Payment API Architecture

Notes

- Live demonstration: https://f3.devenney.io/
    - *Endpoints are described below and in the code.*
- Source repository: https://github.com/devenney/form3

Architecture Diagram

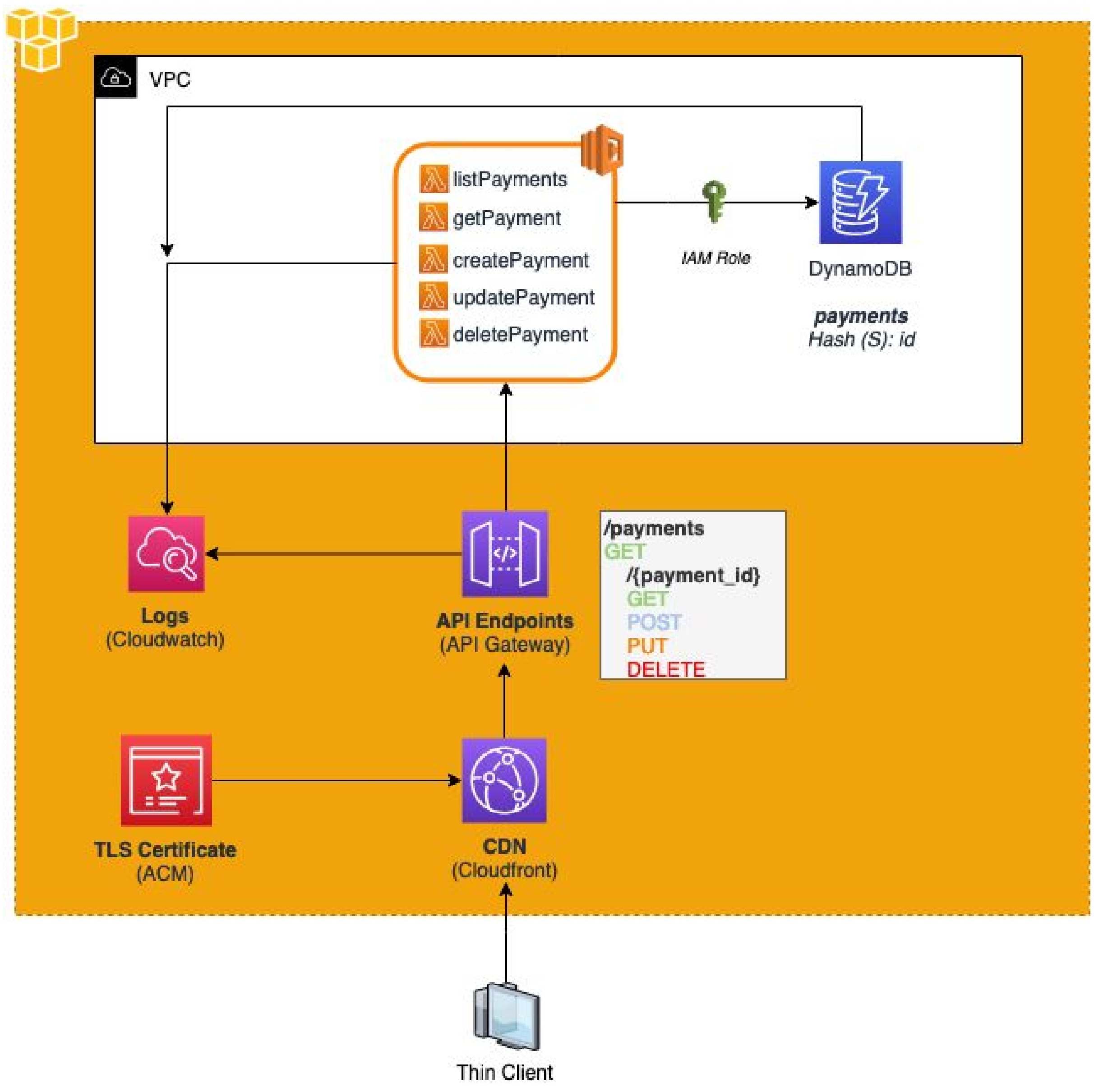

Fig. 1: High-Level Design Diagram

Technology Justification

- **Golang**: Mandatory requirement.
- **AWS:** Best known technology chosen for rapid development.
- **Serverless Framework**
    - Underlying: Lambda, API Gateway, Cloudfront, ACM
    - Prototype has a strong focus on code, Serverless Framework will allow best-practice deployment with minimal effort.
- **DynamoDB:** Chosen over RDS for the sake of simplicity (mock data set has many keys which would take significant time to model in SQL migrations).

API Endpoints

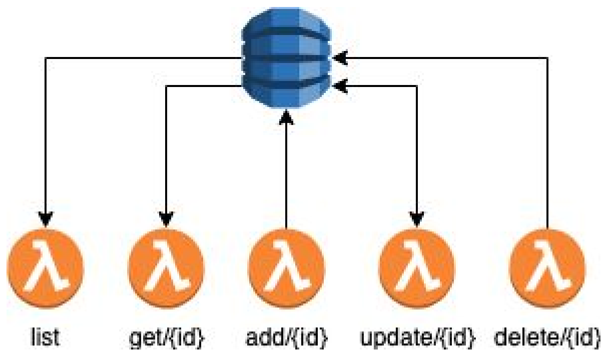

Fig 2. API Data Flow

The API Endpoints are provided via a common serverless pattern on AWS:

- Cloudfront provides scalability, traffic scrubbing and HTTPS.
- API Gateway provides routing to the backend based on endpoint and request type.
    - Can easily be extended to support authentication and such.
- Lambda implements the business logic, split out at the function level to maximise scalability.

Note: Refer to the code repository as the single source of truth for documentation of the individual endpoints.

Data Persistence

Data is persisted to DynamoDB. Initial demonstration will use a single table with a solitary Hash key (the Payment ID) - this is almost certainly incorrect but can easily be extended.

Logging and Monitoring

Cloudfront, API Gateway, Lambda and DynamoDB all natively support logging to Cloudwatch Logs. This ensures the solution is auditable at a basic level and aids in end-to-end debugging.

Development Setup

A development environment will be provided to demonstrate the flexibility of the code and the consideration of the developers’ quality of life. This will include:

- A docker-compose network with:
    - A load balancer which provides a well-known local DNS record.
    - A local instance of DynamoDB which behaves exactly like the AWS endpoint.
- A Makefile which provides common tasks such as installation and CI tests.

Testing

Unit tests will be provided which cover the full traffic flow (from HTTP client to DB transaction and back again). These will, where possible, use extensible test maps which are fed into the logic. To add further tests the developer would then only have to extend the test map - not duplicate the logic.

Pact testing will also be demonstrated. As a real client is out of scope, a mock client will be written in Go to produce a fake Pact file which can be used for this testing.

Seeding

The provided mock data set will be used to seed unit tests, integration tests and Pact testing. This will also be formalised into a command which can be used to seed the demonstration API (assuming time allows this live demonstration to be completed).