Architecture Project #1

Overview: Running App

Software Architecture Course

Project Part 1

The goal of the running application is to allow users to track their running habits and get motivated by visualizing their progress.

Unfortunately, customers have been complaining about some problems, and our business is losing customers to the competition.

In this document we have devised a one month plan to tackle the three immediate pain points with an improve in our software architecture.

Primary Goals

- the app is not responsive during a run, especially trying to start and stop it
- some customers mentioned that their runs are disappearing from the web dashboard
- when customers finish a run, it takes more than one day to view on the dashboard

Secondary Goals

- Reduce cost of total cloud account

Scope: Our Running application System and their interactions

Intended audience: CEO, CTO, and CFO are the primary audience for this presentation. Eventually it will be extended to the development team in order to provide clarity on the goals to follow ahead.

Assumptions:

- The dev team is capable to work on the code and provide required refactoring.
- The devops team is capable to work and manage our cloud provider
- The QA team can actually reproduce the issues and test the app is running.
- No staff change is likely to happen on the suggested time frame.
- We will utilize a bit over half the IT team separated at 3 teams of 6 devs/devops/qa, for 2 consecutive sprints.

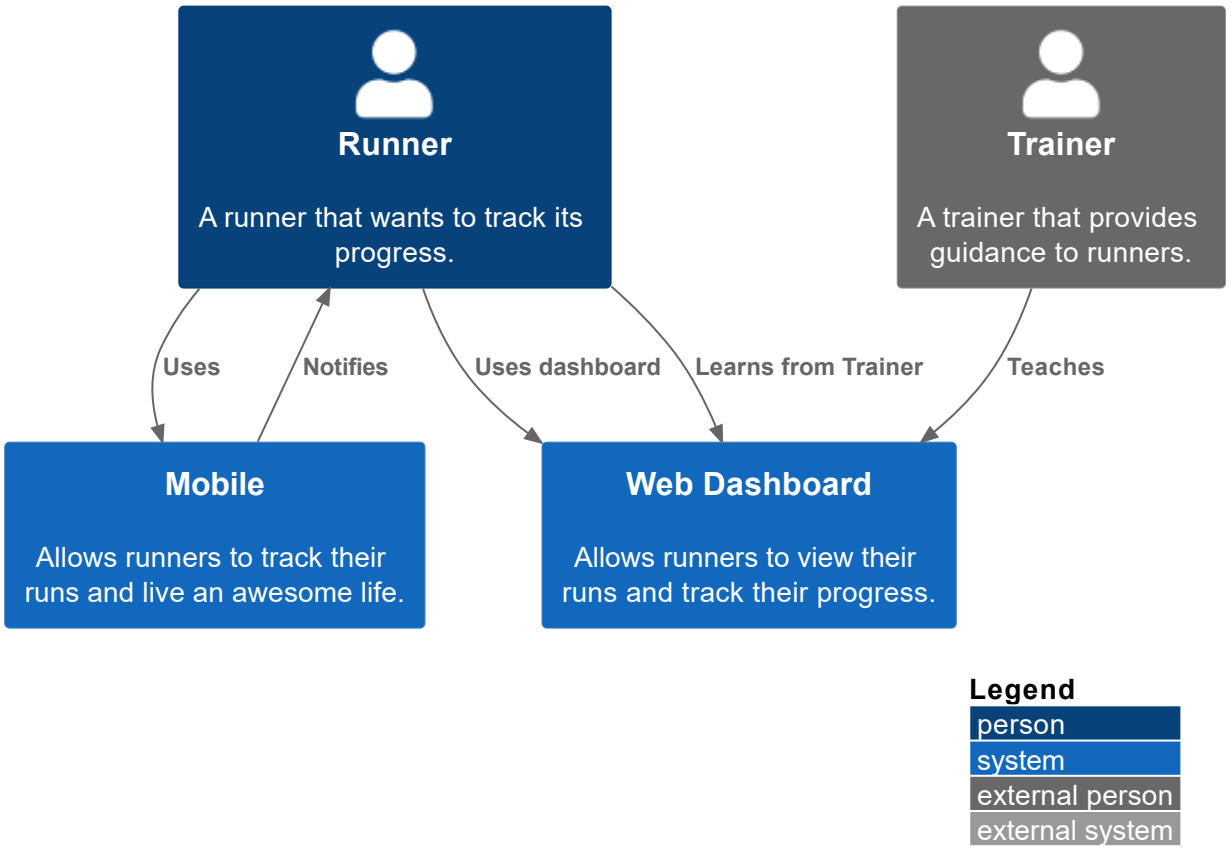
- The allocation of staff is agreed upon and no changes to the plan will be done on the next month

