

Architecture Project #1

- [Overview: Running App](#)
 - [1 System Context Diagram](#)
 - [1.1 Container Diagram](#)
 - [2 Current Deployment Diagram](#)
 - [3 One Month Plan](#)
 - [4 Long Term Plan](#)
-

Overview: Running App

Software Architecture Course

Project Part 1

A small sized company has a running application which allows users to track their running habits and get motivated by visualizing their progress.

Unfortunately, customers have been complaining about problems and the 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. Also , we will suggest a long term plan to have the best possible architecture available for the given requirements.

Pain Points

- 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

Primary Goals

- make the mobile app more responsive during a run
- have better data consistency on web dashboard
- immediately view runs on dashboard after a run ends

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: From the 4 suggested diagrams on C4 models, We will not be generating component and code diagrams. Instead we will create System Context and Container diagrams along with Deployment Diagrams , which will map our current platform to the physical cloud architecture, for current setup, one month attack plan and the final solution. This will help us visualize better the high level architectural changes proposed.

Tools: This document utilizes [Markdown files](#) for text notes and [PlantUML](#) for diagram generations.

The diagrams follow the [C4 Diagrams](#) model of visualization.

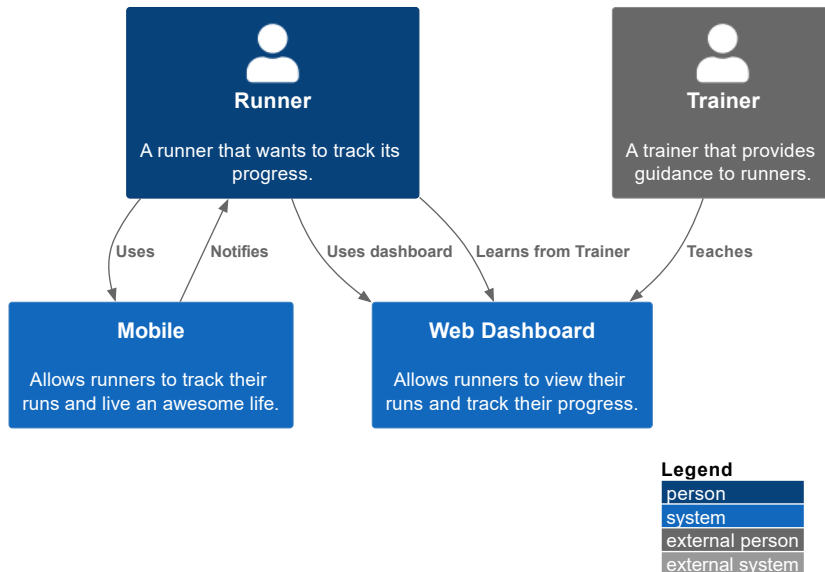
The text editor of choice was [VSCode](#) using the [PlantUML plugin](#) and the [NodeJS c4builder project](#).

Github:<https://github.com/SilvioAmaral/SolutionAchitecture>

trigger 1

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 beginner runners into their journey. They are considered External persons to our application, as they connect directly to the customer via external providers (zoom, webex). We will no longer be listing the trainers on our diagrams, as it is not a source of problems and therefore not the main focus of this optimization project.

Analysis of problems

As listed on our overview page, some of the main problems we have are:

- the app is not responsive during a run, especially trying to start and stop it

Let's consider that the responsiveness issue is due to the mobile app storing all data points during the run and performing one single POST with all data to the API once. This can cause the server to have spikes in utilization, both on service and database.

- some customers mentioned that their runs are disappearing from the web dashboard

