# Learn to Design Cloud Architecture

# What is a URL Shortener?

- A URL shortener is a tool that takes a long web address (URL) and converts it into a much shorter one. It redirects anyone who clicks the short link to the original long URL.

- Example:
- Long URL:
  - https://www.longurl.com/very/very/long/url

- Shortened URL:
  - https://short.com/url



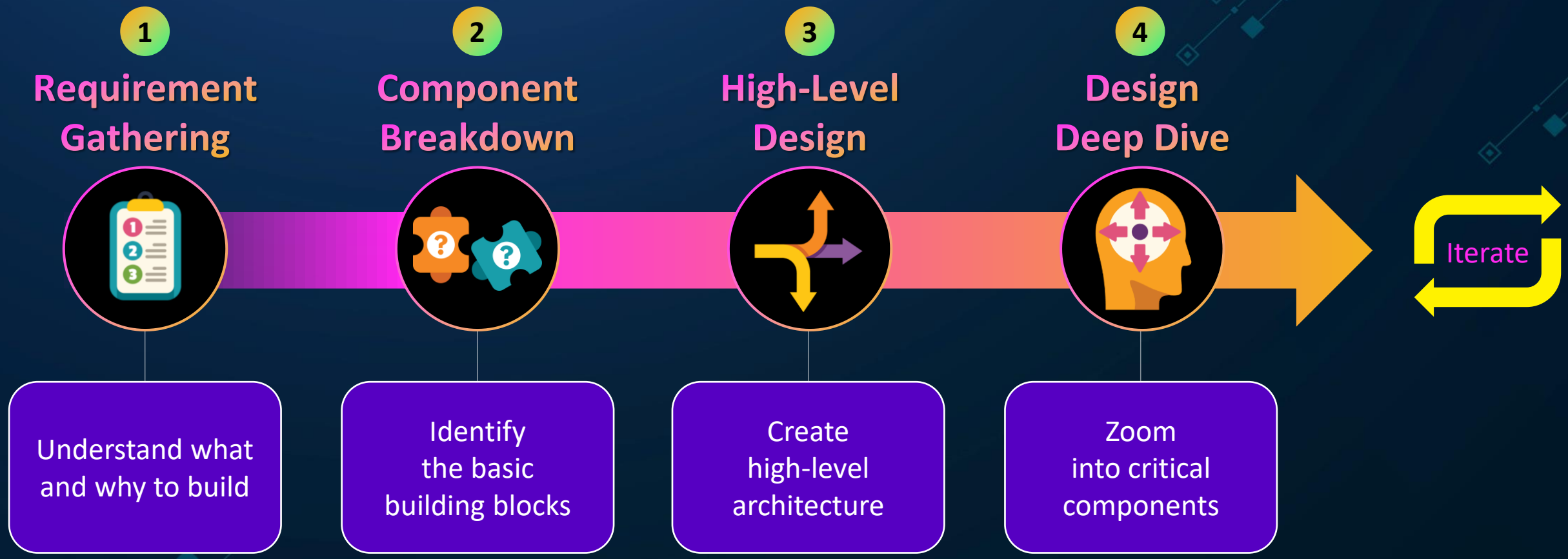


# A URL Shortener System


- Easy to remember / type
- Cleaner appearance
- Character space saving
- Tracking and analytics


- Originality of the brand lost
- Dependency on third-party
- Security Concern
- Additional Redirect time

# Designing a System – A simple framework




# Requirement Gathering

# Expectations from a URL shortener service

- Functional Requirement (What the systems should do?)
  - User can create generate a shortened URL by providing a long URL as input.
  - Users should be redirect to the long URL when the short URL is accessed.

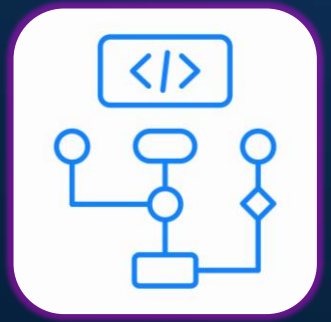
- Nice to have
  - Custom URL creation support
  - Custom domain name support
  - Analytics on the URL access patterns
  - Configurable URL expiration
  - APIs exposer to third-party clients

# Non-functional requirement

- Generate short URL from long URL:
  - 1,000 Requests Per Second (RPS)
- Short URL to long URL redirection:
  - 20,000 Requests Per Second (RPS)
- Average duration of URL persistence in system:
  - 10 years

# Component Breakdown

# Component Breakdown

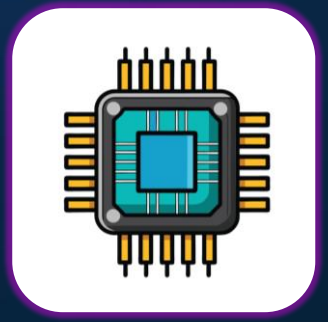


**Algorithm**

To create
Short URLs

**Redirection**

To redirect requests



**Compute**

To process
requests

**Storage**

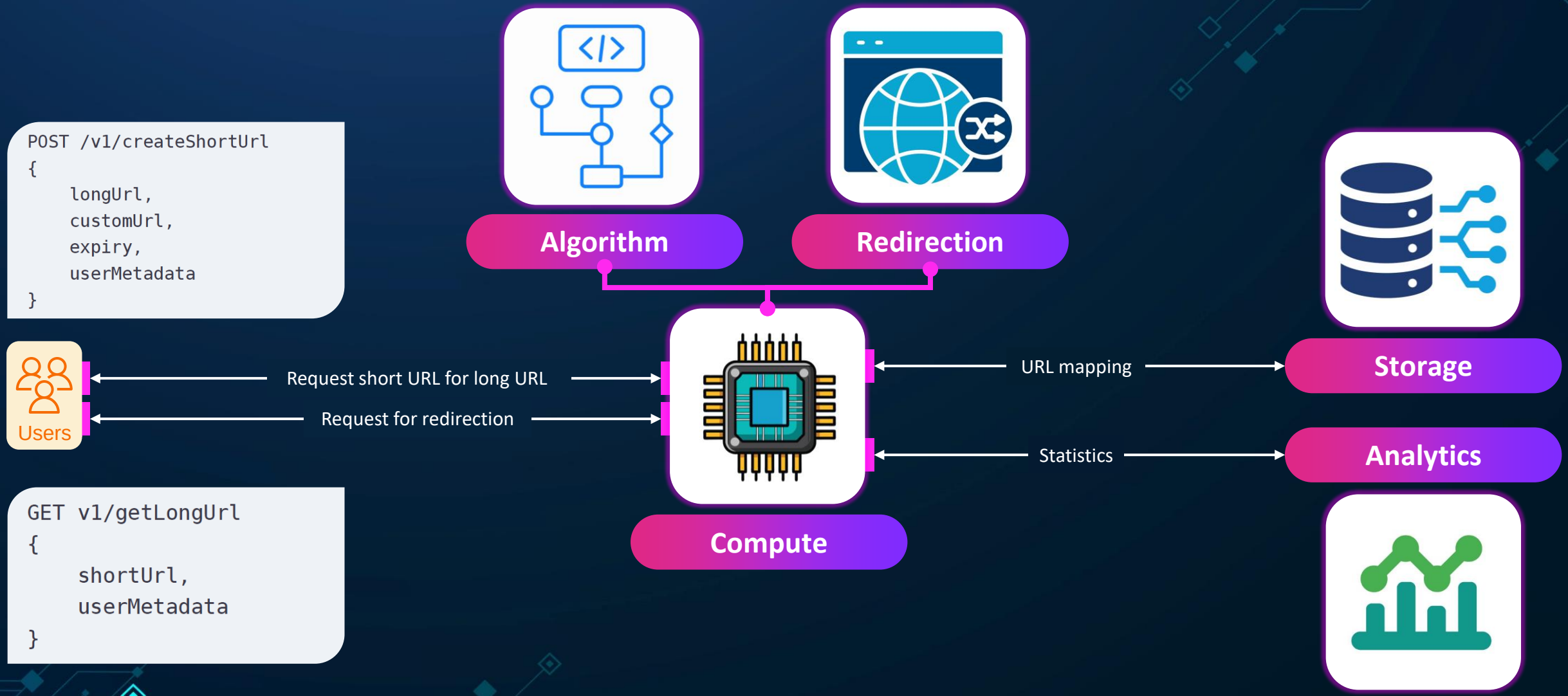
To store
URL mappings

**Analytics**

To generate
reports

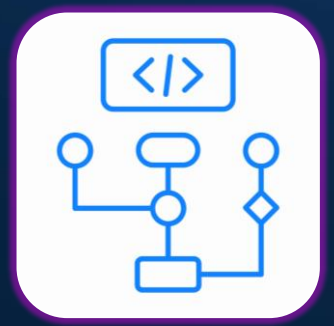
# High-Level Design

# High-level architecture



# Design Deep Dive

# Deep Dive

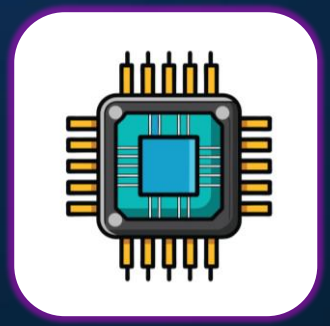


**Algorithm**

To create
Short URLs

**Redirection**

To redirect requests



**Compute**

To process
requests

**Storage**

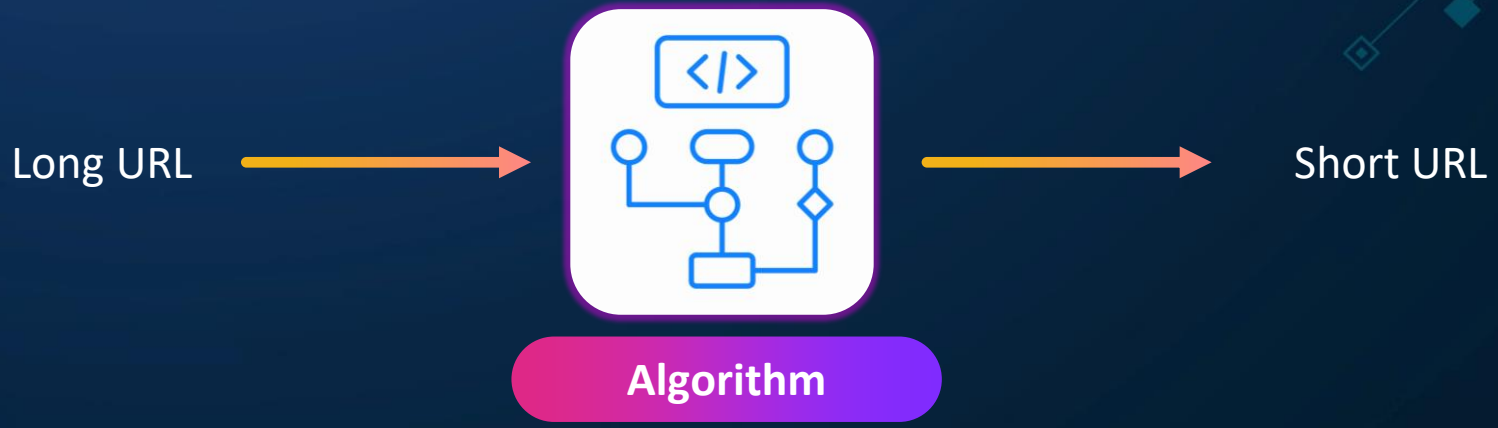
To store
URL mappings

**Analytics**

To generate
reports

# Algorithm

- Rules designed to perform a specific task.




- Requirements for short URL
  - Short
  - Random
  - Human readable

# URL Shortening Algorithm

- Hashing
  - Hashing is the process of converting data into a fixed-size string of characters, typically for fast data retrieval or comparison.
  - Common algorithms:
    - MD5 SHA-1 SHA-256 SHA-512
    - SHA-3 bcrypt sscrypt Argon2

https://www.md5hashgenerator.com/

| | |
|---|---|
| ✗ | Short |
| ✗ | Random |
| ✗ | Human Readable |

# URL Shortening Algorithm

- UUID (Universally Unique Identifier)
  - A UUID is a 128-bit number used to uniquely identify information across systems without significant risk of duplication.
  - UUIDs are perfect when you need uniqueness without coordination across systems.

https://www.uuidgenerator.net/

| | |
|---|---|
| ☒ | Short |
| ☑ | Random |
| ☒ | Human Readable |

# URL Shortening Algorithm

- Random ID Generator
  - Auto Incremental
    - MySQL's AUTO_INCREMENT or PostgreSQL's SERIAL or SEQUENCE or Redis's Increment feature
  - Challenge
    - Predictable
    - DB dependency

| | |
|---|---|
| ☑ | Short |
| ☒ | Random |
| ☑ | Human Readable |

# URL Shortening Algorithm

- Random ID Generator
  - Base 62 encoding
    - a to z (26 characters) + A to Z (26 characters) + 0 to 9 (10 characters) = 62 characters

High level overview :

1 - Convert characters in ASCII
2 - Split them in group of 6
3 - Convert it back to decimal
4 - Map those characters to base62 index table

| | |
|---|---|
| ☑ | Short |
| ☑ | Random |
| ☑ | Human Readable |

| Language | Common Function Name | Comment |
|---|---|---|
| Python | `encode_base62(num)` | Typically custom implemented |
| JavaScript | `encodeBase62(number)` | camelCase naming |
| Java | `encodeBase62(long number)` | Static method, camelCase |
| Go (Golang) | `EncodeBase62(number int64) string` | PascalCase for exported function |
| C# | `EncodeBase62(long number)` | PascalCase |
| PHP | `encode_base62($number)` | snake_case or camelCase |
| Ruby | `encode_base62(number)` | snake_case |
| Swift | `func encodeBase62(_ number: Int) -> String` | camelCase, Swift style |
| Rust | `fn encode_base62(number: u64) -> String` | snake_case in Rust |
| Kotlin | `fun encodeBase62(number: Long): String` | camelCase, Kotlin style |

# How short is the short?

- Requirement
  - System should be able to generate **X** number of URLs per second
  - The URLs should be retained for **10** years.
    - **X** x 60 (per minute) x 60 (per hour) x 24 (per day) x 365 (per year) x 10 = **Y**

    - $62^1$ = **62**
    - $62^2$ = **3844**
    - $62^3$ = **238,328**
    - **……**
    - $62^6$ = **~ 56.8 billion**
    - $62^7$ = **~ 3.52 trillion**

$$62^n > Y$$

$$n = \log_{62} Y$$

# Our calculation

- Requirement
  - System should be able to generate **1000** number of URLs per second
  - The URLs should be retained for **10** years.

