-0-0-5F4
{
  "BuildVersion": "5.01",
  "Region": "USCentral", "GameMode": "0",
  "LobbyId": "Lobby 32",
  "EnableQueue": false
}
```

## Response from Web Services:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
{
  "LobbyID": "Lobby 32",
  "ServerHostname": "192.168.0.1",
  "ServerPort": 7777,
  "Ticket": "e98yf289f248902f4904f0924f9pj",
  "Status": "waiting",
  "Queue": [
    "User1",
    "User2"
  ]
}
```



# Additional Important Points



## Choice of Instances

- HTTP Web Servers -- Storage optimized
  - **I3 and I3en** -- They are optimized to deliver tens of thousands of low-latency, random I/O operations per second (IOPS) to applications.
  - Default vCPUs – 2 to 96 RAM – 16 to 768 GB
- Game Servers -- **C5 EC2 instance**
  - Compute bound high performance processor
  - Latest and highly Cost optimized as compared to precursors(C4/C3)
  - Powered by AWS Nitro Systems(hypervisor that provides high bandwidth; great throughput and low latency)
  - vCPU : 2 to 72 RAM 4 to 192 GB Network B/W : upto 25Gbps
  - uses Elastic Network Adapter(ENA) for enhanced networking



## Auto Scaling Policy

- Target Tracking Scaling Policy – clearly define the target metric(say CPU utilization) and ASG scales to always stay near it.
- The developers set the maximum, minimum and desired capacity of ASG
- Scale-out and scale-in policies are always decided together and not in isolation
- It is a dynamic policy and can even update maximum and minimum ASG capacities to meet the target
- Best among other options – simple scaling policy, step-up policy, predictive scaling(rarely used)



## Health Checks

- EC2 instance 2/2 status check report (status must be “running”)
  - ELB’s instance health check report (status must be “in service”)
  - Elastic IP is released when an ASG terminates an unhealthy EC2 instance
- Note that the health must exclude the warm-up and cool-down period of instances(specified by us)

# Scenarios



# Player wants to connect to a match

---

**01**

Player contacts the matchmaking service and a flag against the player is set stating that the player is ready for matchmaking

---

**02**

A matchmaking request is sent through the load balancer to an appropriate game stack(nearest server is preferred)

---

**03**

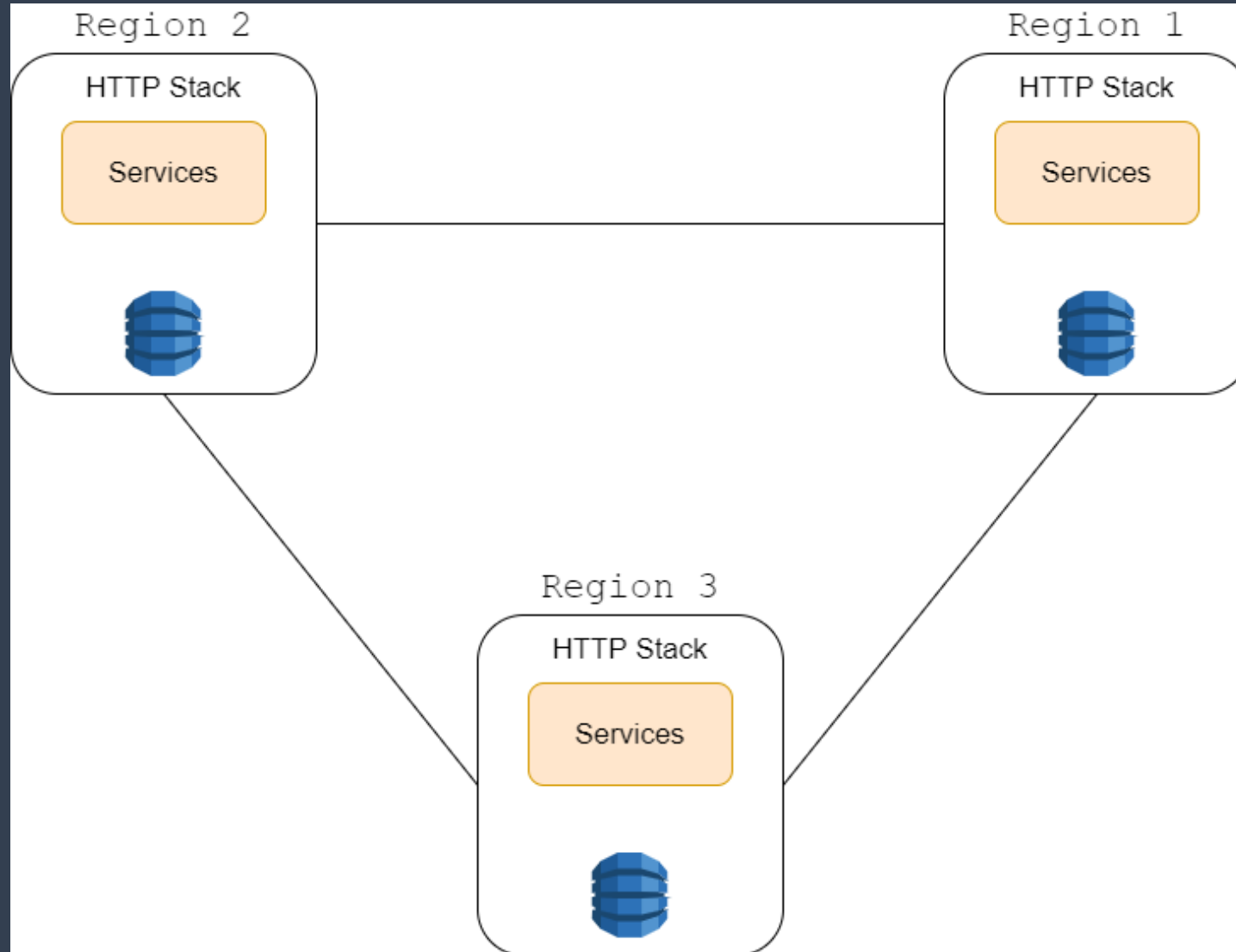
The appropriate game stack checks the availability of a match or creates a new match and sends a response with the player's match code.