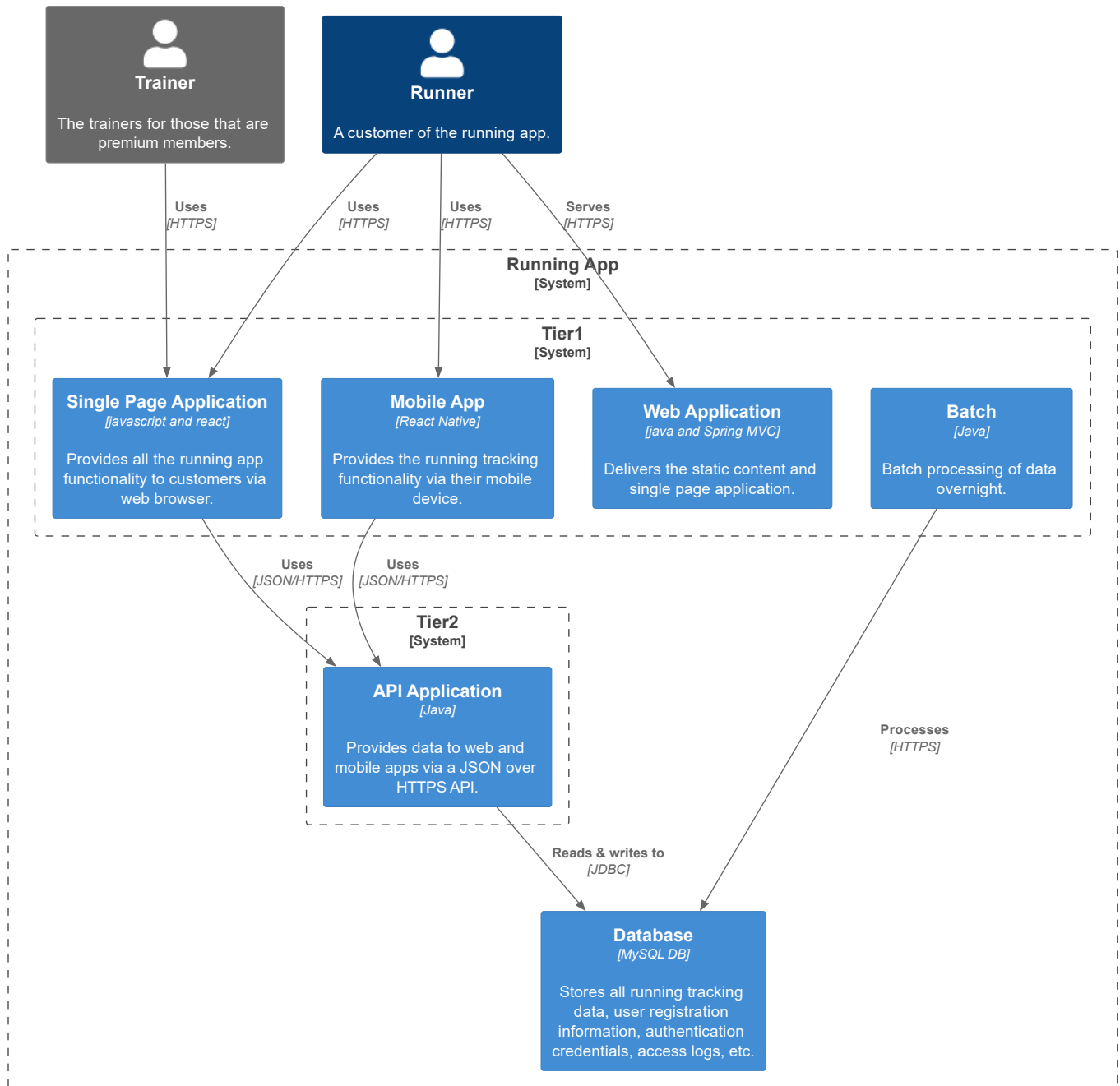
Due to the amount of data that the mobile app accumulates, sometimes the app crashes or sometimes the upload of data fails silently, therefore never showing on dashboard.

- when customers finish a run, it takes more than one day to view on the dashboard

Let's consider for this scenario that the data written on the DB after each run gets processed overnight on a batch process so it can be displayed to the user at the dashboard part of the application.