  - **Total URLs**
    - **1000** x 60 x 60 x 24 x 365 x 10 = **315,360,000,000 = ~ 315 billion = ~ 0.315 trillion**

  - **$62^6$ = ~ 56.8 billion (Can't be used)**
  - **$62^7$ = ~ 3.52 trillion (Let's go with this)**

  - **So we need minimum 7 characters for our URL**

# Another Approach (1)

- **Total URLs = ~ 315 billion = ~ 0.315 trillion**
- Base 62 encoding
  - a to z (26 characters) + A to Z (26 characters) + 0 to 9 (10 characters) = 62 characters
- Let’s increase usability
- Option 1
  - a to z (26 characters) + 0 to 9 (10 characters) = 36 characters
  - Base 36 encoding
  - **$36^7$ = ~ 78.36 billion**
  - **$36^8$ = ~ 2.82 trillion**

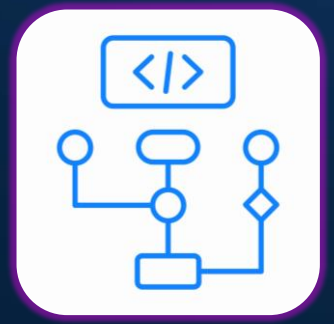
| Base | No. of characters |
|---|---|
| 62 | 7 |
| 36 | 8 |

# Another Approach (2)

- **Total URLs = ~ 315 billion = ~ 0.315 trillion**
- Option 2
  - a to z (26 characters) + 0 to 9 (10 characters) = 36 characters
  - Remove o (letter o) and 0 (number zero)
  - Remove l (letter l) and 1 (number one)
  - 36 - 4 = 32
  - Base 32 encoding
  - **$32^7$ = ~ 34.36 billion**
  - **$32^8$ = ~ 1.09 trillion**

| Base | No. of characters |
|---|---|
| 62 | 7 |
| 36 | 8 |
| 32 | 8 |

# Deep Dive

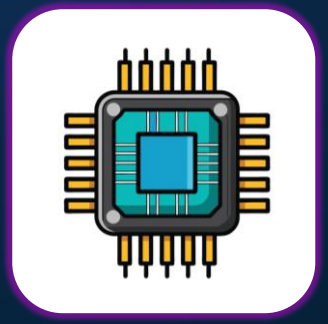


**Algorithm**

To create
Short URLs

**Redirection**

To redirect requests



**Compute**

To process
requests

**Storage**

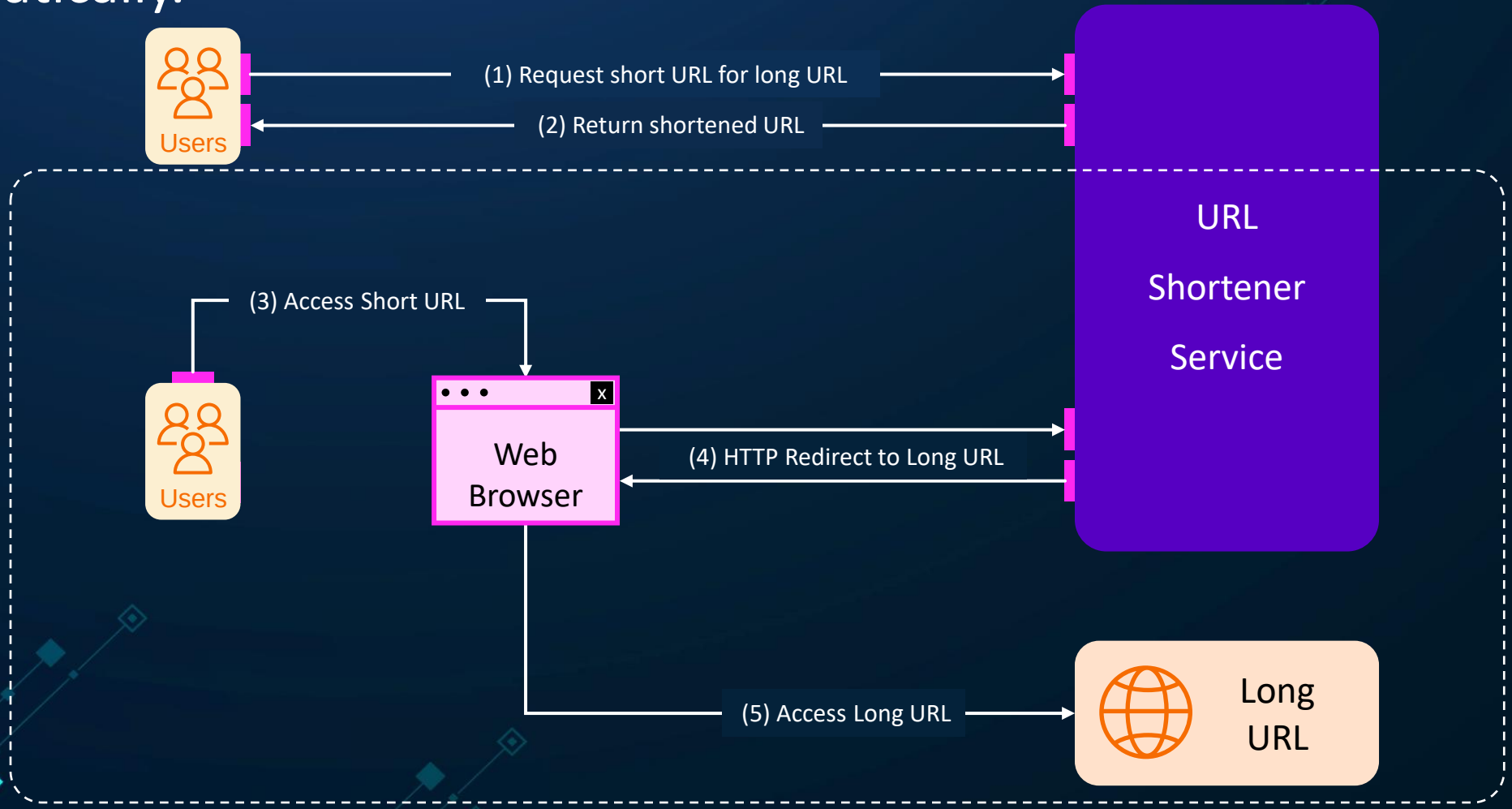
To store
URL mappings

**Analytics**

To generate
reports

# URL Redirection

- A technique used to send users (or search engines) from one URL to another automatically.



# Redirection


**Our new address**
A nice and cosy place,
On a scenic road,
Not too far,
In the same City

Royal Mail

**Redirection Service**

# Common Use Cases

- Moving a website to a new domain.
- Redirecting traffic from outdated URLs to updated ones.
- Handling URL typos or alternative spellings.
- Mobile/desktop site redirection.

# Types of URL redirection

1. 301 Redirect (Permanent)
   - Tells browsers that the page has moved permanently to a new location.

2. 302 Redirect (Temporary)
   - Tells browsers that the page has moved temporarily.

3. Meta Refresh
   - A client-side redirect that happens through a <meta> tag in the HTML.

4. JavaScript Redirect
   - Uses JavaScript (client side) to change the location.

Not suitable for our requirement

# Choosing between 301 and 302

| Feature | 301 Redirect (Permanent) | 302 Redirect (Temporary) |
|---|---|---|
| **Browser Cache** | Browsers cache the new location | Browsers will not cache the redirect |
| **User Experience** | Seamless transition to new page | Seamless, but user may end up back at the original page again |
| **When to Use** | • URL restructuring<br>Changing URL (/about-us → /about)<br>• Domain change<br>Moving example.com to newsite.com<br>• HTTP to HTTPS<br>Redirecting all HTTP traffic to HTTPS<br>• Content cleanup<br>Removing old pages and consolidating SEO to one page | • A/B testing<br>Temporarily redirecting users to test different page designs<br>• Site maintenance<br>Redirecting users while a page is being updated or fixed<br>• Device-specific routing<br>Redirecting mobile users to a temporary mobile version<br>• Staging environments<br>Directing traffic away from a page not ready for indexing |

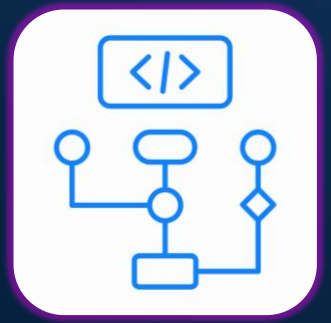
# What suits our requirement?

- We need to capture the traffic information.

- 301 ☒
  - Browsers will cache it and we won’t have correct traffic information

- 302 ☑
  - Browsers will NOT cache it so we will have correct traffic information
  - This allows flexibility in the future to change the destination URL without search engines caching or indexing the redirect as permanent.