Limitations: We will not be generating code diagrams for this project. Instead we will focus on the Deployment Diagram, which will map our current platform to the physical cloud architecture, which will help us visualize better the high level architectural changes.

Tools: This document utilizes C4 Diagrams and (Markdown files)[https://www.markdownguide.org/] to generate all the pages using (VSCode)[https://code.visualstudio.com/], the (PlantUML plugin) [https://marketplace.visualstudio.com/items?itemName=jebbs.plantuml] and the (nodejs c4builder project) [https://adrianvlupu.github.io/C4-Builder/#/].

1 System Context Diagram

```
\1 System Context Diagram
```



Level 1: System Context

The system context diagram represents a high-level overview of the interactions of the main components of the current architecture.

Primary Elements

There are currently one main user and two main components on this application. The primary elements are as follow:

- Our main users are runners that want to track and improve their running habits. These persons range from teens to seniors of all genders.

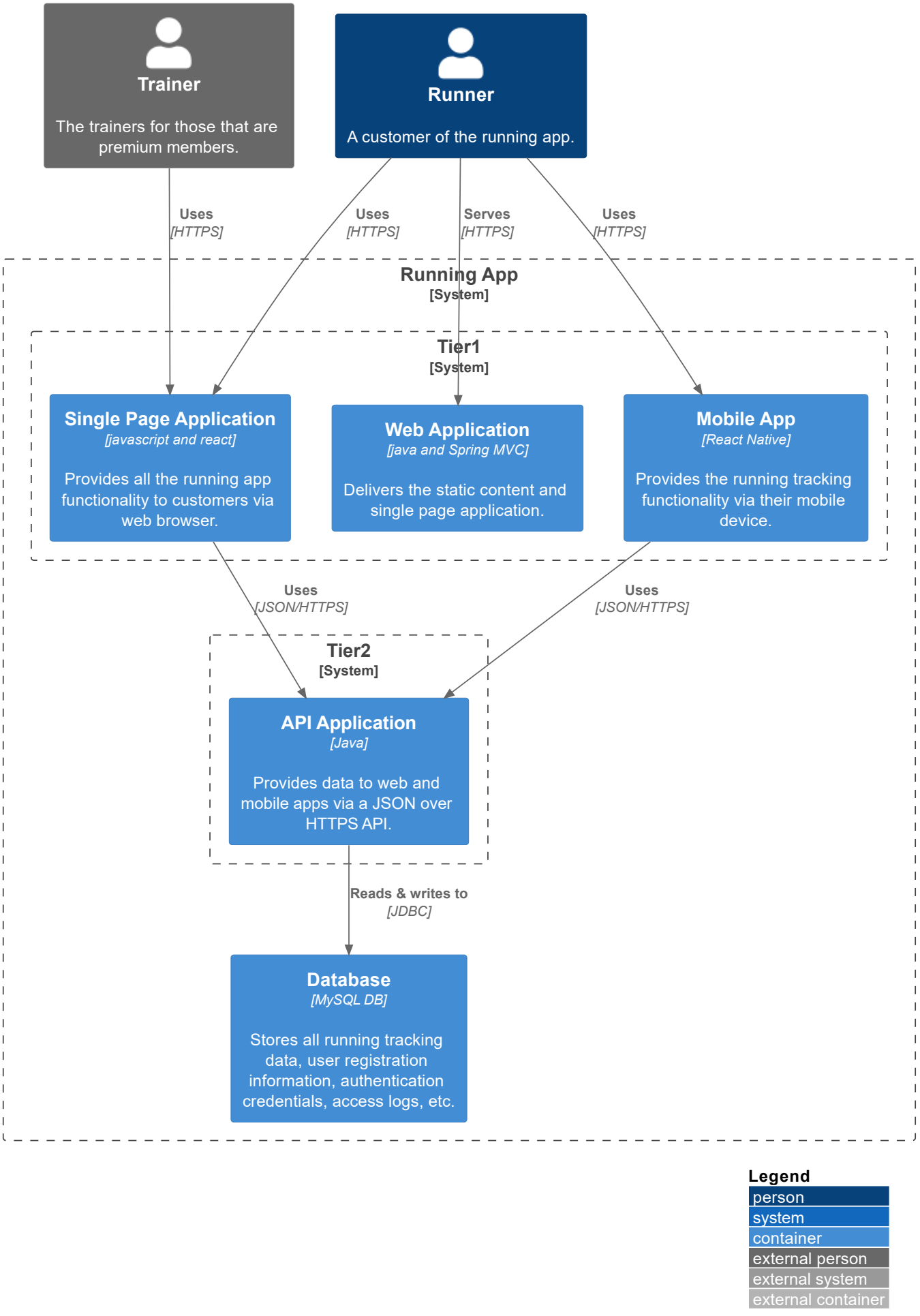
- The first main component is the mobile application, designed to allow users to select a running program, track their runs with GPS data , count calories of each run , and enhance the running experience with a music playlist.
- The second main component is a web dashboard that is designed to give users an overview of their runs and track their progress over time. They can also manage their accounts and purchase advanced plans with live trainer support.

