# LIDP Cloud Challenge

## Technical Documentation

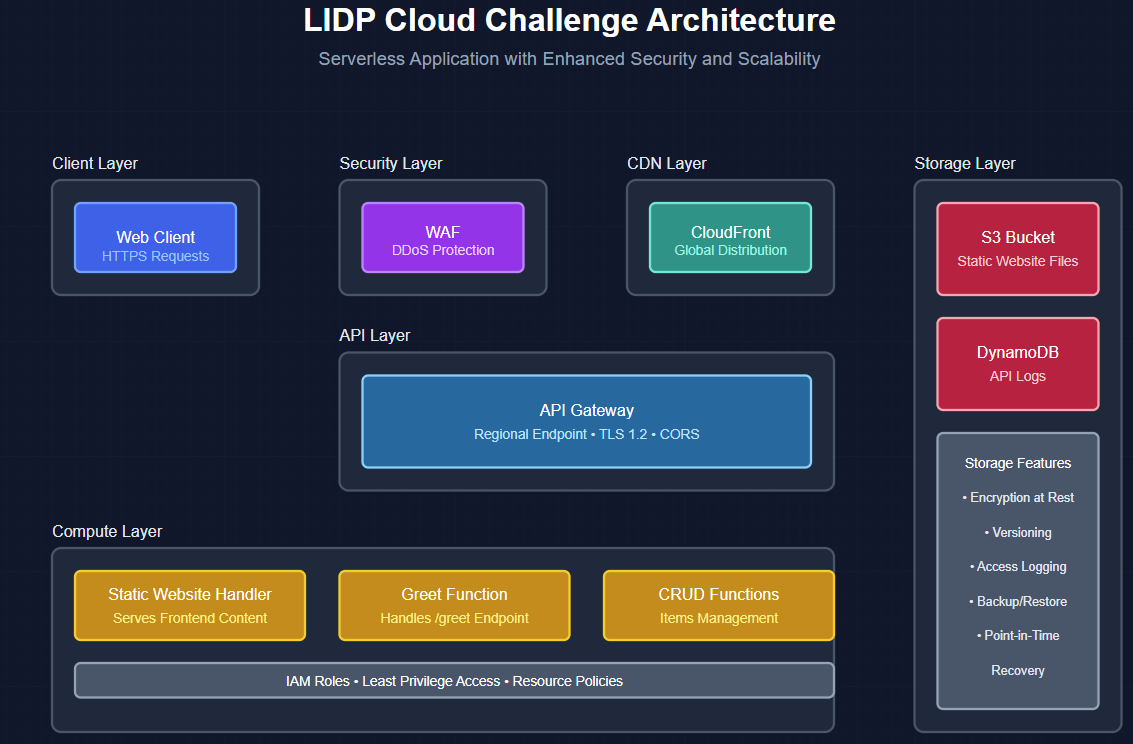
### Version 1.0 - November 2024

## Table of Contents

1. [Overview](#Xc8951b796c7b61cbdd4998c72efbdeed0056e13)
2. [Architecture](#X136a724f0ad6bedecc70d52a088da11b9355002)
3. [Implementation](#X0175033c2822423472b54582ff8b6553c4b237e)
4. [Security](#Xcd216216200afb0703e898683a8d10359544695)
5. [Scalability](#Xe474482f6dcf6e2dddde86bdc2b5e00fc93b2c8)
6. [Monitoring](#Xec1d037a3565c981c81527ac25bc43cb18d1491)
7. [Local Development](#Xaa7dd187d7dc71dcd658cb66cabbb2c90acec69)
8. [Testing](#X710ebb9aa99a164f0edb25d2a0e10e7a1052431)
9. [Deployment](#X0e56c1791bcf637d23ca5080a8af4ff21d00041)
10. [Maintenance](#X3ca02622a7561e870c47ecd74031a48a29566d3)

## 1. Overview

### 1.1 Project Objectives

The LIDP Cloud Challenge demonstrates the implementation of a secure and scalable cloud infrastructure using AWS services. Key features include: - Static website hosting - RESTful API implementation - Comprehensive logging system - Advanced security measures - Complete monitoring solution

### 1.2 Technology Stack

{  
 "core": {  
 "framework": "AWS SAM",  
 "runtime": "Node.js 18.x",  
 "database": "Amazon DynamoDB",  
 "storage": "Amazon S3",  
 "cdn": "Amazon CloudFront",  
 "security": ["AWS WAF", "IAM"],  
 "monitoring": ["CloudWatch", "X-Ray"]  
 },  
 "development": {  
 "containerization": "Docker",  
 "testing": "Jest",  
 "ci\_cd": "AWS CodePipeline"  
 }  
}