# Deep Dive

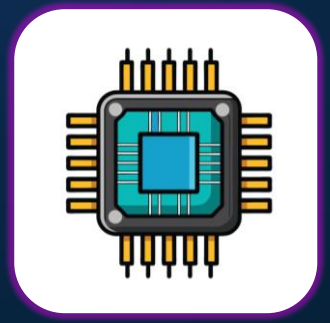


**Algorithm**

To create
Short URLs

**Redirection**

To redirect requests



**Compute**

To process
requests

**Storage**

To store
URL mappings

**Analytics**

To generate
reports

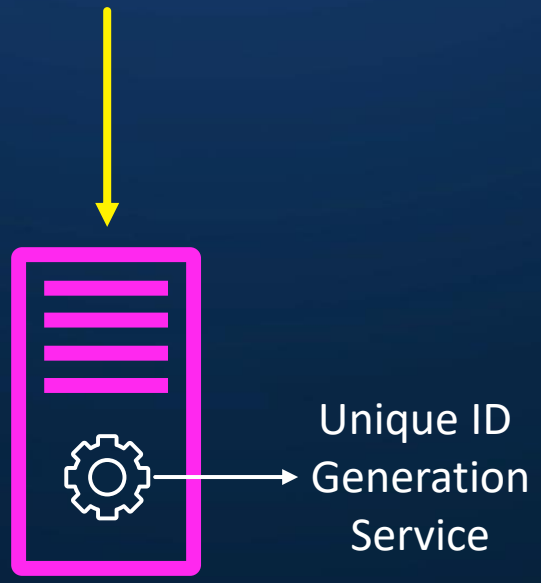
# Single node vs. multiple nodes

Single node

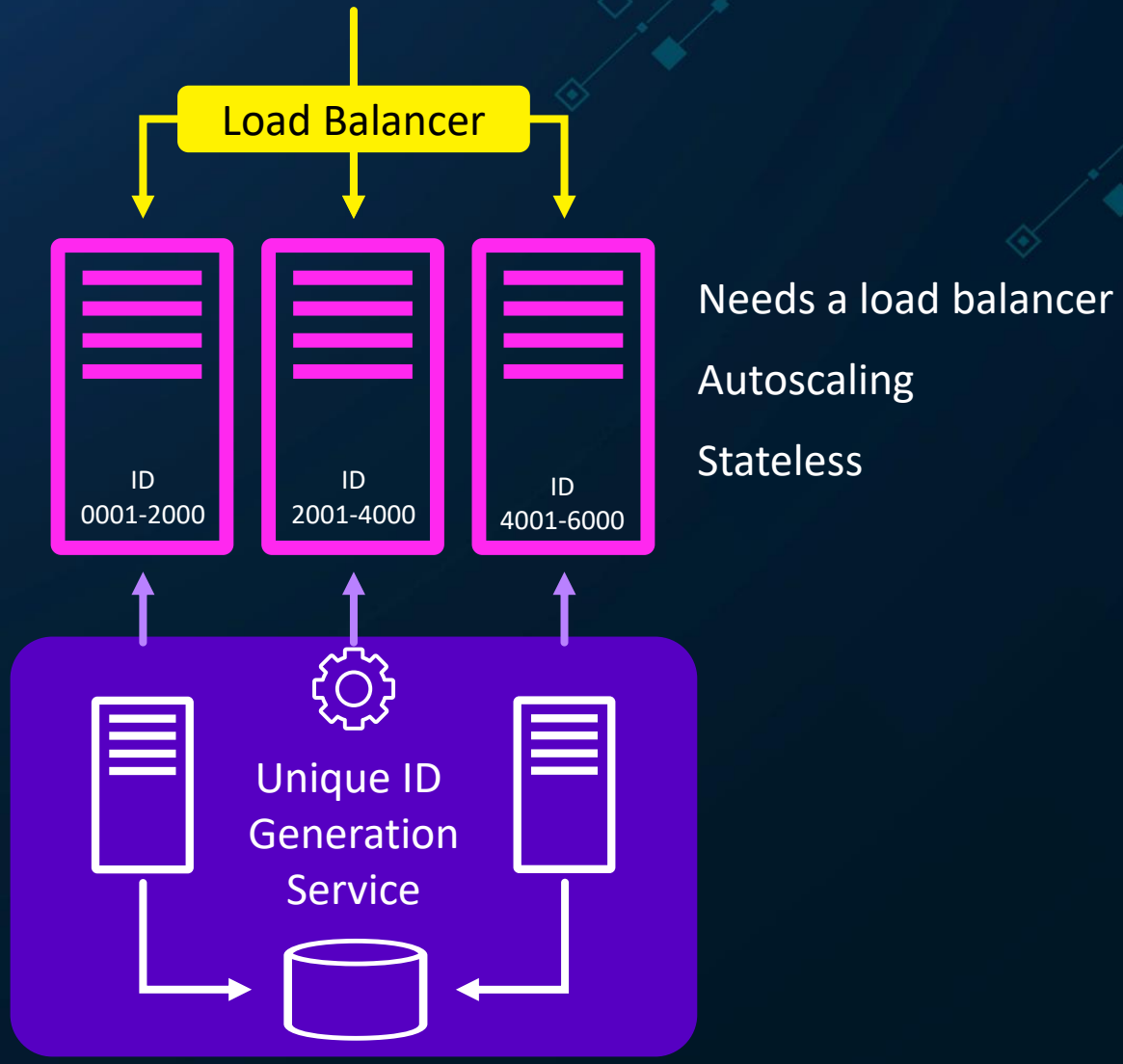
Single point of failure

Capacity issues

Stateful




Multiple nodes




Needs a load balancer

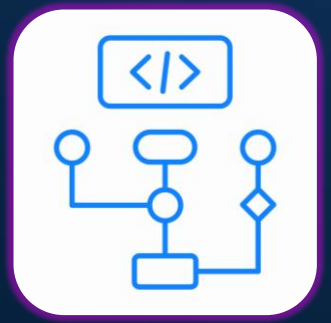
Autoscaling

Stateless

# Virtual Machine vs. Container

| Factor | Virtual Machines (VMs) | Containers |
|---|---|---|
| Isolation | Strong (full OS-level isolation) | Moderate (shares host OS kernel) |
| Resource Overhead | High (includes full OS) | Low (lightweight, minimal overhead) |
| Boot Time | Slow (seconds to minutes) | Fast (milliseconds to seconds) |
| Security | Better for untrusted/multi-tenant apps | Requires hardening; better for trusted apps |
| Portability | Limited (OS dependent) | High (build once, run anywhere) |
| Use Case Fit | Legacy, monolithic, stateful apps | Microservices, cloud-native, stateless apps |
| Scalability | Slower (provisioning takes time) | Fast, dynamic scaling |
| Persistent Storage | Easy with built-in disks | Requires volumes, StatefulSets, etc. |
| Management Tools | Mature (e.g., VMware, Hyper-V) | Modern (e.g., Docker, Kubernetes) |
| Learning Curve | Less steep | Steeper (esp. with orchestration) |
| Cost Efficiency | Higher per workload due to overhead | More efficient at scale |

# Deep Dive

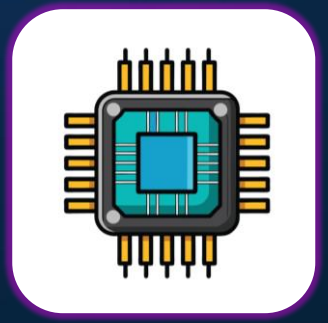


**Algorithm**

To create
Short URLs

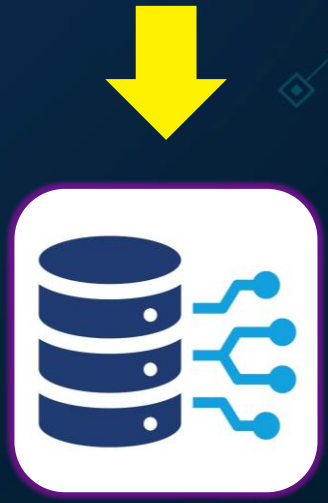
**Redirection**

To redirect requests



**Compute**

To process
requests



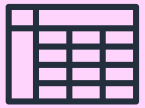
**Storage**

To store
URL mappings

**Analytics**

To generate
reports

# Do we need a database or a blob storage?

| Factor |  Blob Storage  |  Database Storage  |
|---|---|---|
| Data Type | Unstructured (e.g., images, videos) | Structured (tables, rows, columns) |
| Query Capability | Limited (via metadata or external index) | Advanced (SQL/NoSQL queries) |
| Use Case Fit | Media, backups, logs, large files | Transactional apps, analytics, reporting |
| Performance | Optimized for large file storage | Optimized for fast reads/writes and queries |

# Storage capacity

- Storage required per URL:

| | |
|---|---|
| Short URL (7 Characters) | 7 bytes |
| Average long URL (100 characters) | 100 bytes |
| Expiration date (long) | 8 bytes |
| Average metadata | 1000 bytes |
| **Total** | **1115 bytes** |
| **Rounded off** | **1024 bytes = 1 KB** |

- Storage requirement for 1 year
  - **Total URLs** (based on 1000 URLs shortened per second)
    - **1000** x 60 x 60 x 24 x 365 = **31,536,000,000 = ~ 31.5 billion**
  - **Required storage**
    - **31.53 billion × 1 KB = ~ 32 TB**

# Relations or Non-relational database?

- Both database types can support our storage capacity requirement.

- A few observations about the nature of the data we will store:
  - We need to store billions of records.
  - Each object we store is small (~1K).
  - We don’t need complex queries and transactions.

| Type | Pros. | Cons. |
|---|---|---|
| Relational DBs | Efficient searching | Difficult to scale |
| Non-relational DBs | Easier to scale | Eventual consistency |

# Proprietary or Open-Source DB?

- Quick Guidelines:

  - Use Proprietary DB when:
    - You're in a regulated industry, need enterprise support, or must scale reliably with minimal internal DBA effort.
      - Example - Amazon DynamoDB, Azure Cosmos DB

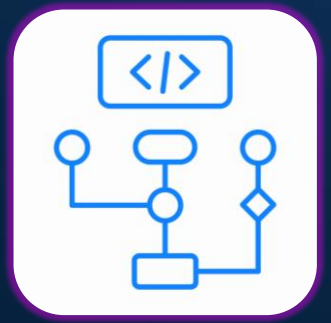
  - Use Open-Source DB when:
    - You're optimizing for cost, want customization, or prefer flexibility and community-driven innovation.
      - Example - MongoDB, Apache Cassandra

# Amazon DynamoDB

- Why?
  - Non-relational DB
  - Fully managed service
  - Scalability
  - Specific suitable features:
    - Configurable TTL for object expiration:
      - Can be used for URL expiry
    - DynamoDB Stream:
      - Can be used for analytics
    - Global Table:
      - Can be used if we need to expand to multiple geographies

Amazon DynamoDB

# Deep Dive

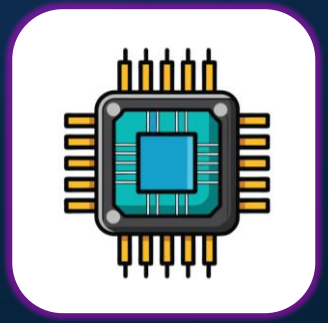


**Algorithm**

To create
Short URLs

**Redirection**

To redirect requests



**Compute**

To process
requests

**Storage**

To store
URL mappings

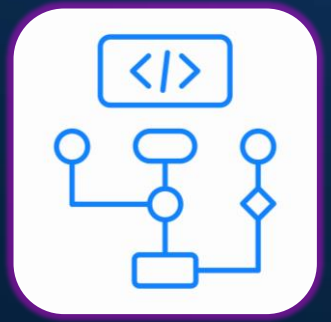
**Analytics**

To generate
reports

# Real-time or batch analytics?

| Feature | Real-time analytics | Batch analytics |
|---|---|---|
| Data Processing | Continuous / Streaming | Periodic / Scheduled |
| Latency | Low (seconds or milliseconds) | High (minutes to hours) |
| Use Cases | Fraud detection, live monitoring, alerts | Reporting, trend analysis, historical data |
| Data Size | Smaller chunks, processed immediately | Large volumes, processed in bulk |
| Infrastructure | Requires low-latency, stream processing | Suited for distributed batch systems |
| Complexity | More complex, needs real-time pipelines | Simpler to implement and manage |
| Examples | Kafka + Spark Streaming, Amazon Kinesis | Hadoop, AWS Glue, Amazon EMR |
| Cost | Potentially higher (always-on systems) | Generally lower (scheduled jobs) |