Secondary Elements

- Trainers are professional runners that provide services guiding our runners into their journey.

1.1 Container Diagram

`\1 System Context Diagram\1.1 Container Diagram`



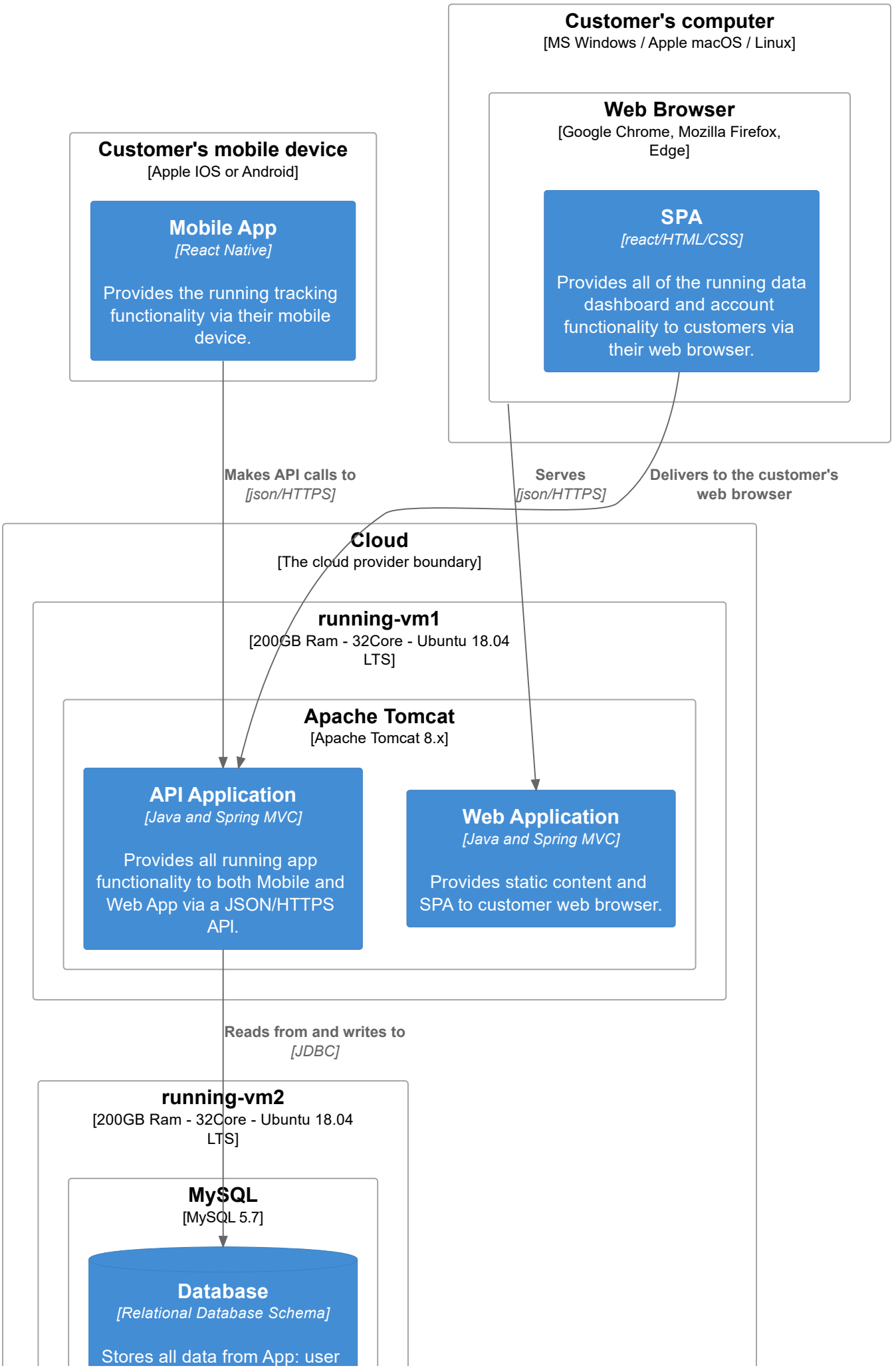
Level 2: Container diagram