### 1.3 System Requirements

#### Development Environment

requirements:  
 node: ">=18.x"  
 npm: ">=8.x"  
 docker: ">=20.x"  
 aws-sam-cli: ">=1.x"  
 memory: ">=4GB"  
 disk\_space: ">=10GB"

#### Production Environment

aws\_requirements:  
 account: required  
 permissions:  
 - AWSCloudFormationFullAccess  
 - AWSLambda\_FullAccess  
 - IAMFullAccess  
 - AWSCloudWatchFullAccess

## 2. Architecture

### 2.1 Core Components

#### 2.1.1 Static Website (Frontend)

StaticWebsiteBucket:  
 Type: AWS::S3::Bucket  
 Properties:  
 WebsiteConfiguration:  
 IndexDocument: index.html  
 ErrorDocument: error.html  
 BucketEncryption:  
 ServerSideEncryptionConfiguration:  
 - ServerSideEncryptionByDefault:  
 SSEAlgorithm: AES256

Features: - S3-hosted static content - CloudFront distribution - SSL/TLS enforcement - Optimized caching - Public access blocking - Version control

#### 2.1.2 API Gateway (Backend Interface)

ApiGateway:  
 Type: AWS::Serverless::Api  
 Properties:  
 StageName: Prod  
 EndpointConfiguration:  
 Type: REGIONAL  
 Auth:  
 DefaultAuthorizer: NONE  
 Cors:  
 AllowMethods: "'GET,POST,OPTIONS'"  
 AllowHeaders: "'Content-Type,Authorization'"  
 AllowOrigin: "'\*'"

Characteristics: - Regional endpoint - CORS configuration - HTTP/HTTPS support - Request/response mapping - API key management

#### 2.1.3 Lambda Functions

// Greeting Function Example  
export const handler = async (event) => {  
 const timestamp = new Date().toISOString();  
 const logEntry = {  
 id: `log\_${Date.now()}`,  
 timestamp,  
 path: event?.path || "",  
 method: event?.httpMethod || ""  
 };  
  
 try {  
 // Log to DynamoDB  
 await logRequest(logEntry);  
  
 return {  
 statusCode: 200,  
 headers: {  
 "Content-Type": "application/json",  
 "Access-Control-Allow-Origin": "\*"  
 },  
 body: JSON.stringify({  
 message: "Hello from AWS Lambda!",  
 timestamp  
 })  
 };  
 } catch (error) {  
 console.error('Error:', error);  
 return {  
 statusCode: 500,  
 body: JSON.stringify({  
 error: "Internal Server Error"  
 })  
 };  
 }  
};

Implementation details: - Error handling - Logging integration - Response formatting - CORS headers - Performance optimization

#### 2.1.4 DynamoDB (Database)

APILogsTable:  
 Type: AWS::DynamoDB::Table  
 Properties:  
 AttributeDefinitions:  
 - AttributeName: id  
 AttributeType: S  
 - AttributeName: timestamp  
 AttributeType: S  
 KeySchema:  
 - AttributeName: id  
 KeyType: HASH  
 - AttributeName: timestamp  
 KeyType: RANGE  
 BillingMode: PAY\_PER\_REQUEST

Features: - Serverless NoSQL database - Auto-scaling capabilities - Point-in-time recovery - Encryption at rest - Backup management

## 3. Implementation

### 3.1 Project Structure

project/  
├── src/  
│ ├── handlers/  
│ │ ├── greet.mjs  
│ │ ├── get-all-items.mjs  
│ │ └── put-item.mjs  
│ └── utils/  
│ └── dynamodb-client.js  
├── tests/  
│ └── unit/  
│ └── handlers/  
├── template.yaml  
├── package.json  
└── README.md

### 3.2 Configuration Management

Globals:  
 Function:  
 Runtime: nodejs18.x  
 Timeout: 29  
 MemorySize: 128  
 Tracing: Active  
 Environment:  
 Variables:  
 LOG\_LEVEL: INFO

### 3.3 Development Workflow

# Setup Development Environment  
npm install  
docker network create sam-network  
docker run -p 8000:8000 amazon/dynamodb-local  
  
# Build and Deploy Locally  
sam build  
sam local start-api --docker-network sam-network  
  
# Test API  
curl http://localhost:3000/greet

## 4. Security

### 4.1 IAM Roles and Policies

FunctionRole:  
 Type: AWS::IAM::Role  
 Properties:  
 AssumeRolePolicyDocument:  
 Version: '2012-10-17'  
 Statement:  
 - Effect: Allow  
 Principal:  
 Service: lambda.amazonaws.com  
 Action: sts:AssumeRole  
 ManagedPolicyArns:  
 - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole  
 Policies:  
 - PolicyName: DynamoDBAccess  
 PolicyDocument:  
 Version: '2012-10-17'  
 Statement:  
 - Effect: Allow  
 Action:  
 - dynamodb:PutItem  
 - dynamodb:GetItem  
 Resource: !GetAtt APILogsTable.Arn