# Deep Dive

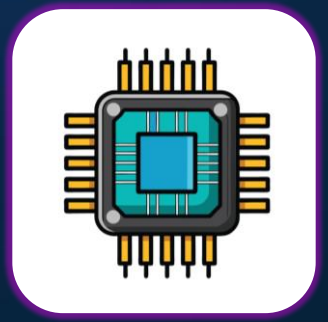


**Algorithm**

To create
Short URLs

**Redirection**

To redirect requests



**Compute**

To process
requests

**Storage**

To store
URL mappings

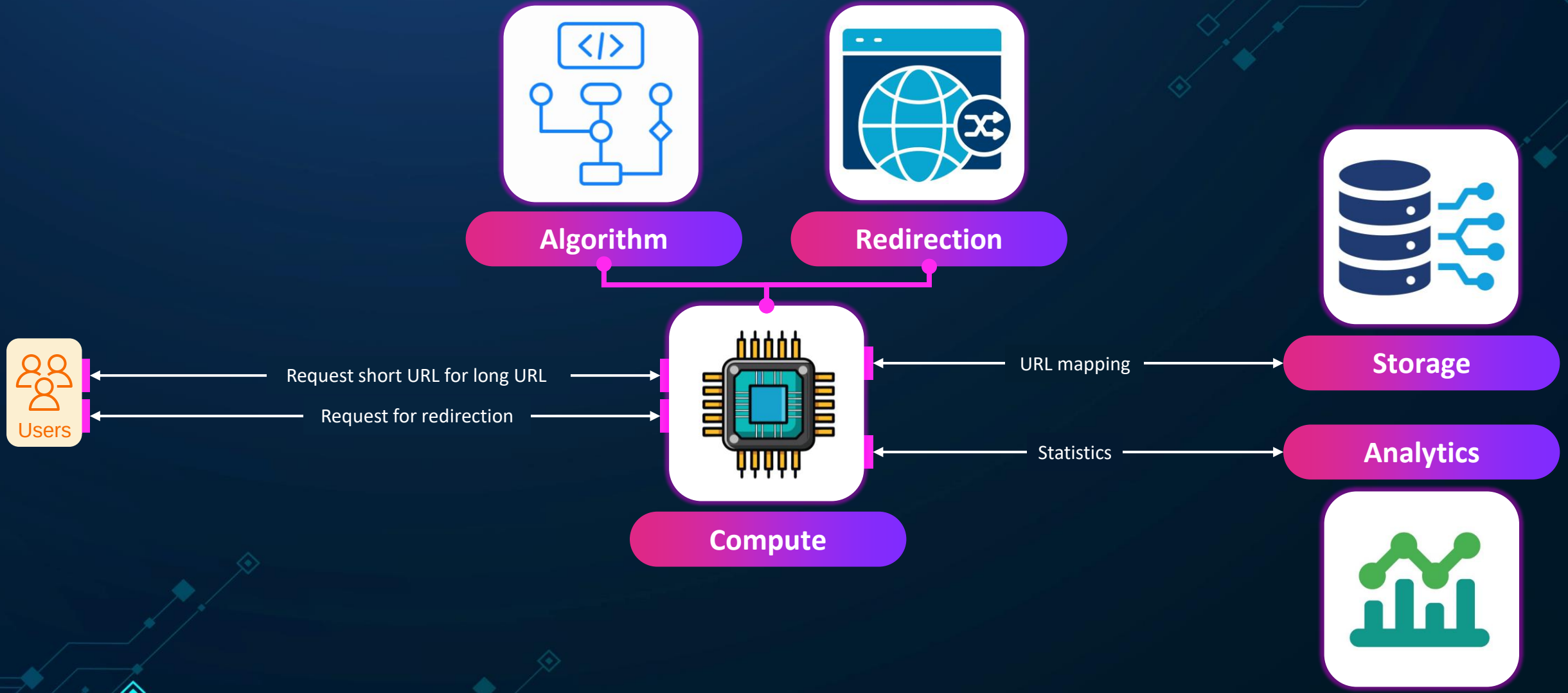
**Analytics**

To generate
reports

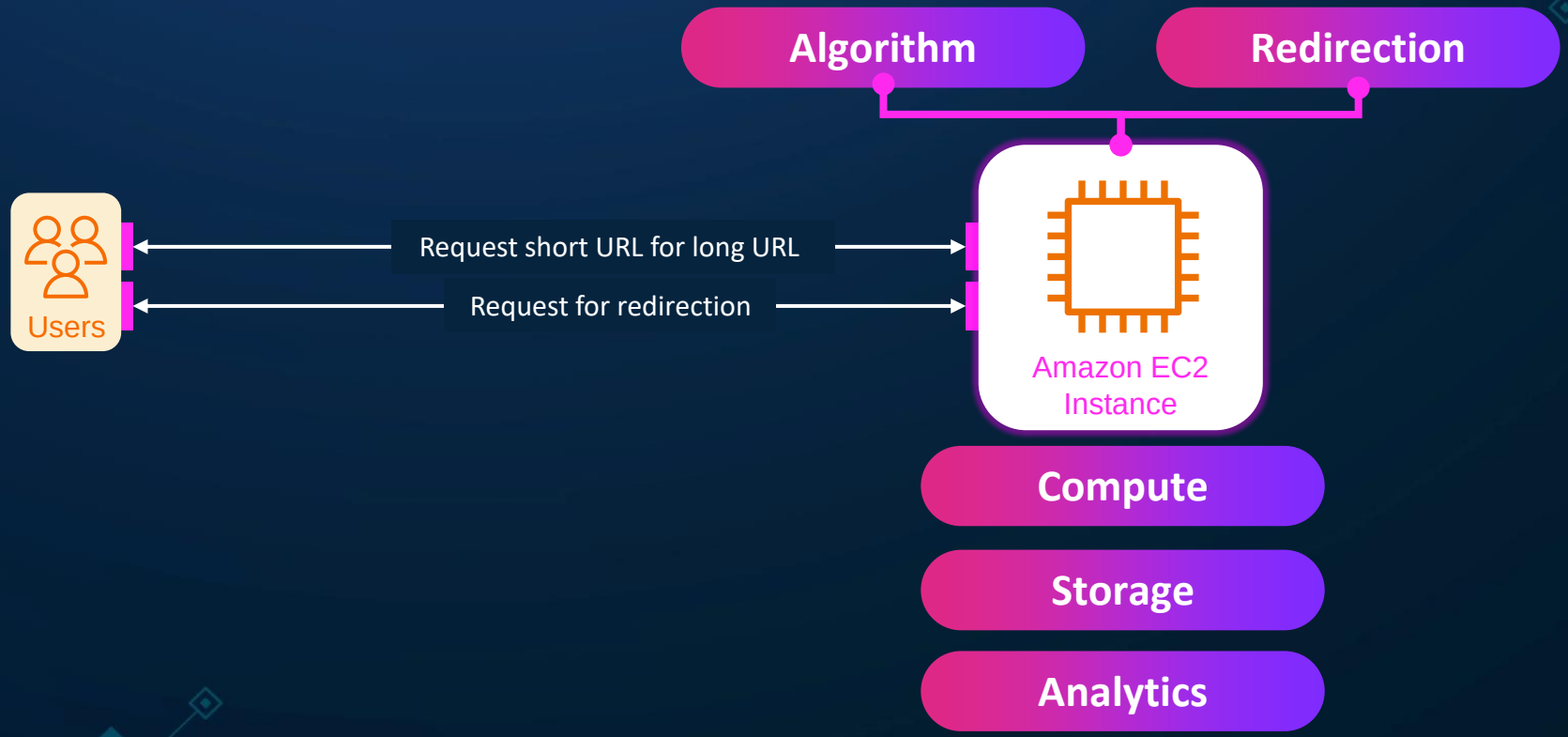
# Day Zero Architecture

- Make It Work, Make It Right, Make It Fast.
- Focus on launching the product quickly without overplanning
- Prioritize familiar technologies to avoid steep learning curves
- Avoid experimenting with untested or complex new tools early on
- Aim for minimal time to market to deliver value fast

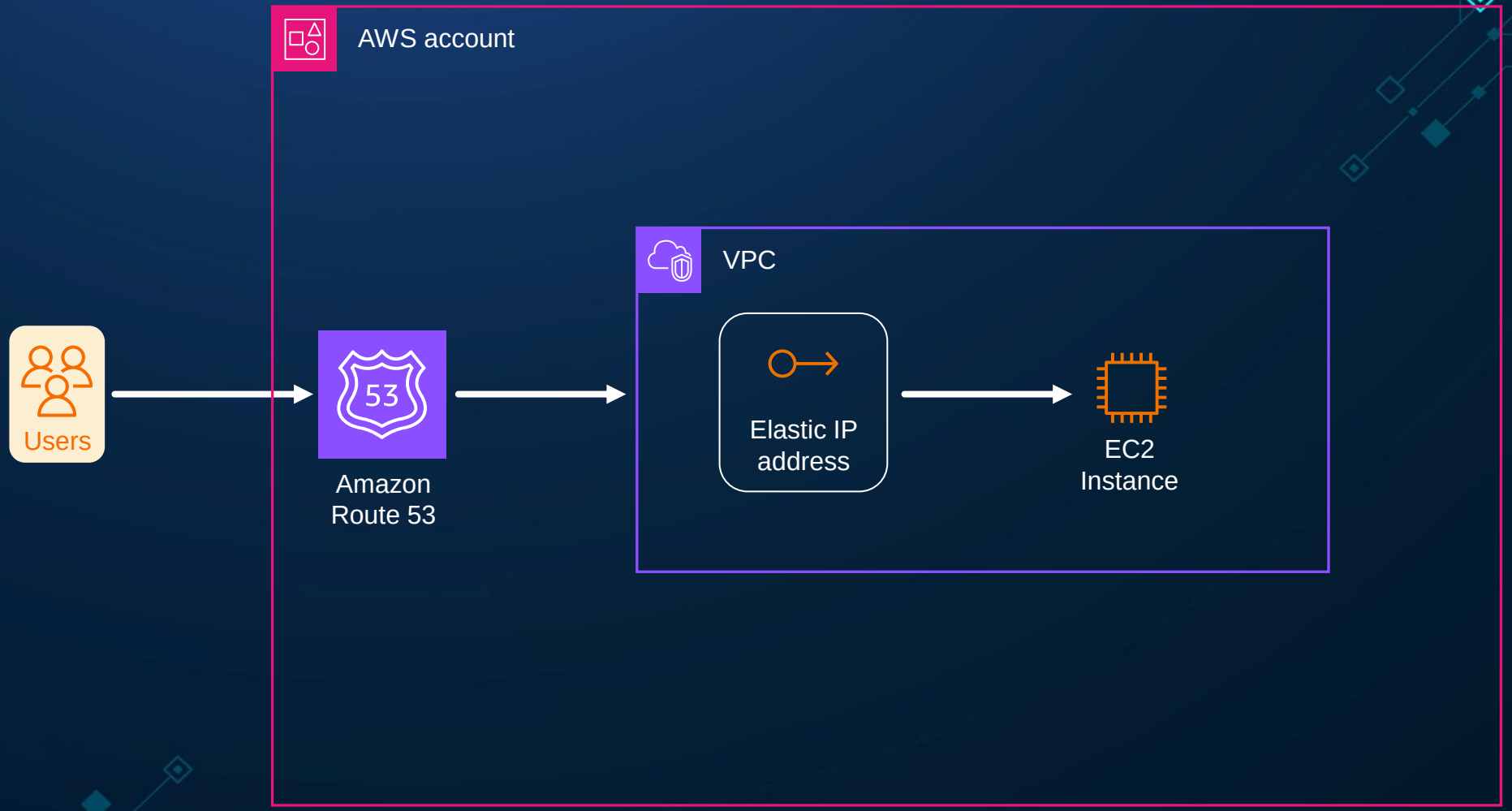
# High-level architecture



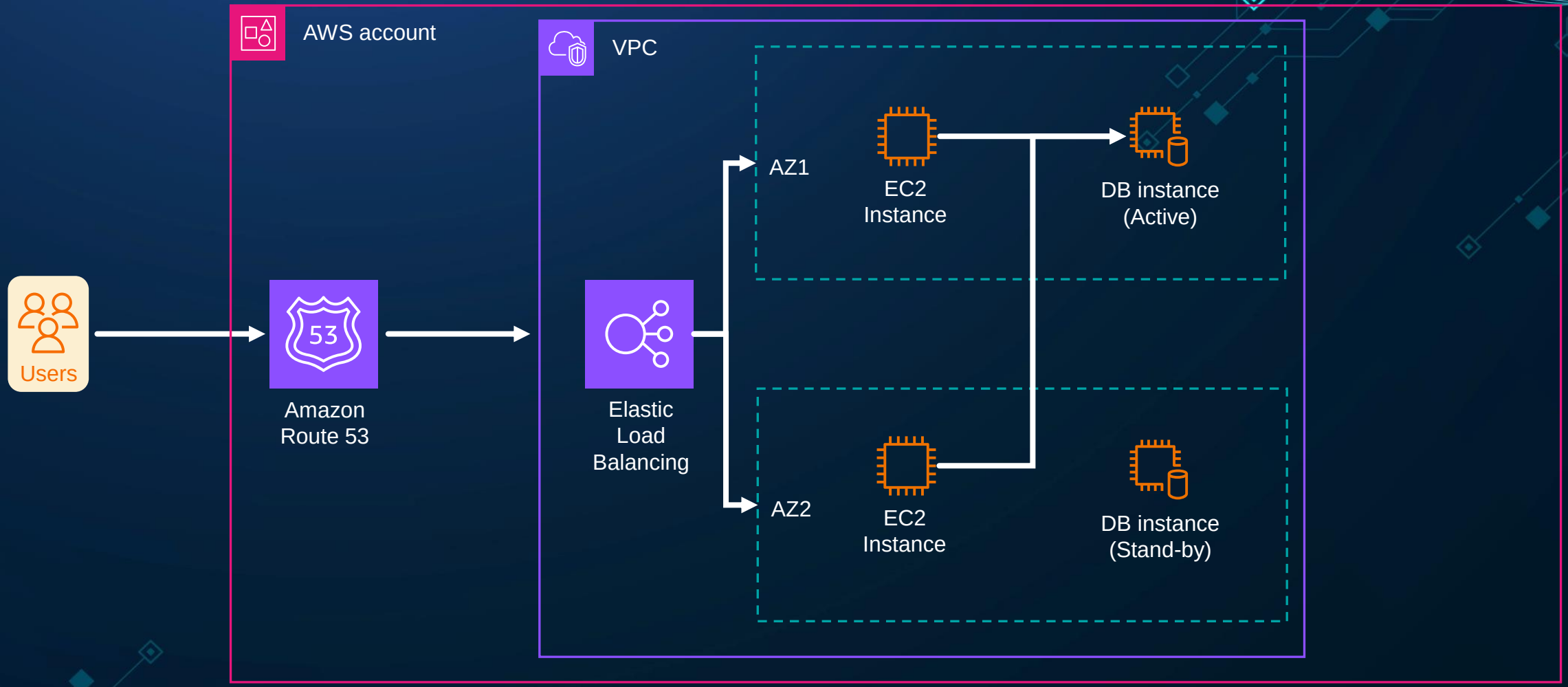
# A monolith


Algorithm
Redirection
Users
Request short URL for long URL
Request for redirection
Amazon EC2 Instance
Compute
Storage
Analytics