---

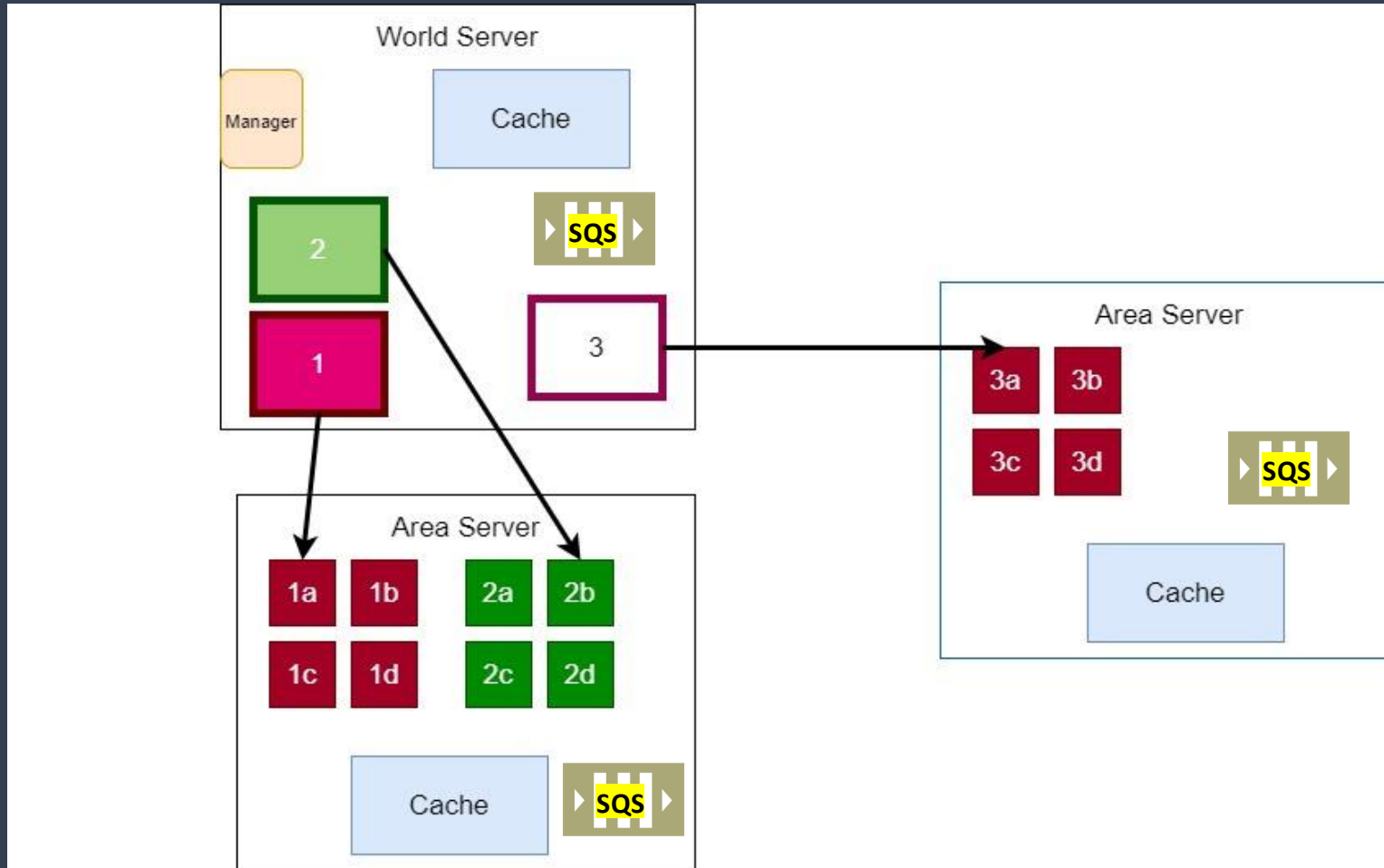
**04**

The player connects to the match using the code sent to him.

Player wants to play from a different region than he played from last time