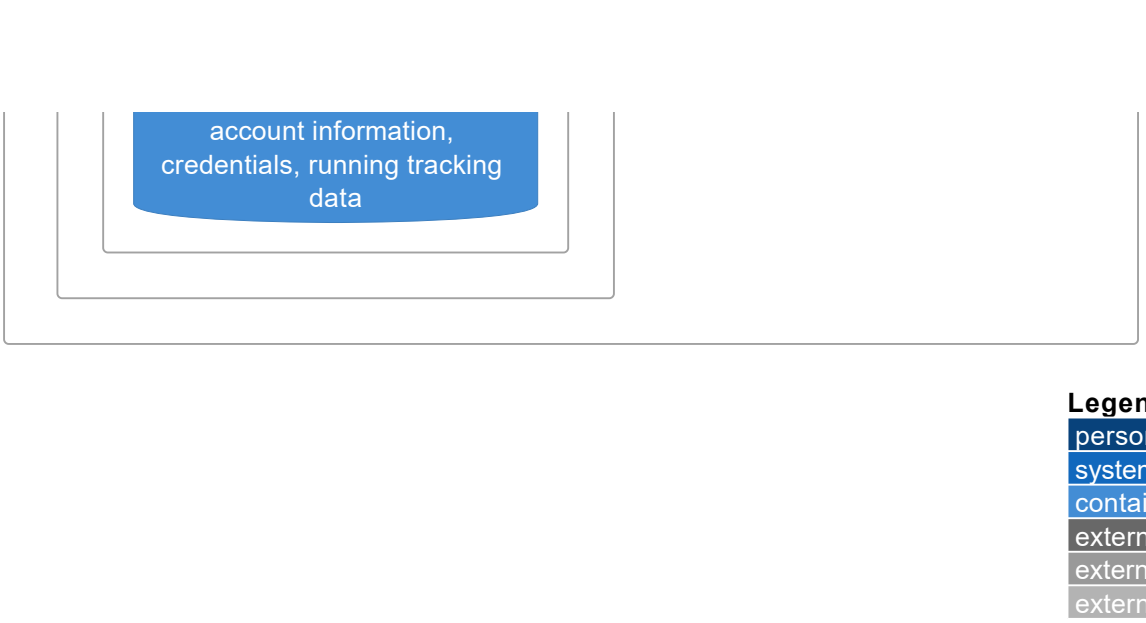
Each part of our application is a two tier app, which means we have a code layer for presentation and one for business logic. The presentation layer on mobile is coded on native code and fetches data from the API via RESTful calls.