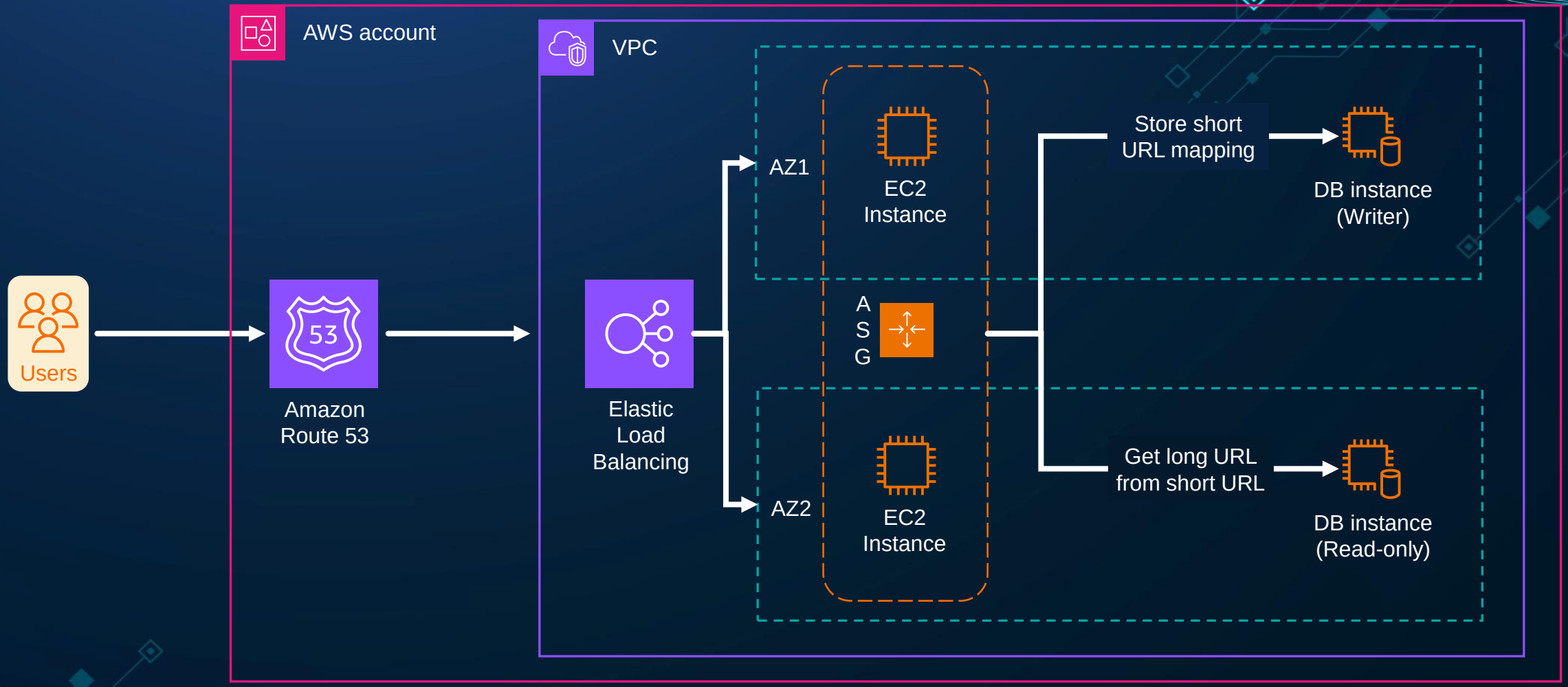
# A monolith


AWS account
VPC
Users
53
Amazon
Route 53
Elastic IP
address
EC2
Instance

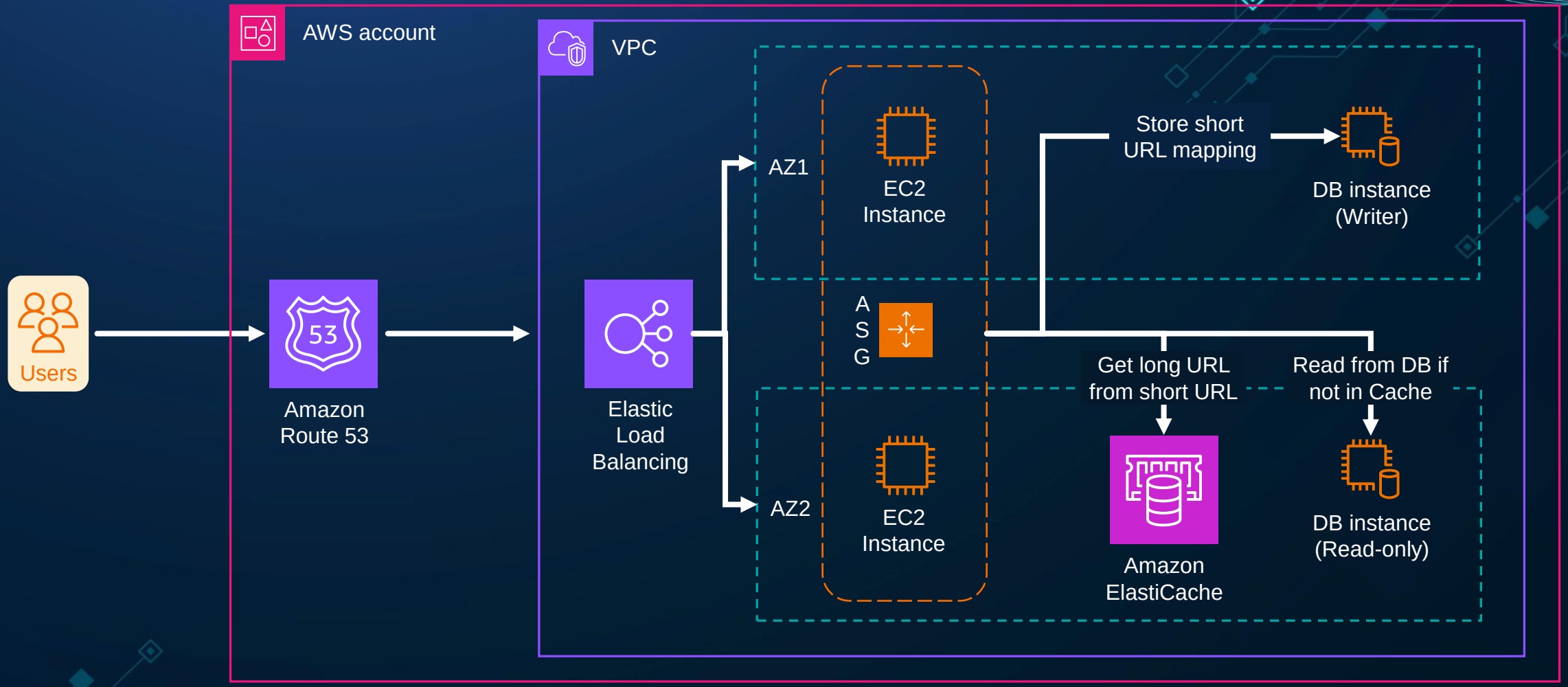
# Improved architecture



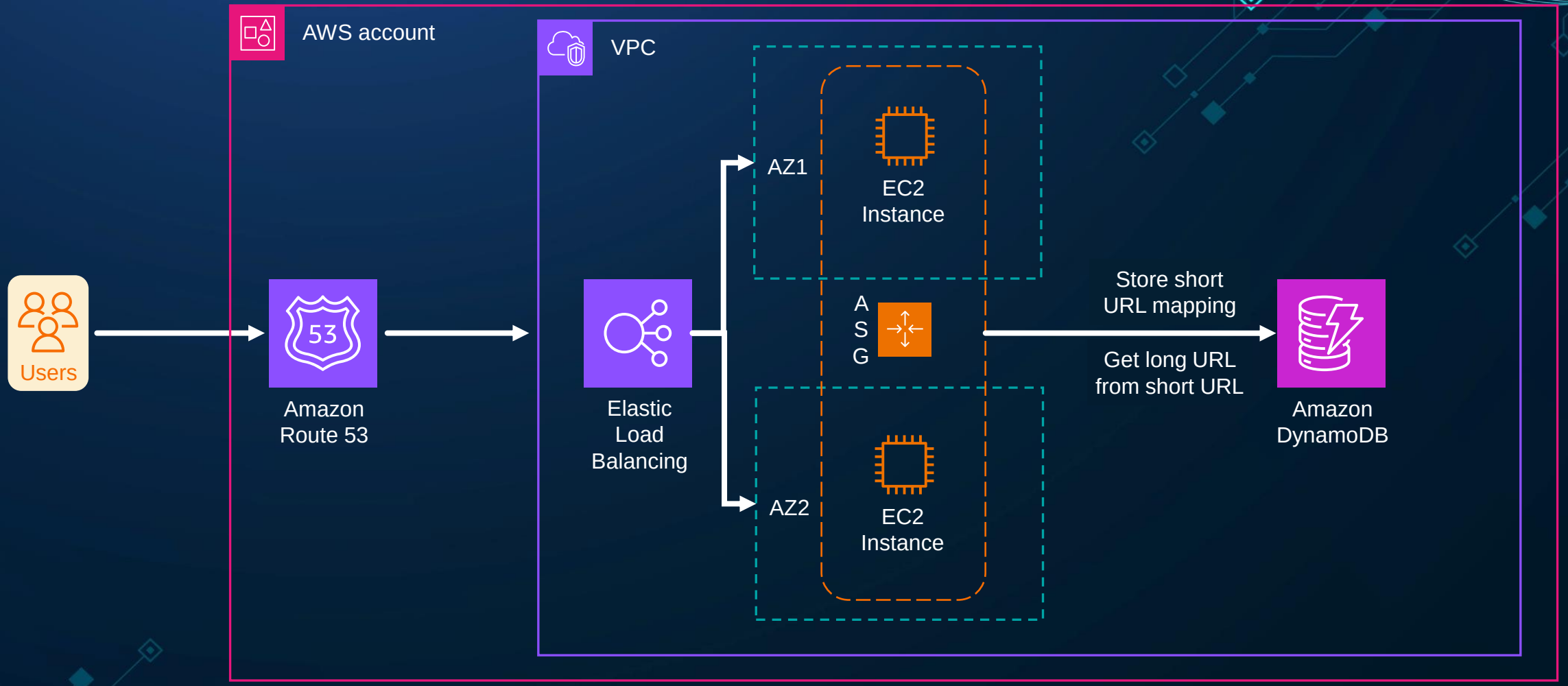
# Improved architecture



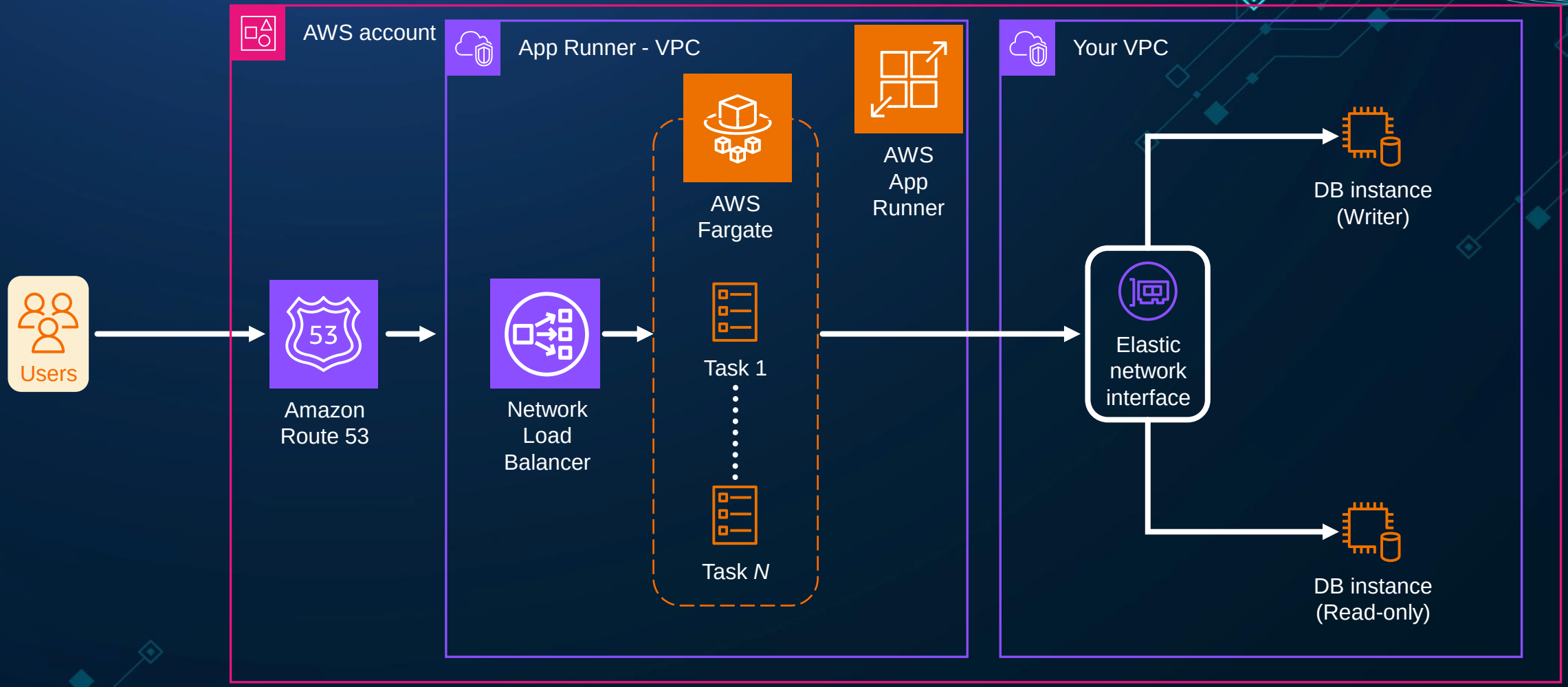
# Improved architecture



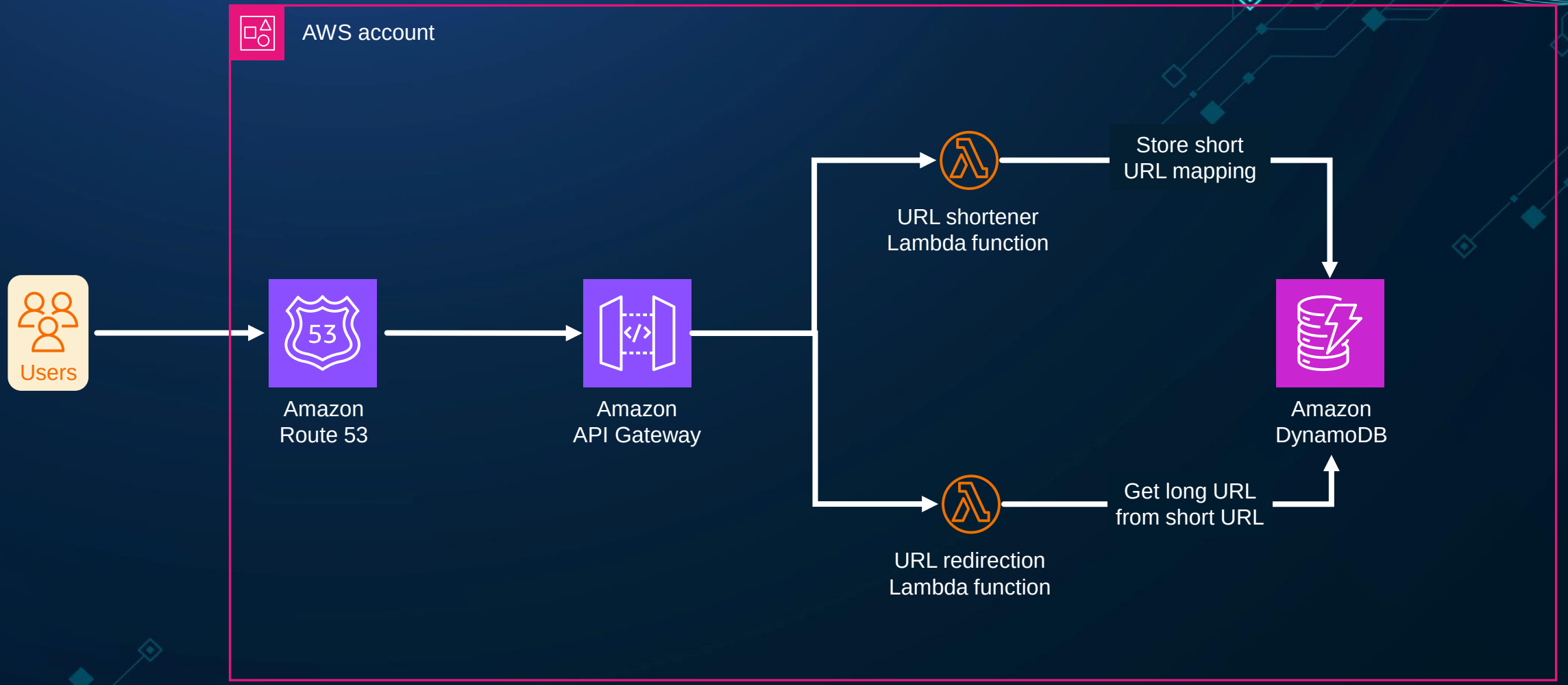
# Improved architecture



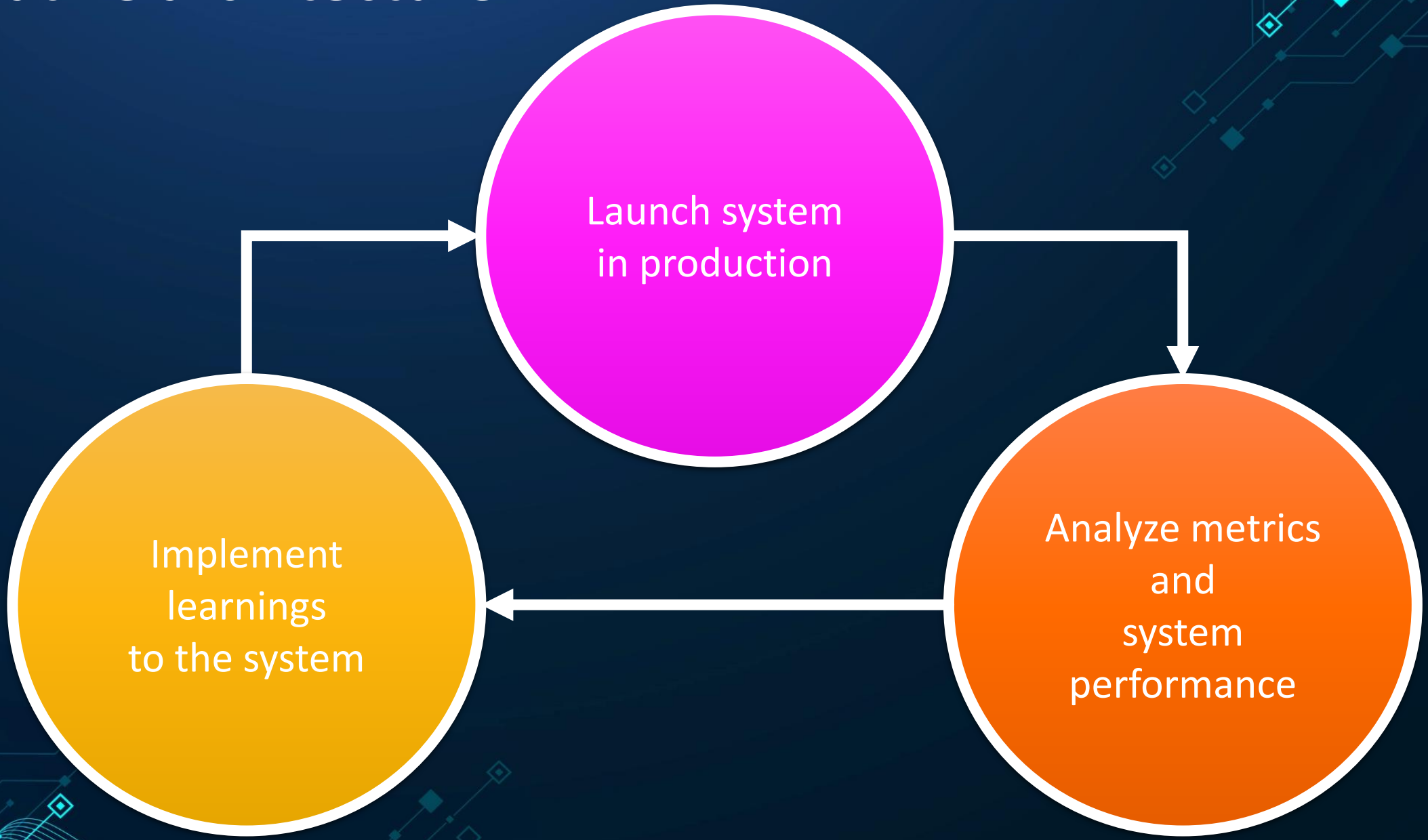
