High Level Design

Persistent date-time stamp application

Revision Number: 1.0

Last date of revision: 19/7/20

Damon Wright

Contents

1. Introduction

1.1. Why this High Level Design Document?

The purpose of this High Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

1.2. Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

1.3. Definitions

- HLD – High Level Design
- AWS – Amazon Web Service is a public cloud computing service provided by Amazon
- API – Application programming interface; a computing interface which defines interactions between multiple software intermediaries
- API Gateway – Managed service provided by AWS to build API endpoints
- Serverless – Allows users to write and deploy code without the need to build or provision the underlying infrastructure.
- Lambda – An AWS service for providing serverless computing
- NoSQL – A database type that does not allow strict relations
- Dynamodb – An AWS Managed NoSQL database
- IAM – Access control system within AWS
- Route53 – AWS provided DNS service
- DNS – Domain Name System; lookup of server friendly names to alternate names or IP addresses
- IP – Physical location of a server that is routable in numeric terms, e.g. 192.168.13.4
- X509 – A certificate type for exchanging keys used for TLS
- TLS – Transport Layer Security – Allows servers to send data between them with encryption
- Certificate Manager – AWS service for providing x509 certificates
- Ansible – Configuration management tool written in python
- Python – Interpreted programming language
- Cloudwatch – AWS managed logging solution
- Git --- Source control software

2. General Description

2.1. Product Perspective

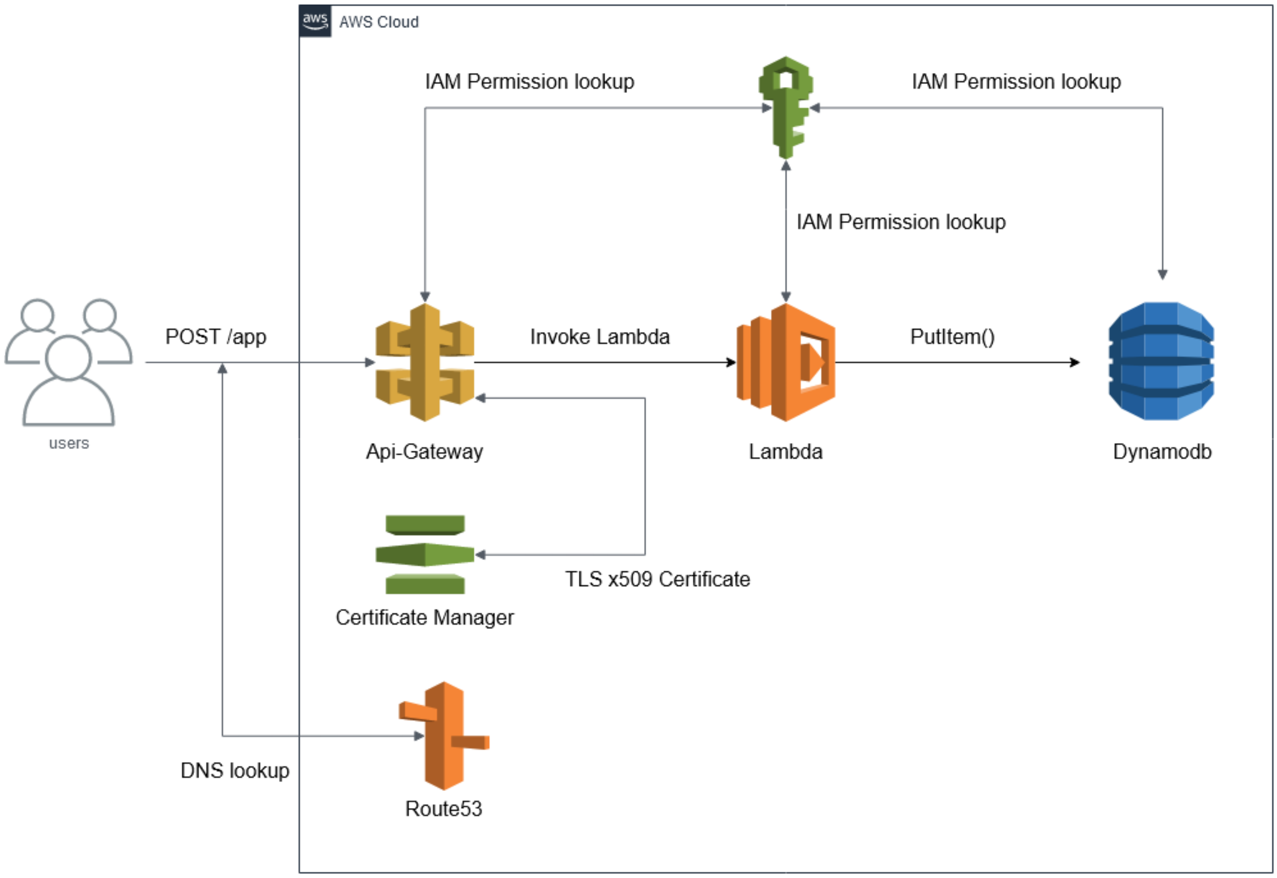
The Persistent date-time stamp application will comprise of a frontend provided by api-gateway, the backend a lambda function written in python and a persistent data store of Dynamodb. All of this will be running within AWS. The user will only be permitted to POST information to the application and only to the /app endpoint. The application will be deployed and managed using ansible.

