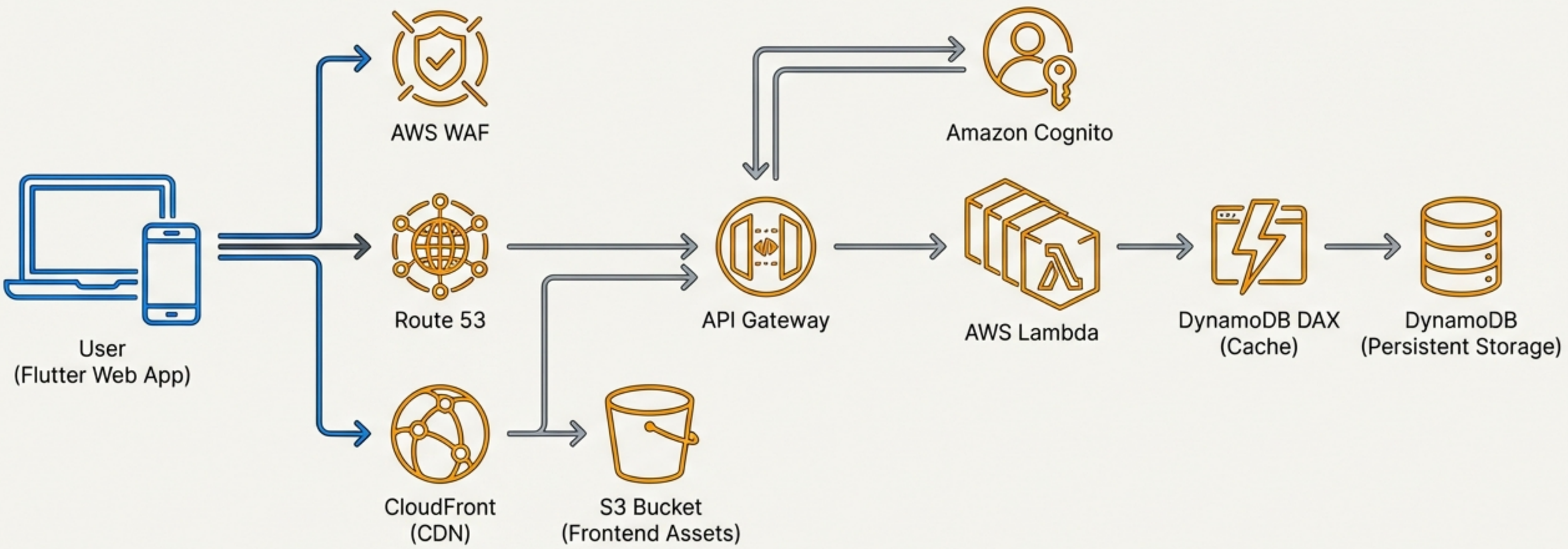
An Architectural Blueprint: Building a Full-Stack Serverless Application with Flutter and AWS

A deep dive into the design, implementation, and operation of a scalable, multi-region architecture.

The Complete System Architecture at a Glance



This presentation will deconstruct this serverless architecture layer by layer, starting from the user experience and moving down to the foundational infrastructure. This diagram will be our map throughout the journey.

A Core Philosophy: Develop Locally, Deploy Globally

Debug Mode (`isDebugMode = true`)

The entire Flutter application can be developed and tested without any AWS setup.

- All API calls return instant mock data.
- No network requests are made.
- No AWS credentials or costs are incurred during UI development.

✅ You can develop the entire app without any AWS setup!

Production Mode (`isDebugMode = false`)