# Using containers



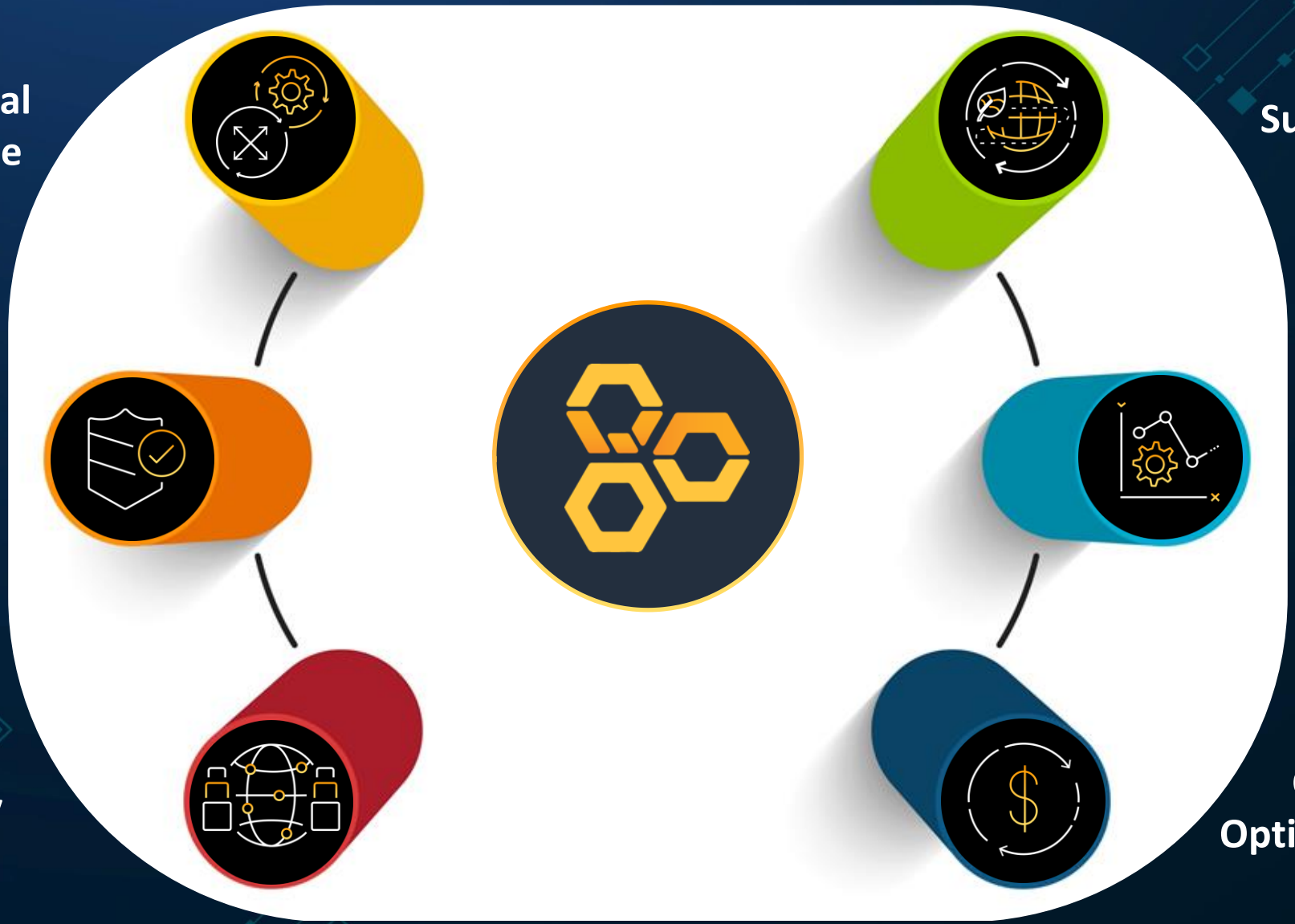
# Serverless architecture



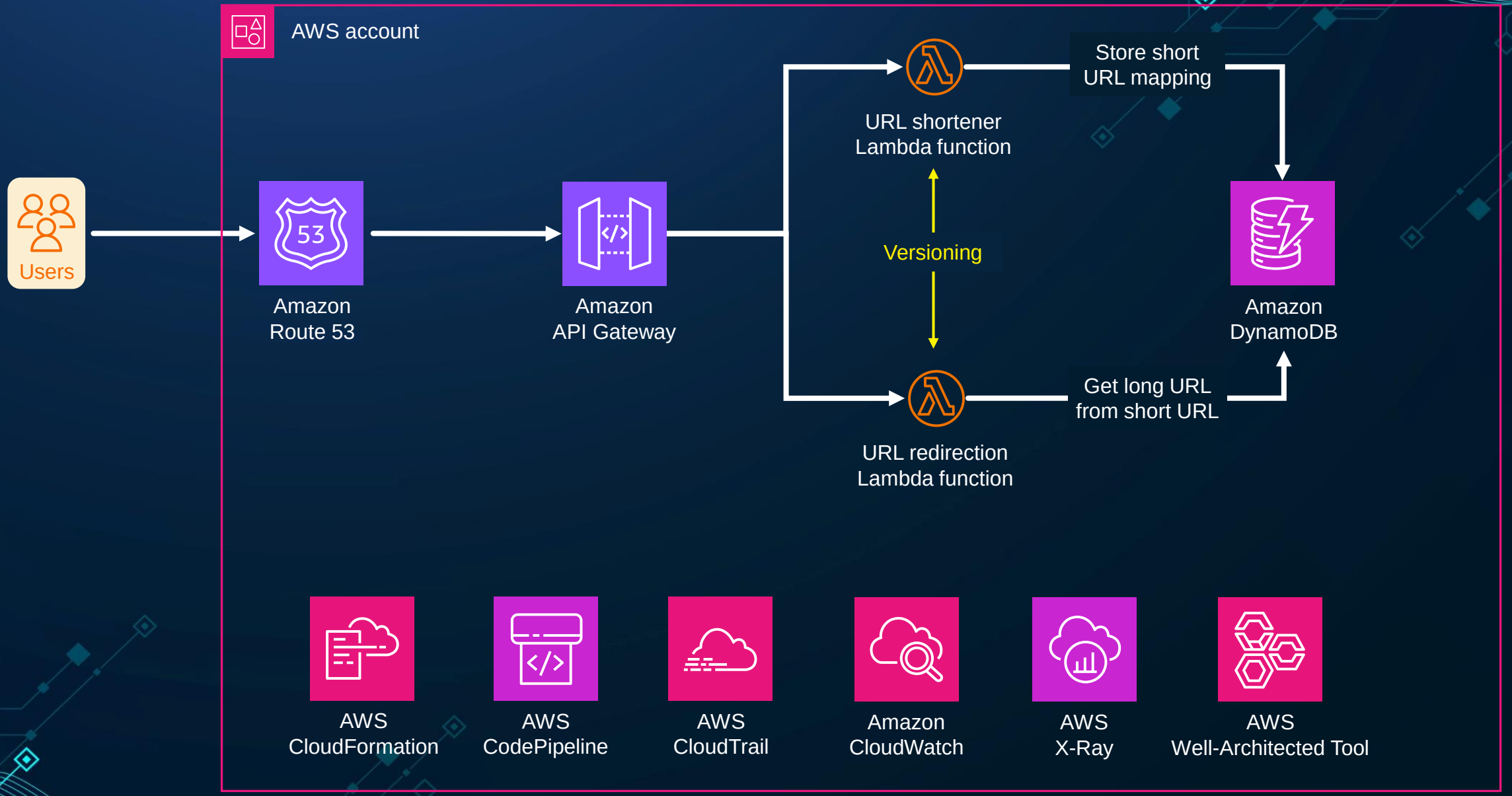
# Iterative architecture



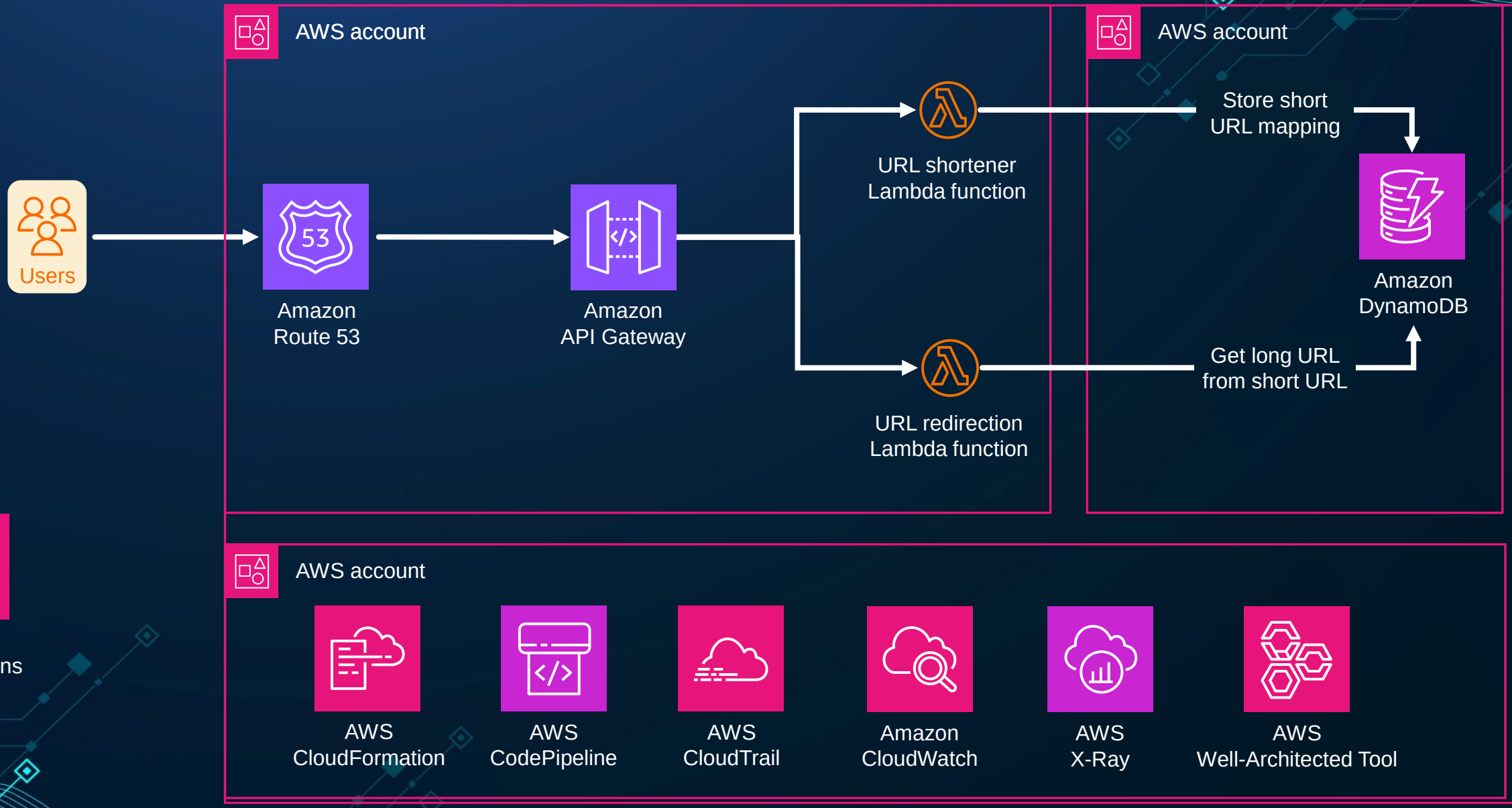
# Well-architected framework



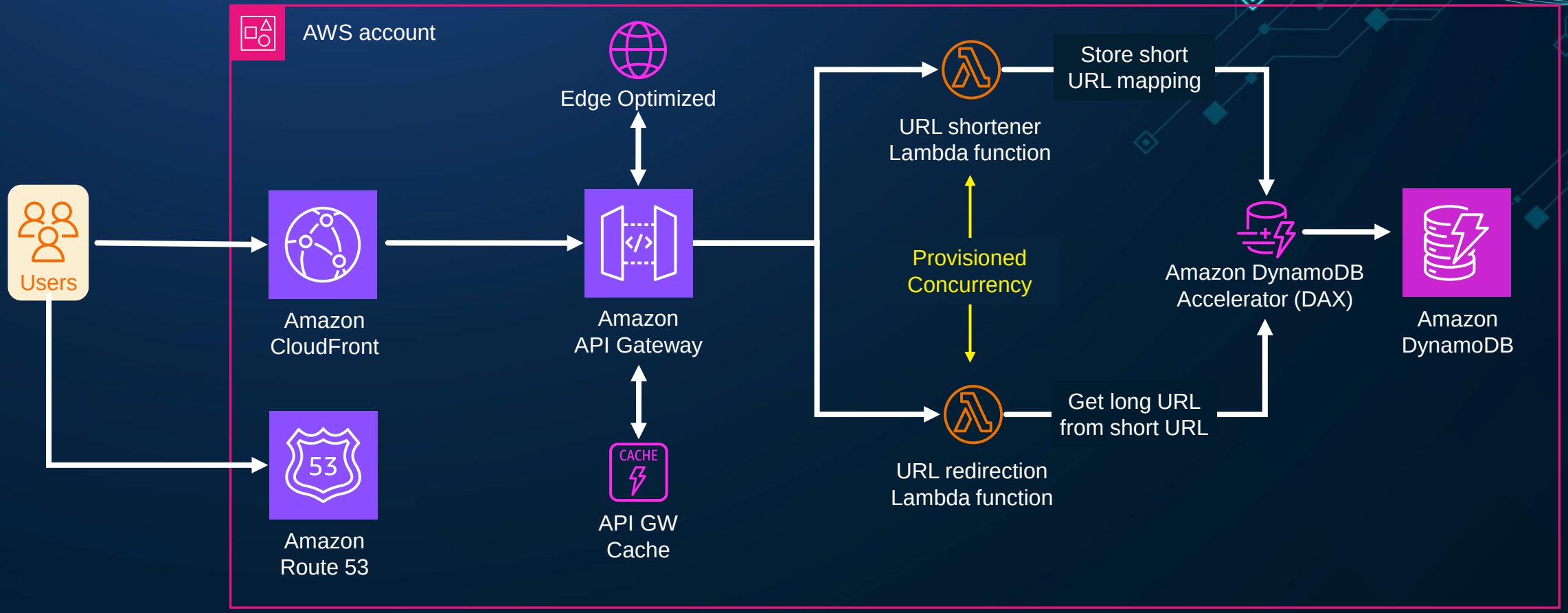
# Operational Excellence



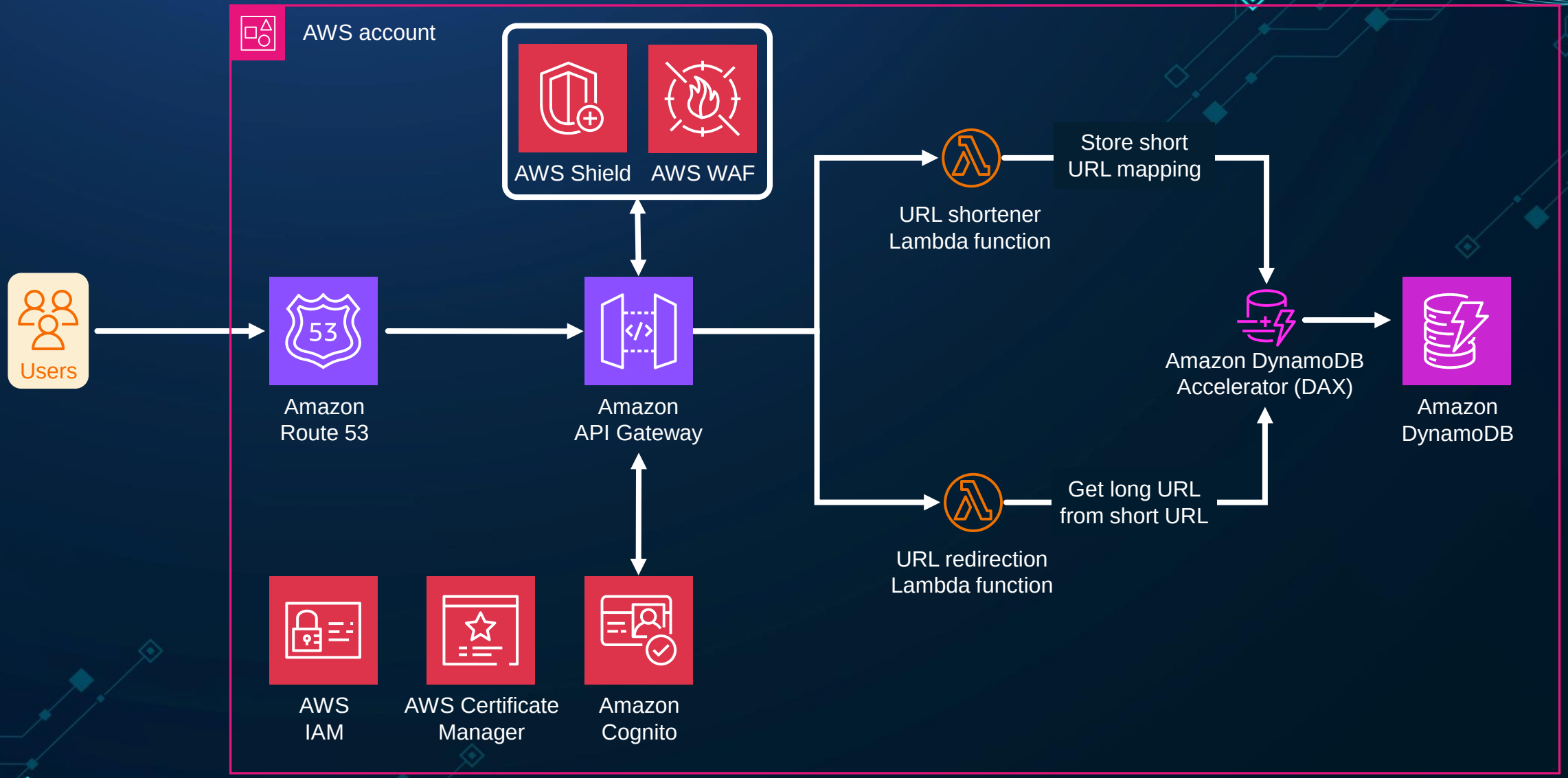
# Operational Excellence



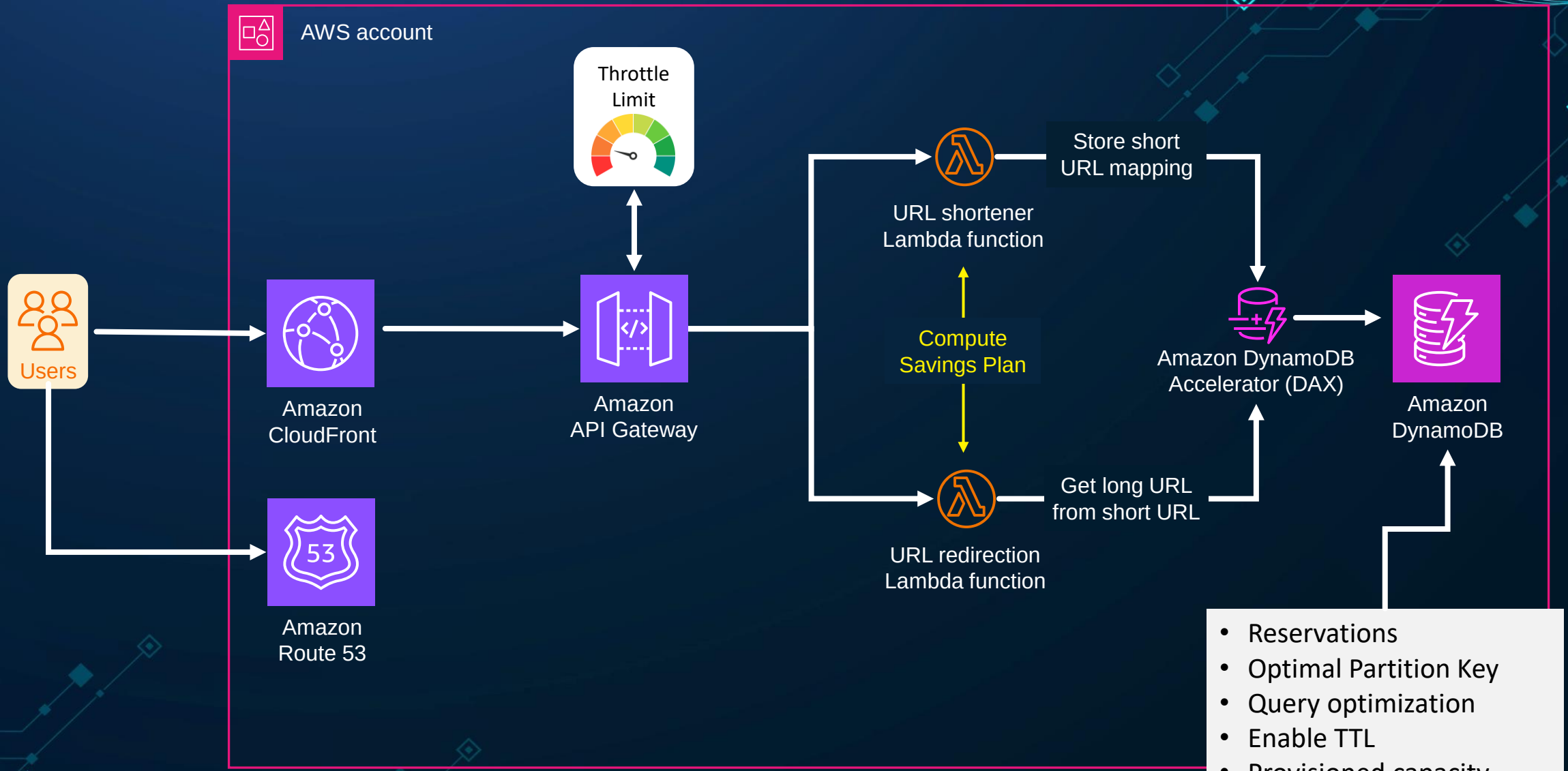