The same pattern is observed with our Web Dashboard application, which retrieve data from the API layer and serve the HTML pages to the clients.

2 Current Deployment Diagram

```
\2 Current Deployment Diagram
```


Customer's computer
[MS Windows / Apple macOS / Linux]
Web Browser
[Google Chrome, Mozilla Firefox, Edge]
SPA
[react/HTML/CSS]
Provides all of the running data dashboard and account functionality to customers via their web browser.
Customer's mobile device
[Apple IOS or Android]
Mobile App
[React Native]
Provides the running tracking functionality via their mobile device.
Makes API calls to
[json/HTTPS]
Serves
[json/HTTPS]
Delivers to the customer's web browser
Cloud
[The cloud provider boundary]
running-vm1
[200GB Ram - 32Core - Ubuntu 18.04 LTS]
Apache Tomcat
[Apache Tomcat 8.x]
API Application
[Java and Spring MVC]
Provides all running app functionality to both Mobile and Web App via a JSON/HTTPS API.
Web Application
[Java and Spring MVC]
Provides static content and SPA to customer web browser.
Reads from and writes to
[JDBC]
running-vm2
[200GB Ram - 32Core - Ubuntu 18.04 LTS]
MySQL
[MySQL 5.7]
Database
[Relational Database Schema]
Stores all data from App: user



Deployment diagram

This section we view how containers in the static model are mapped to infrastructure.

Currently we have the mobile app installed on the client directly fetching data from the API server. The installation files are served to the users phone by the respective platform marketplaces.