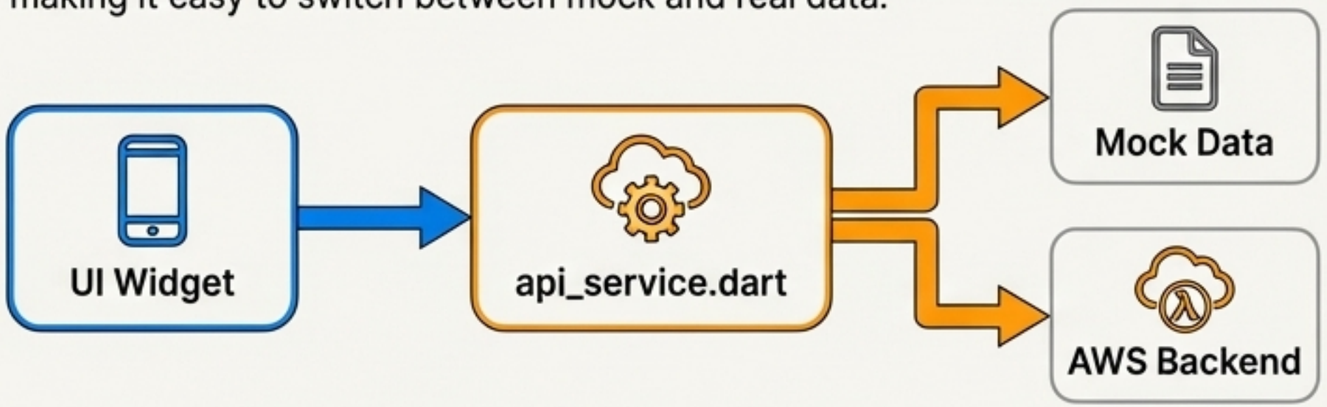
A single boolean flip connects the application to the live AWS backend.

- Real HTTP requests are sent to AWS Lambda via API Gateway.
- Authentication is handled by a live Amazon Cognito User Pool.
- Data is read from and written to DynamoDB.

The Experience Layer: The Flutter Frontend

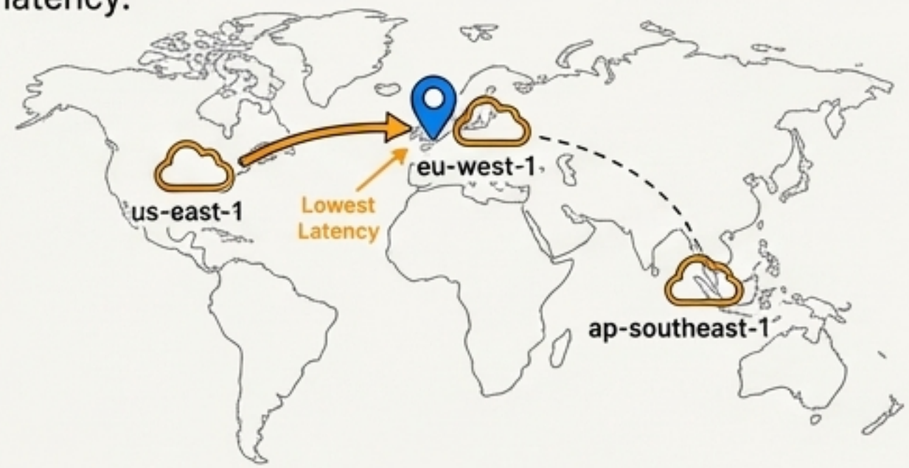
Centralised API Logic

A dedicated `api_service.dart` layer abstracts all backend communication, making it easy to switch between mock and real data.



Global Performance

The app dynamically detects the user's region to connect to the nearest AWS endpoint, minimising latency.



Building a Responsive UI

Optimistic UI patterns are used for critical actions like URL creation. The interface updates instantly, while the network request completes in the background.

create_url_screen.dart

```
// Immediately add the new URL to the local state
ref.read(urlListProvider.notifier).add(newUri);

// Then, send the request to the backend
await apiService.createUrl(newUrl);
```

UI updates instantly before the network call completes.