# Performance Efficiency



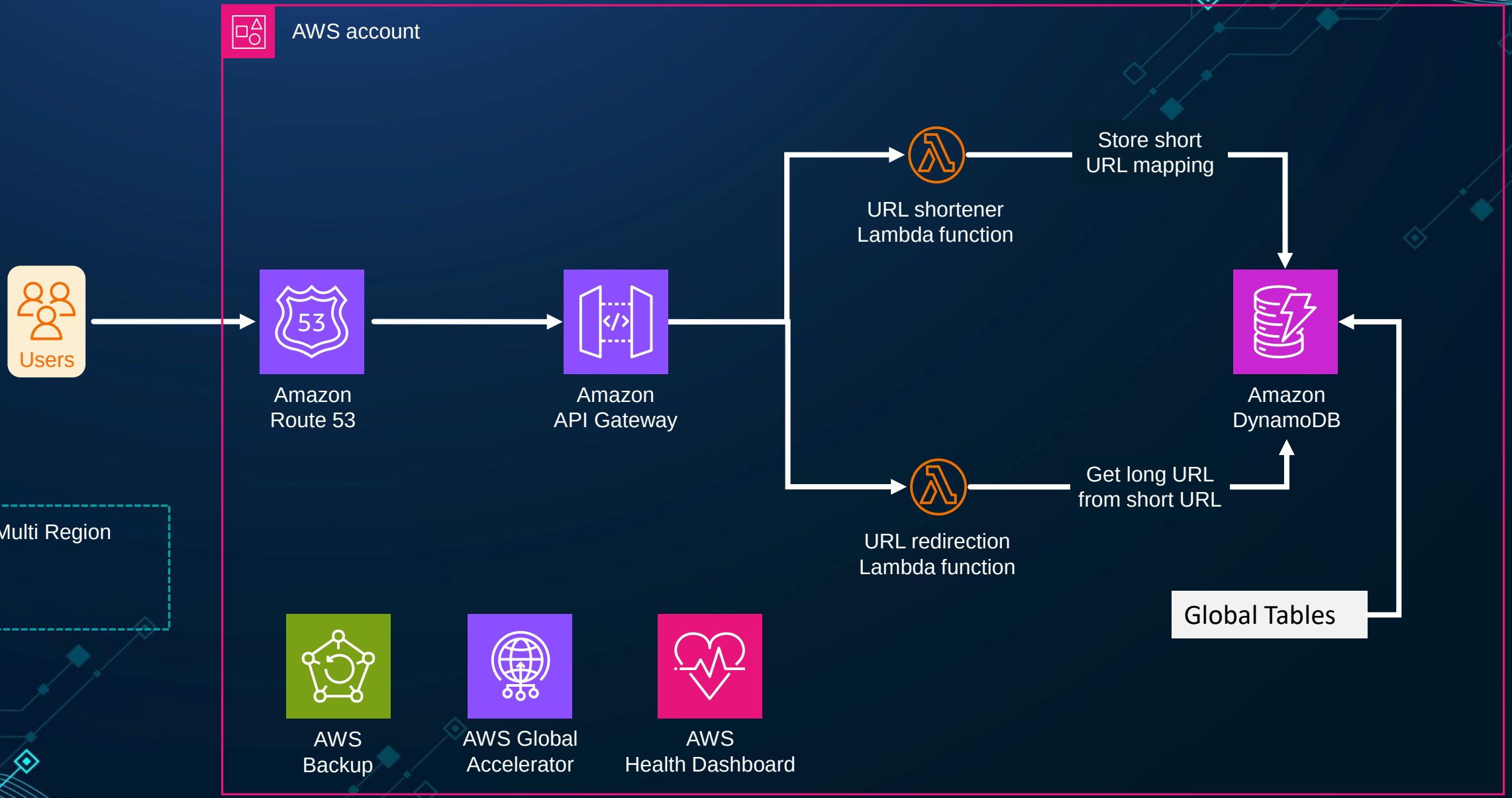
# Security


AWS account
AWS Shield
AWS WAF
Users
Amazon Route 53
Amazon API Gateway
URL shortener Lambda function
Store short URL mapping
Amazon DynamoDB Accelerator (DAX)
Amazon DynamoDB
URL redirection Lambda function
Get long URL from short URL
AWS IAM
AWS Certificate Manager
Amazon Cognito

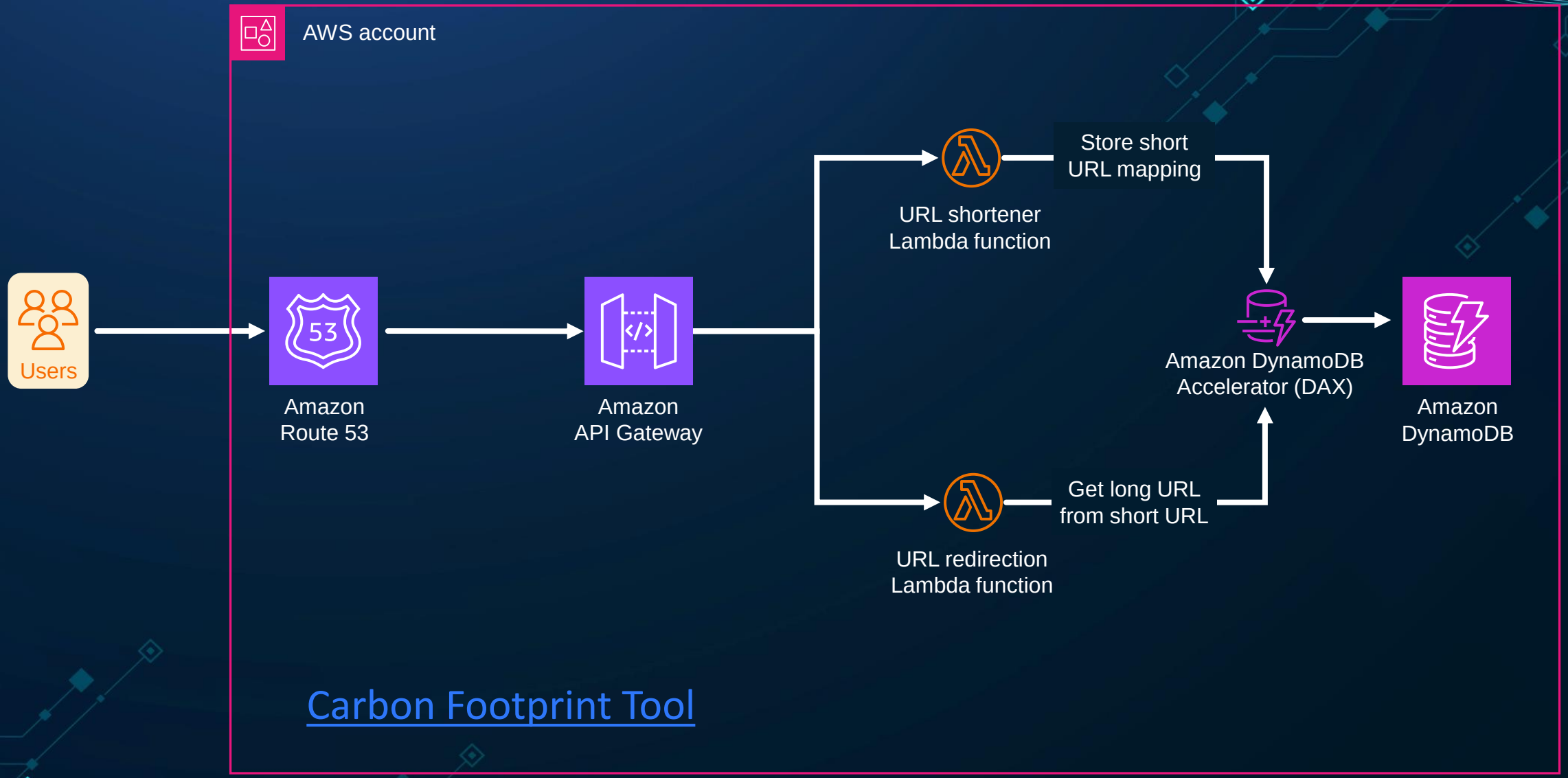
# Cost Optimization



# Reliability

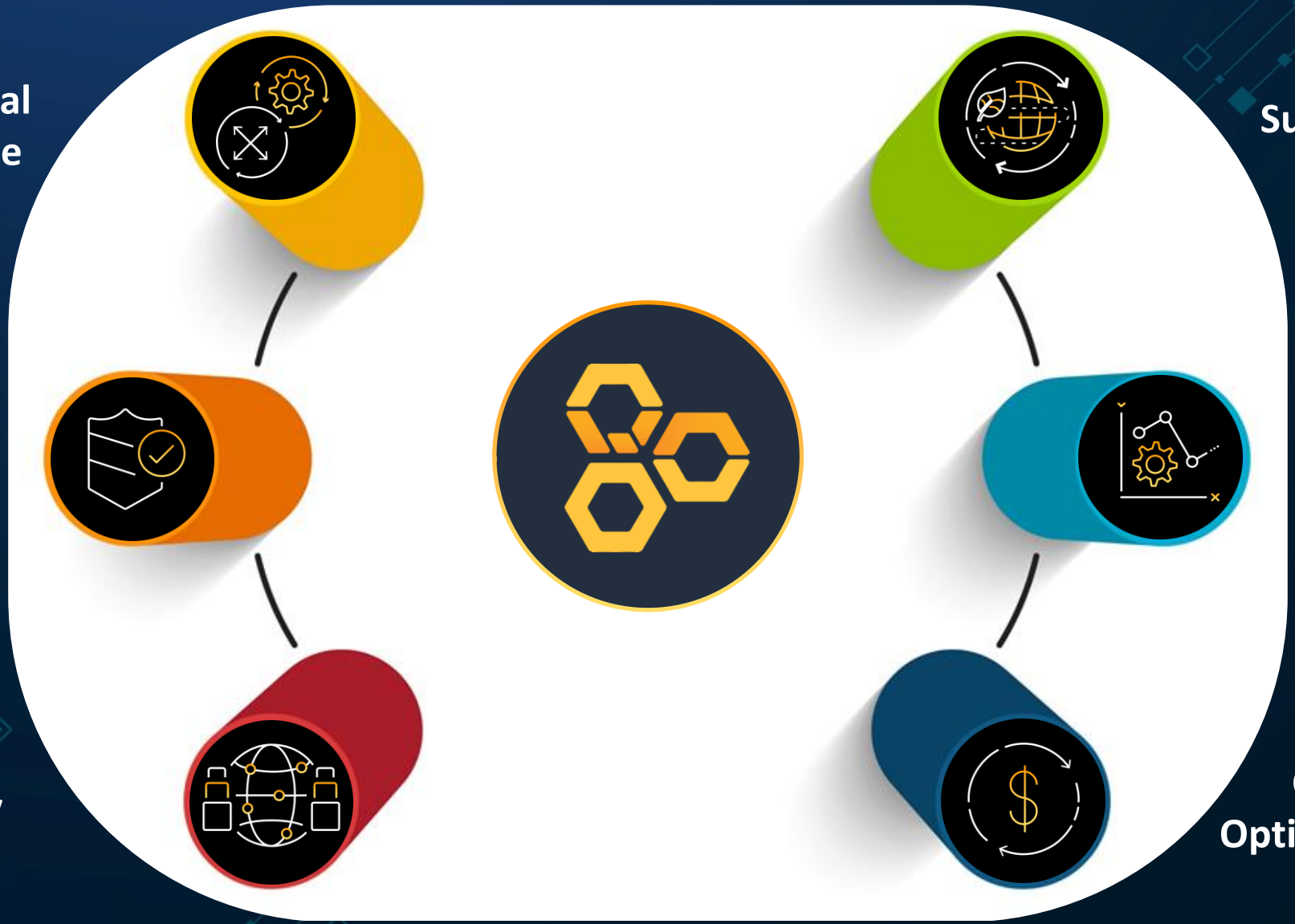


# Sustainability




[Carbon Footprint Tool]

# Well-architected framework



# Commercial Aspect

- Revenue Generation
  - Free User
  - Paid Users
    - URL Expiry
    - Detailed Statistics
    - Custom URL
    - Update functionality
    - Custom Domain Name
    - API Exposure
    - Bulk Creation

# Other considerations

Efficient Table Design

Cache Eviction Policy

Analytics Pipeline

API Details

And many more…