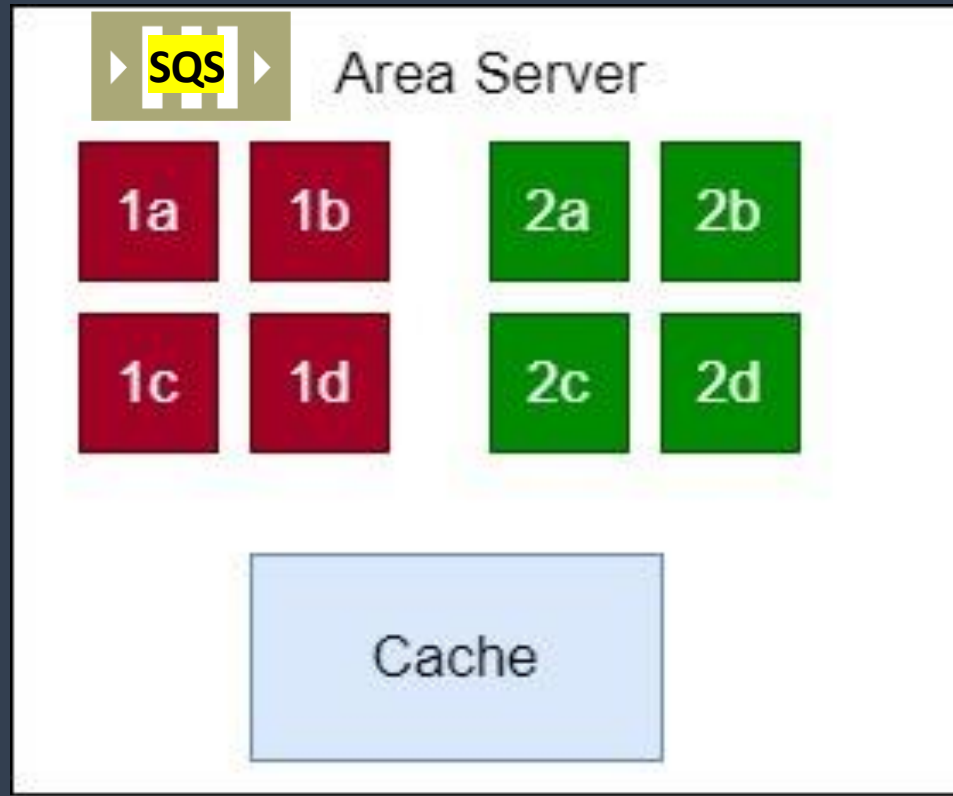


# What happens when a match instance or a map instance goes down during gameplay?





# How are across-the-map shots handled?



THANK YOU