Analytics

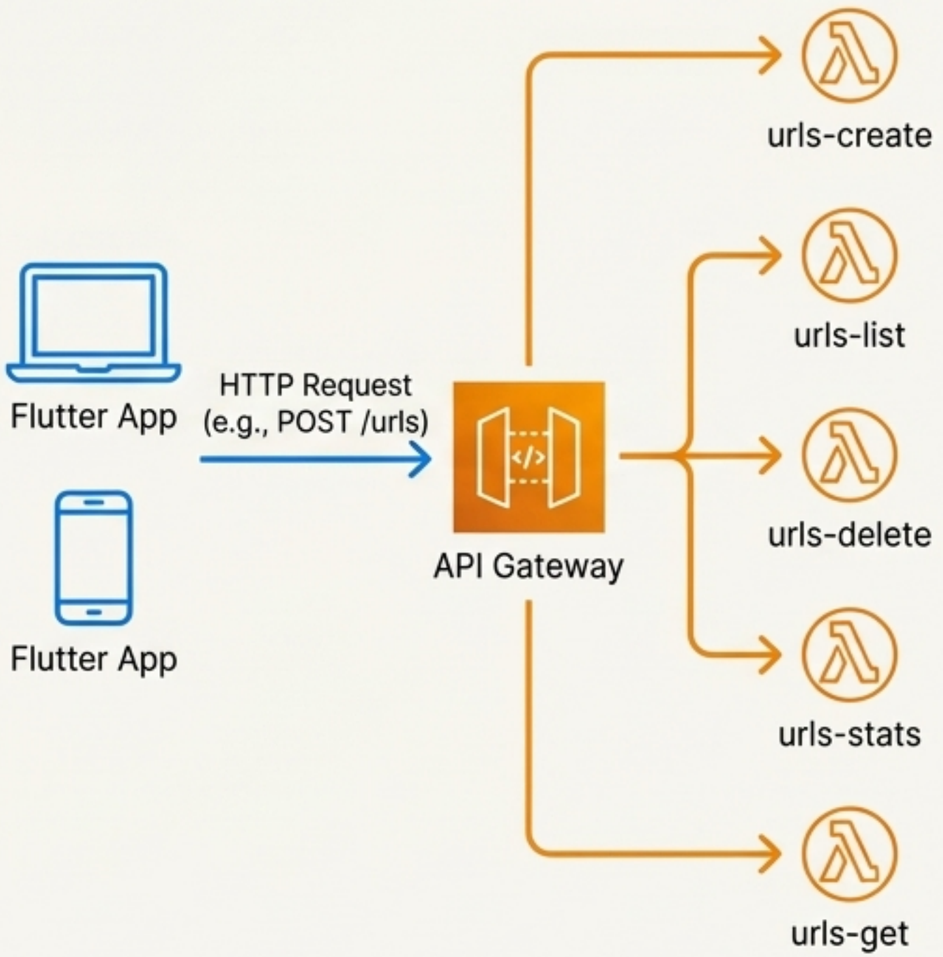
User events are tracked using AWS Pinpoint to provide insights into application usage.

The Logic Layer: API Gateway as the Central Router

Defined API Endpoints

```
GET /urls?userId=...
POST /urls
DELETE /urls/{id}
GET /stats/{id}
GET /{id}

POST /urls?userId=...
GET /stats/{id}
GET /{id}
```



Key Configurations

CORS

Proper CORS headers are configured to allow requests from the web application.

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Content-Type,Author
Access-Control-Allow-Methods: OPTIONS,POST,GET,DE
```

Security

API endpoints are secured using a Cognito JWT authoriser and rate limiting is configured to prevent abuse.

Integration

Each method is mapped to a specific Lambda function to execute the business logic.

The Brains of the Operation: Node.js Lambda Functions

A single, powerful Lambda function (`url-shortener-logic`) handles all core URL management operations. The logic is event-driven, routing based on the HTTP method and path.

Routing

Internal routing logic dispatches requests based on the event's method and path.

Performance

Parallel database calls for maximum speed when loading user dashboards.

```
export const handler = async (event) => {
  const method = event.requestContext?.http?.method;
  const path = event.rawPath;

  // ...
  if (method === "GET" && path.endsWith("/urls")) {
    const historyCommand = new QueryCommand({...});
    const statsCommand = new GetCommand({...});

    const [historyResult, statsResult] = await Promise.all([
      docClient.send(historyCommand),
      docClient.send(statsCommand)
    ]);
    // ...
  }

  if (method === "GET") {
    // ...
    const updates = [
      docClient.send(new UpdateCommand({
        TableName: TABLE_URLS, Key: { id: shortCode },
        UpdateExpression: "ADD clicks :inc",
        ExpressionAttributeValues: { ":inc": 1 }
      })),
      // ...
    ];
    // We do NOT await them before returning the redirect...
    Promise.all(updates);
    return { statusCode: 301, ... };
  }
};
```

Data Integrity

Atomic counters ensure analytics data is always consistent, even under high load.

Efficiency

Analytics updates are executed in parallel and are not awaited before returning the redirect, ensuring the user experiences near-instant redirection.

Blueprint vs. Reality: The `url-shortener-logic` Function

The Blueprint

Code & Logic

```
if (method === "POST") {
  const { originalUrl, userId } = body;
  const shortCode = randomUUID().substring(0, 6);

  const item = {
    id: shortCode,
    originalUrl,
    userId: userId || "anonymous",
    clicks: 0,
    createdAt: new Date().toISOString(),
  };

  await docClient.send(new PutCommand({
    TableName: TABLE_URLS,
    Item: item
  }));