### 4.2 WAF Configuration

WebACL:  
 Type: AWS::WAFv2::WebACL  
 Properties:  
 DefaultAction:  
 Allow: {}  
 Rules:  
 - Name: RateLimit  
 Priority: 1  
 Statement:  
 RateBasedStatement:  
 Limit: 2000  
 AggregateKeyType: IP

### 4.3 SSL/TLS Implementation

CloudFrontDistribution:  
 Type: AWS::CloudFront::Distribution  
 Properties:  
 DistributionConfig:  
 ViewerCertificate:  
 CloudFrontDefaultCertificate: true  
 DefaultCacheBehavior:  
 ViewerProtocolPolicy: redirect-to-https

## 5. Scalability

### 5.1 Auto-scaling Configuration

ScalableTarget:  
 Type: AWS::ApplicationAutoScaling::ScalableTarget  
 Properties:  
 MinCapacity: 1  
 MaxCapacity: 10  
 ResourceId: !Sub table/${TableName}  
 ScalableDimension: dynamodb:table:ReadCapacityUnits  
 ServiceNamespace: dynamodb

### 5.2 Performance Optimization

// Lambda cold start optimization  
const AWS = require('aws-sdk');  
const dynamoDB = new AWS.DynamoDB.DocumentClient();  
const connectionConfig = {  
 maxRetries: 3,  
 timeout: 1000,  
 httpOptions: {  
 connectTimeout: 1000,  
 timeout: 1000  
 }  
};

## 6. Monitoring

### 6.1 CloudWatch Integration

Alarms:  
 HighErrorRate:  
 Type: AWS::CloudWatch::Alarm  
 Properties:  
 AlarmDescription: High error rate detected  
 MetricName: Errors  
 Namespace: AWS/Lambda  
 Statistic: Sum  
 Period: 300  
 EvaluationPeriods: 1  
 Threshold: 5  
 AlarmActions:  
 - !Ref AlertsTopic

### 6.2 X-Ray Tracing

Globals:  
 Function:  
 Tracing: Active

## 7. Local Development

### 7.1 Setup Instructions

# Clone repository  
git clone [repository-url]  
cd [repository-name]  
  
# Install dependencies  
npm install  
  
# Setup Docker network  
docker network create sam-network  
  
# Start DynamoDB Local  
docker run --rm -p 8000:8000 --network sam-network --name dynamodb amazon/dynamodb-local  
  
# Build and start SAM application  
sam build  
sam local start-api --docker-network sam-network

### 7.2 Development Tools

* VS Code with AWS Toolkit
* AWS SAM CLI
* Docker Desktop
* Postman for API testing

## 8. Testing

### 8.1 Unit Tests

import { handler } from '../src/handlers/greet.mjs';  
  
describe('Greet Handler', () => {  
 test('should return 200 with greeting', async () => {  
 const event = {  
 httpMethod: 'GET',  
 path: '/greet'  
 };  
  
 const response = await handler(event);  
 expect(response.statusCode).toBe(200);  
 expect(JSON.parse(response.body)).toHaveProperty('message');  
 });  
});

### 8.2 Integration Tests

# Test API endpoints  
curl http://localhost:3000/greet  
curl http://localhost:3000/items

## 9. Deployment

### 9.1 Production Deployment

# Deploy with guided setup  
sam deploy --guided  
  
# Deploy with existing configuration  
sam deploy

### 9.2 Deployment Configuration

version = 0.1  
[default.deploy.parameters]  
stack\_name = "challenge-SAM"  
resolve\_s3 = true  
confirm\_changeset = true  
capabilities = "CAPABILITY\_IAM"

## 10. Maintenance

### 10.1 Backup Strategy

DynamoDBTable:  
 Type: AWS::DynamoDB::Table  
 Properties:  
 PointInTimeRecoverySpecification:  
 PointInTimeRecoveryEnabled: true

### 10.2 Monitoring and Alerts

AlertTopic:  
 Type: AWS::SNS::Topic  
 Properties:  
 DisplayName: "Critical Alerts"

## Conclusion

This implementation provides a robust foundation for: 1. Secure and scalable serverless applications 2. Cost-effective cloud infrastructure 3. Maintainable and testable codebase 4. Production-ready deployment 5. Comprehensive monitoring and logging

For updates or modifications, please refer to the project repository and maintain consistent documentation updates.

## Support and Resources

* Documentation Wiki: [Project Wiki]
* Issue Tracking: GitHub Issues
* Contact: Guilhermergsilva@gmail.com

## License

MIT License - See LICENSE file for details