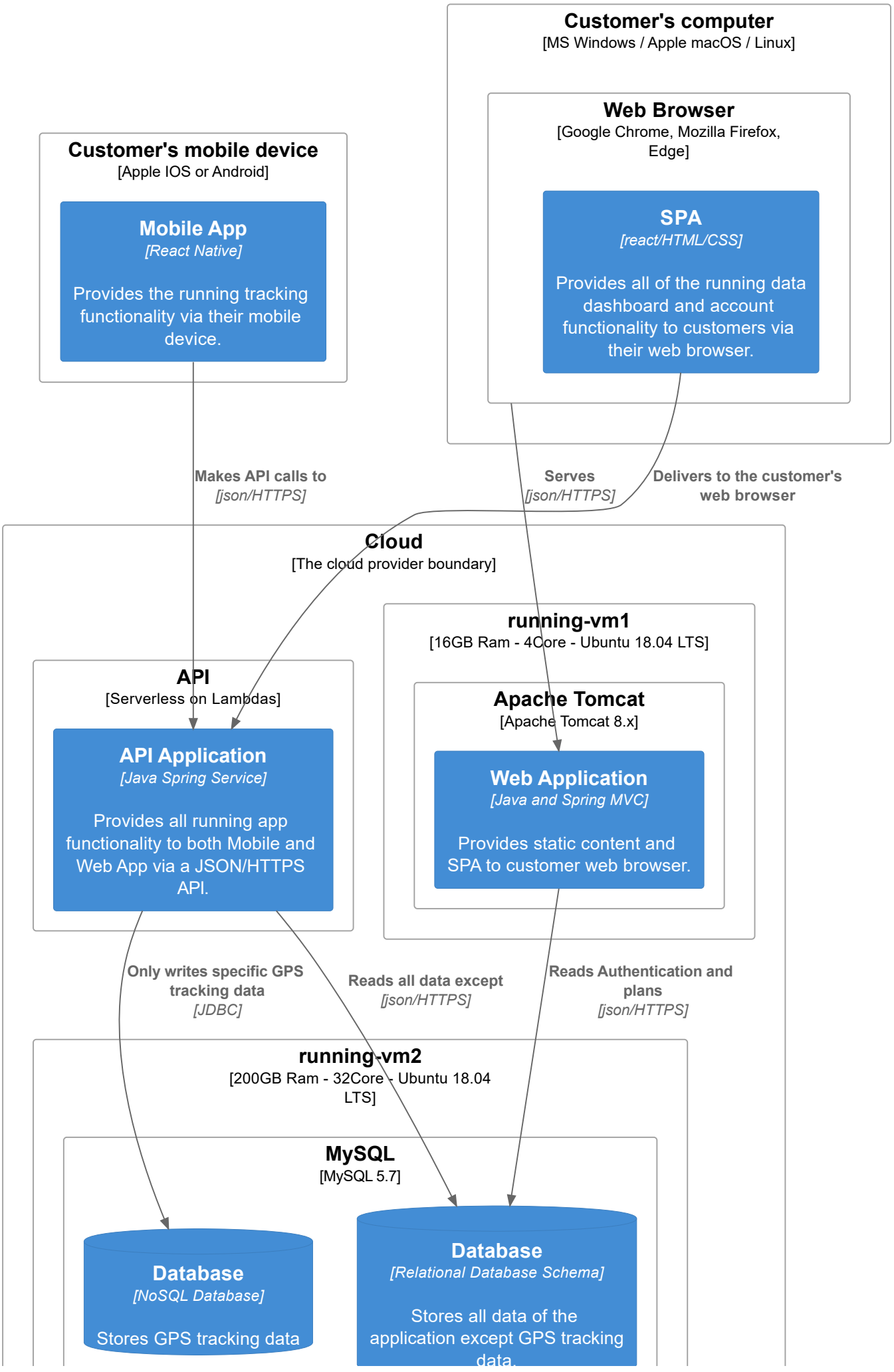
There are two VM instances of 200GB RAM and 32vCPUs. On the first instance , we have our web server that delivers a SPA app and static content to the web clients. The SPA then runs on the browser and communicates directly to the API.

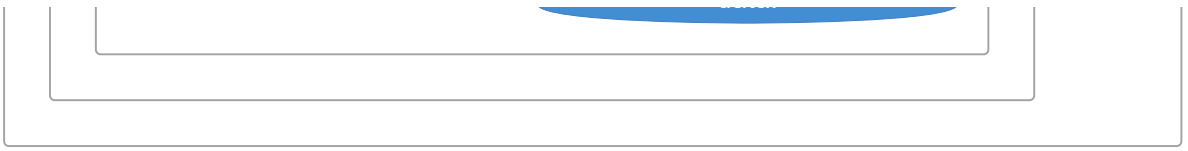
The API layer is also on the the first VM instance and receives all requests from reads and writes all data to the database.

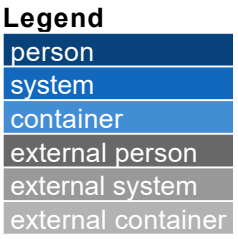
The Database is installed on the second VM and it is a single MySQL database instance that managed by our cloud services provider and we only have read/write access.

3 One Month Plan

```
\3 One Month Plan
```


Customer's computer
[MS Windows / Apple macOS / Linux]
Web Browser
[Google Chrome, Mozilla Firefox, Edge]
SPA
[react/HTML/CSS]
Provides all of the running data dashboard and account functionality to customers via their web browser.
Customer's mobile device
[Apple IOS or Android]
Mobile App
[React Native]
Provides the running tracking functionality via their mobile device.
Makes API calls to
[json/HTTPS]
Serves
[json/HTTPS]
Delivers to the customer's web browser
Cloud
[The cloud provider boundary]
running-vm1
[16GB Ram - 4Core - Ubuntu 18.04 LTS]
API
[Serverless on Lambdas]
API Application
[Java Spring Service]
Provides all running app functionality to both Mobile and Web App via a JSON/HTTPS API.
Apache Tomcat
[Apache Tomcat 8.x]
Web Application
[Java and Spring MVC]
Provides static content and SPA to customer web browser.
Only writes specific GPS tracking data
[JDBC]
Reads all data except
[json/HTTPS]
Reads Authentication and plans
[json/HTTPS]
running-vm2
[200GB Ram - 32Core - Ubuntu 18.04 LTS]
MySQL
[MySQL 5.7]
Database
[NoSQL Database]
Stores GPS tracking data
Database
[Relational Database Schema]
Stores all data of the application except GPS tracking data.