  // ... update user stats
}
```

The Reality

AWS Console View

url-shortener-logic

Function ARN →	....:function:url-shortener-logic
Runtime →	Node.js 20.x
Handler →	index.handler
Package size →	2.5 kB

The Foundation: Secure Identity Management with Cognito

Amazon Cognito provides a fully managed user directory, handling all aspects of authentication, from sign-up and sign-in to MFA and social federation.

User pool information

User Pool ID	us-east-1_OJORVuNml
Estimated Users	4
ARN	arn:aws:cognito-idp:us-east-1:...

MFA
Improve security with SMS, email, and authenticator-app multi-factor authentication.

Social Sign-In
Bring in users with credentials from Google, Facebook, Apple, and more.

Passwordless
Support passwordless sign-in with one-time codes.

Branding
Customise the look and feel of managed login pages.

The Data Core: A Purpose-Built DynamoDB Schema

The data model uses three distinct DynamoDB tables, each optimised for a specific access pattern, ensuring high performance and scalability.” in Inter

Table: `UrlShortener`

Stores the core URL data.

Schema:

Partition Key: `id` (String)

Key Feature:

GSI: `UserHistoryIndex`

GSI Keys:

Partition Key: `userId`

Sort Key: `createdAt`

GSI Purpose:
Enables efficient querying of all URLs created by a specific user, sorted by creation date.

Table: `DailyStats`

Tracks daily click analytics for graphs.

Schema:

Partition Key: `shortCode` (String)

Sort Key: `date` (String)

Allows for fast retrieval of time-series data for a given short URL.

Table: `User`

Stores aggregate user statistics.

Schema:

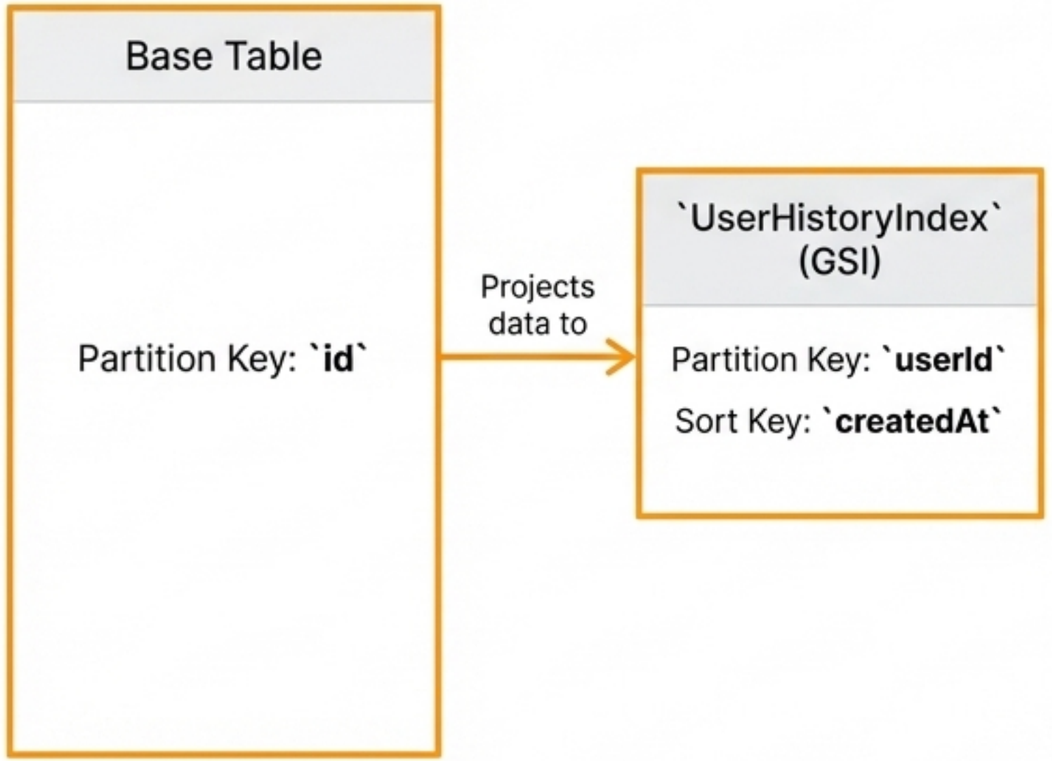
Partition Key: `userId` (String)

Holds atomic counters like `totalLinks` and `totalClicks` for instant dashboard stats.

Blueprint vs. Reality: The `UrlShortener` DynamoDB Table

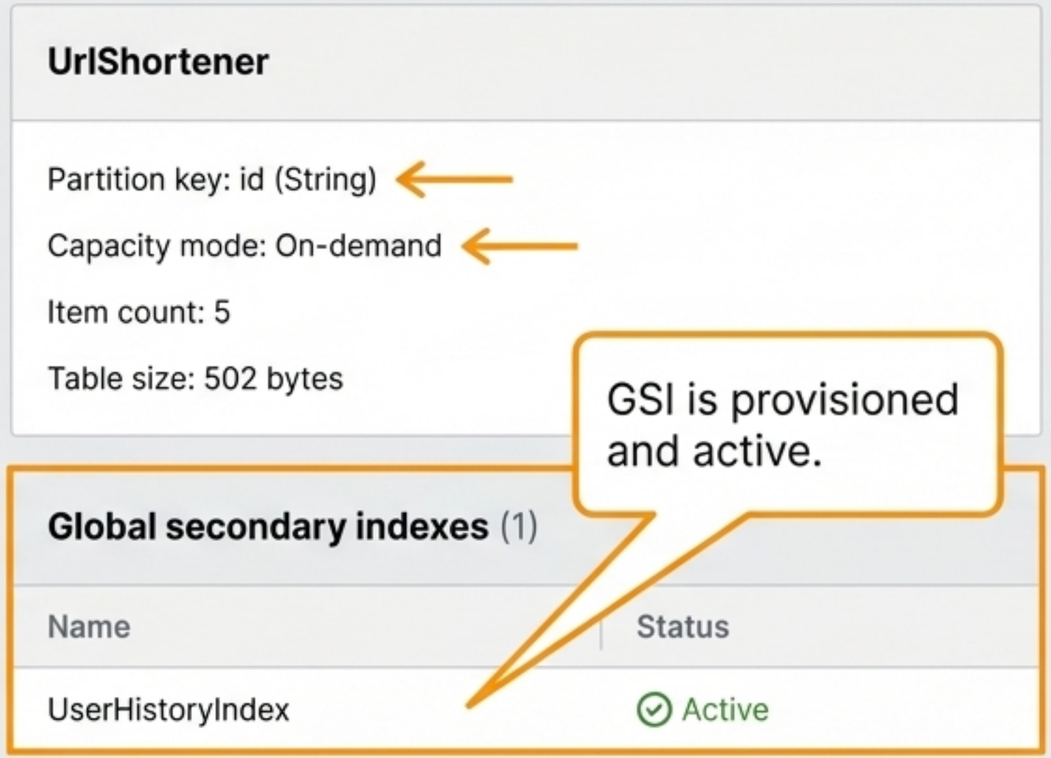
The Blueprint

Table & GSI Design



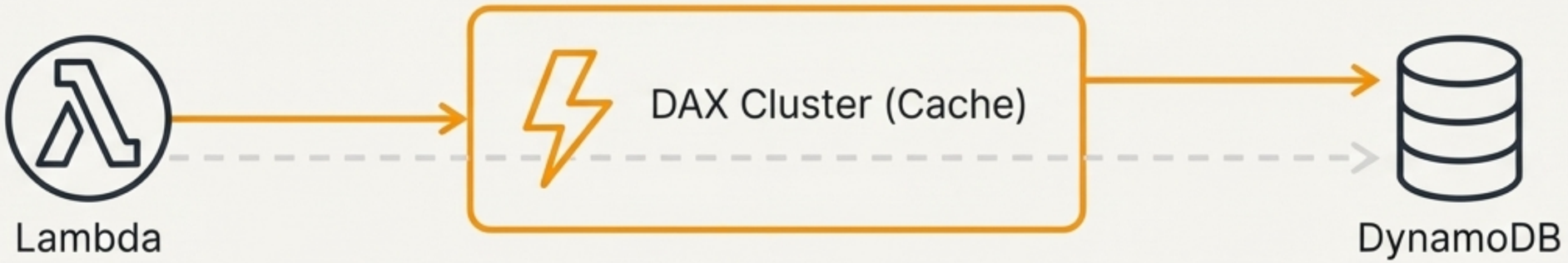
The Reality

AWS Console View