2.2. Tools used

a. Git for source version control
b. Ansible for configuration management
c. Python for the lambda function
d. Data will be sent to the backend database using NoSQL
e. AWS for server hosting

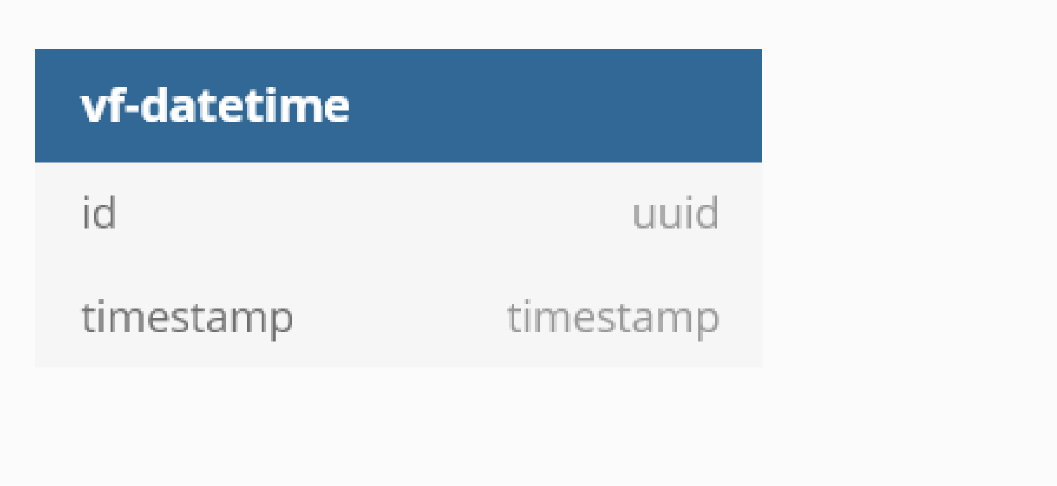
3. Design Details

3.1. Architecture Diagram



3.2. Database Design

The database consists of a single table shown below.



vf-datetime	
id	uuid
timestamp	timestamp

3.3. User interface

This application does not provide a user interface. It is purely API based and will only response successfully to POST requests.

3.4. Security

The application has no defined access control and will create a record on any anonymous well formatted request. Permissions within AWS are to a higher level. The api-gateway configuration only has permission to invoke the lambda function and nothing else. The lambda function has the permission to write logs to cloudwatch and put items to the vf-datetime, nothing else.

3.5 Scalability

The three core components of the application; api-gateway, lambda and dynamodb, where all chosen as they are self-scaling and highly available by default. No extra configuration has been provided to enforce high availability.

3.6 Backup & Restore

Backup and restore is out of scope of this application at this time.