One Month Attack Plan

To improve the user experience for our consumers, we will do the following modifications:

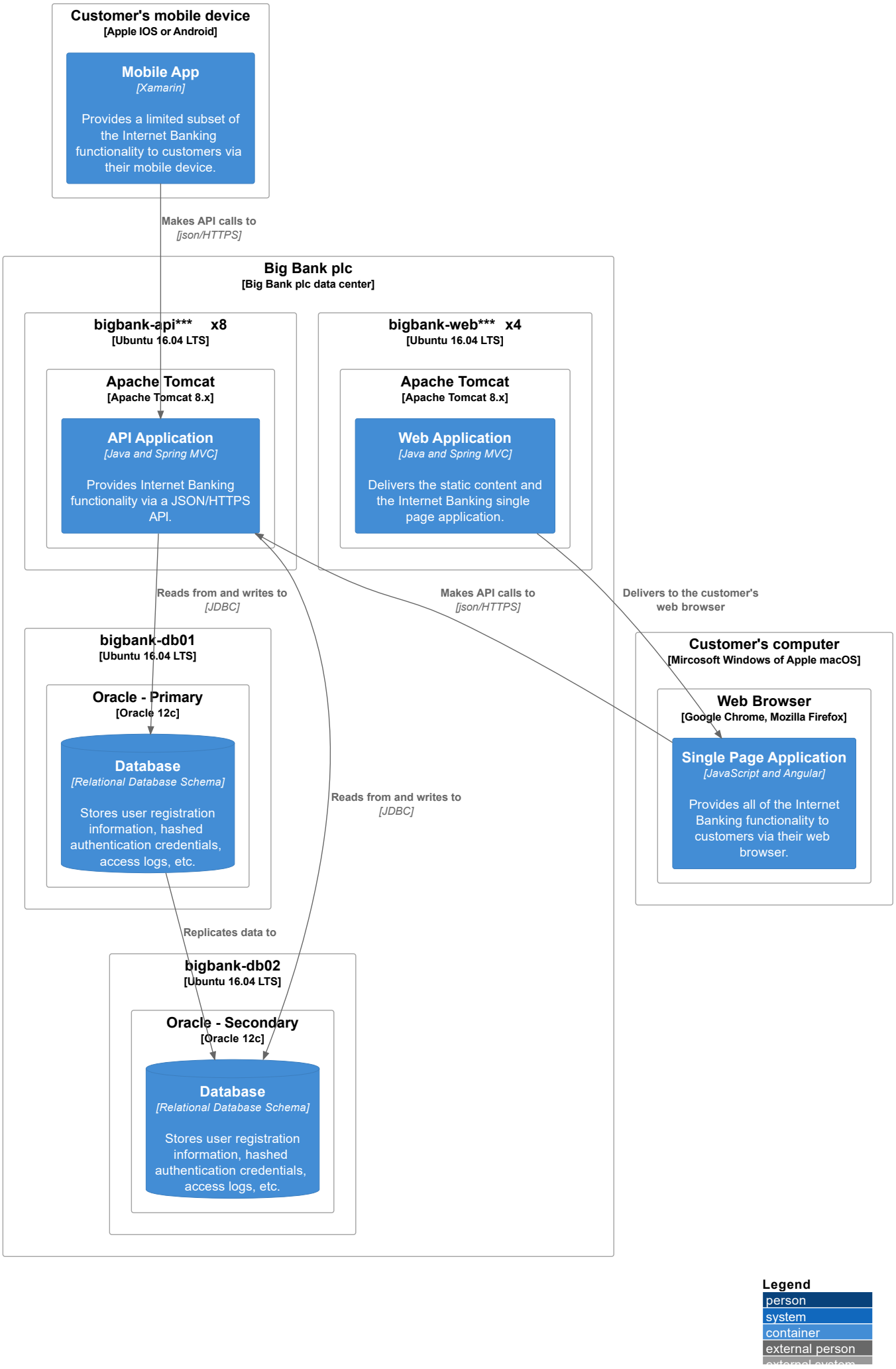
In order to maximize the elasticity of the providers, we will convert all API layer into Functions (lambdas). This will allow us to only use the resources that are actually necessary for each request instead of guessing the size of the VM to host our app. Due to this migration , we will be able to reduce the size of the actual VM Instance 1, since it will only be used to serve static content and the SPA app.