Achieving an ‘Instant’ Dashboard with DynamoDB DAX

The data model uses three distinct DynamoDB tables, each optimised for a specific access pattern, ensuring high performance and scalability.



The Challenge

Loading a user's dashboard requires fetching their URL history and aggregate stats. Even with an optimised schema, this can introduce noticeable latency at scale.

The Solution

DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache for DynamoDB.

<1ms reads
Enable DynamoDB DAX for...

Microsecond
read latency for cached items.

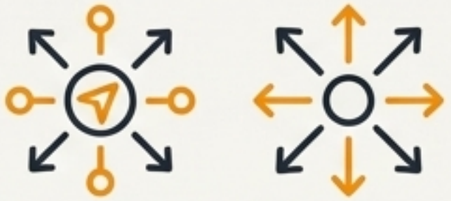
This caching layer is the key to providing an ‘instant’ dashboard feel, as frequent accessed data is served directly from memory, bypassing the database.

Global Delivery: Hosting, Routing, and Security

Frontend Hosting (S3 & CloudFront)

The compiled Flutter web application is hosted as a static site in an S3 bucket. A CloudFront distribution serves these assets from edge locations around the world, ensuring ensuring low latency for all users.

DNS & Routing (Route 53)

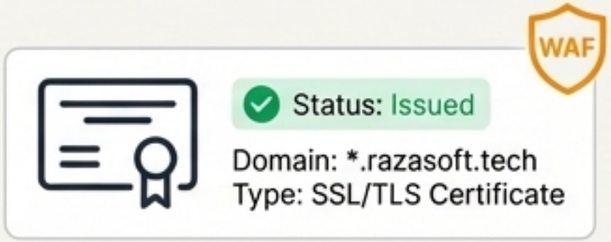


Route 53 manages the application's DNS. It routes user traffic for `razasoft.tech` to the CloudFront distribution and `api.razasoft.tech` to the API Gateway endpoint.

Security (ACM & WAF)

AWS Certificate Manager (ACM) provides the SSL certificate for `*.razasoft.tech`.

AWS WAF protects the API from common web exploits and malicious traffic.



The Path to Production: A Go-Live Checklist

A comprehensive checklist ensures all components are correctly configured and tested before handling live user traffic.

Infrastructure

- [ ] Deploy Lambda functions to all target regions.
- [ ] Configure API Gateway with CORS and authorisers.
- [ ] Create Cognito User Pool.
- [ ] Provision DynamoDB tables and DAX cluster.
- [ ] Configure Route 53, S3, and CloudFront.
- [ ] Deploy AWS WAF with appropriate rules.

Application

- [ ] Set `isDebugMode = false`.
- [ ] Update all API URLs in the Flutter configuration.
- [ ] Integrate API service layer throughout the app.

Validation

- [ ] Test the full authentication flow (sign-up, sign-in, MFA).
- [ ] Test core functionality (URL creation, deletion, analytics).
- [ ] Monitor CloudWatch logs for errors during testing.
- [ ] Set up billing and performance alarms.

Diagnosing and Resolving Common Operational Issues

`401 Unauthorized`

Cause

Missing or invalid JWT token.

Solution

Verify Cognito configuration and ensure the `Authorization` header is correctly sent from the client.

`403 WAF Blocked`

Cause

Request blocked by AWS Web Application Firewall rules.

Solution

Check WAF logs in the AWS Console.
Whitelist trusted IPs for testing or adjust rules.

`429 Rate Limit Exceeded`

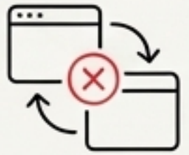
Cause

API Gateway throttling limits have been hit.

Solution

Review throttling settings. Implement exponential backoff in the client. The app already shows a friendly toast message.

`CORS Error`



Cause

API Gateway is not configured to accept cross-origin requests from the web domain.

Solution

Ensure `Access-Control-Allow-Origin` and other required headers are present in the API Gateway method response.

The Result: A Scalable, Performant, and Maintainable System

This architectural blueprint delivers a complete solution that balances developer velocity with production-grade resilience and performance.

Rapid Development

The 'Debug Mode First' approach allows developers to 'start coding features now, and worry about AWS later.'

Global Scale

Multi-region deployment via Lambda and CloudFront provides low latency for a worldwide user base.

Exceptional Performance

DynamoDB DAX delivers microsecond read latency, enabling an instant-feeling user experience.

Robust Security

A layered defence with Cognito, WAF, and secure IAM policies protects the application and its users.

Latency: 45ms (us-east-1)