Also, for the Vm Instance 2 that hosts our single MySQL database instance, we will create a second instance that will be focused on write only coming from the mobile apps, such as tracking data that is provided on each run. The original MySQL instance will

Although it seems that we will be utilizing more of the cloud provider, in fact we will be able to better utilize resources by scaling the app up when we need it most, such as 7AM-8AM, 12PM-01PM, 07PM-08PM, and scale down when there are almost no usage, such as during the night, from 11PM-06AM. Remember that for the time being our customer base is only the USA.

4 Long Term Plan

```
\4 Long Term Plan
```


Customer's mobile device
[Apple IOS or Android]
Mobile App
[Xamarin]
Provides a limited subset of the Internet Banking functionality to customers via their mobile device.
Makes API calls to
[json/HTTPS]
Big Bank plc
[Big Bank plc data center]
bigbank-api*** x8
[Ubuntu 16.04 LTS]
Apache Tomcat
[Apache Tomcat 8.x]
API Application
[Java and Spring MVC]
Provides Internet Banking functionality via a JSON/HTTPS API.
bigbank-web*** x4
[Ubuntu 16.04 LTS]
Apache Tomcat
[Apache Tomcat 8.x]
Web Application
[Java and Spring MVC]
Delivers the static content and the Internet Banking single page application.
Reads from and writes to
[JDBC]
Makes API calls to
[json/HTTPS]
Delivers to the customer's web browser
bigbank-db01
[Ubuntu 16.04 LTS]
Oracle - Primary
[Oracle 12c]
Database
[Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Customer's computer
[Mircosoft Windows of Apple macOS]
Web Browser
[Google Chrome, Mozilla Firefox]
Single Page Application
[JavaScript and Angular]
Provides all of the Internet Banking functionality to customers via their web browser.
Reads from and writes to
[JDBC]
Replicates data to
bigbank-db02
[Ubuntu 16.04 LTS]
Oracle - Secondary
[Oracle 12c]
Database
[Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Legend
person
system
container
external person

external system
external container

Deployment diagram

This section we view how containers in the static model are mapped to infrastructure.

Currently we have the mobile app installed on the client directly fetching data from the API server. The installation files are served to the users phone by the respective platform marketplaces.

Our web server is already on Azure cloud, and its deployed on a single VM using IIS as a web server. The code also fetches data from the API layer.

The API layer reads and writes all data from a single instance MySQL server.

A deployment diagram allows you to illustrate how containers in the static model are mapped to infrastructure. This deployment diagram is based upon a UML deployment diagram, although simplified slightly to show the mapping between containers and deployment nodes. A deployment node is something like physical infrastructure (e.g. a physical server or device), virtualised infrastructure (e.g. IaaS, PaaS, a virtual machine), containerised infrastructure (e.g. a Docker container), an execution environment (e.g. a database server, Java EE web/application server, Microsoft IIS), etc. Deployment nodes can be nested.

Scope: A single software system.

Primary elements: Deployment nodes and containers within the software system in scope.

Intended audience: Technical people inside and outside of the software development team; including software architects, developers and operations/support staff.