1.1 Container Diagram

\1 System Context Diagram\1.1 Container Diagram



Legend
person
system
container
external person
external system
external container

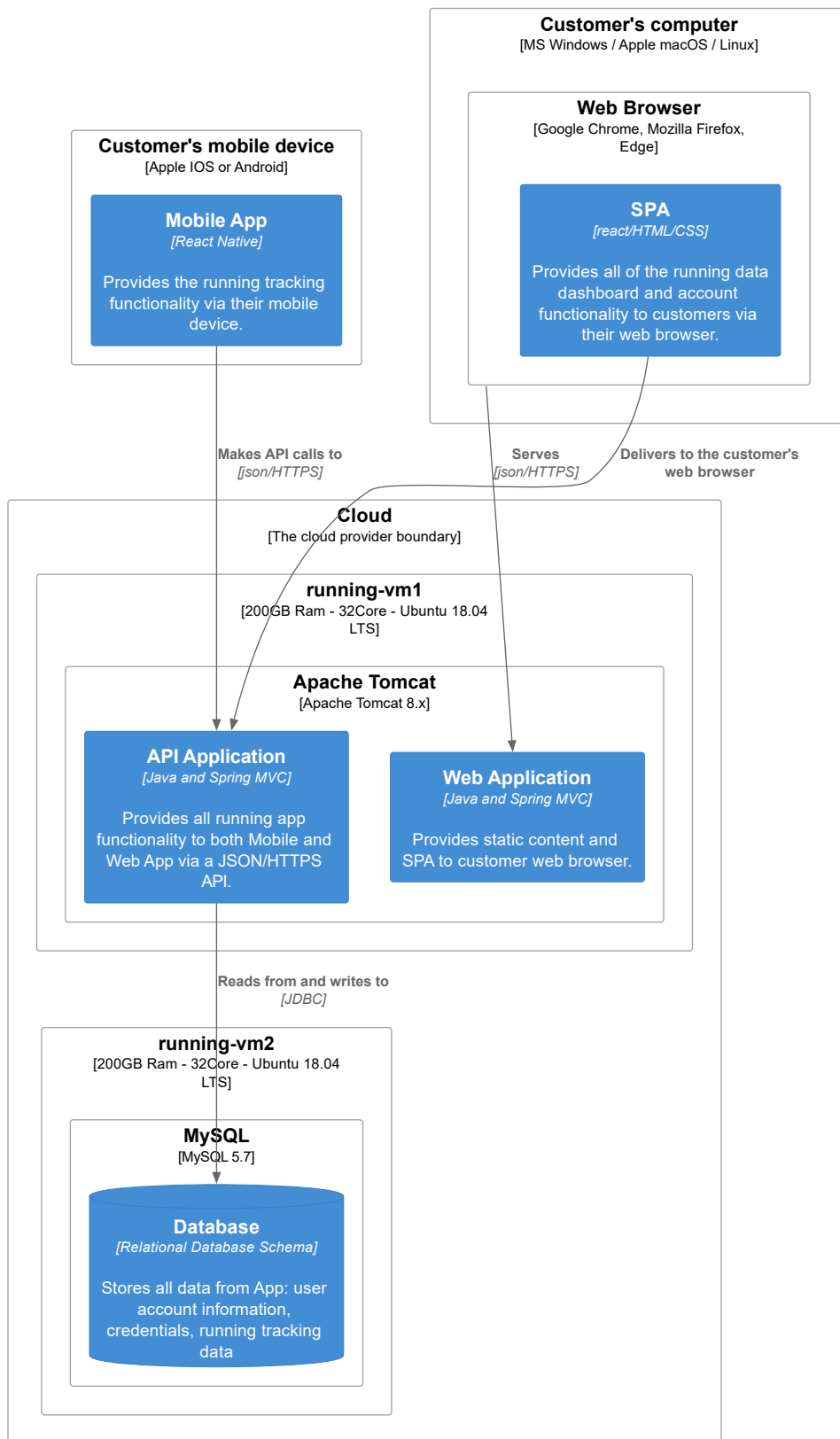
Level 2: Container diagram

Each part of our application is a two tier app, which means we have a code layer for presentation and one for business logic. Code quality is lower because of this, but this problem will be handled at a separate moment.

The presentation layer on mobile is coded on react native. The presentation layer on web is a single page application SPA on React/HTML/CSS and delivered via a web server to the client. Both presentation layers communicate to the API using JSON via RESTful calls over HTTPS.

2 Current Deployment Diagram

\2 Current Deployment Diagram



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 single API server instance. The executable files are hosted by the respective platform marketplaces.

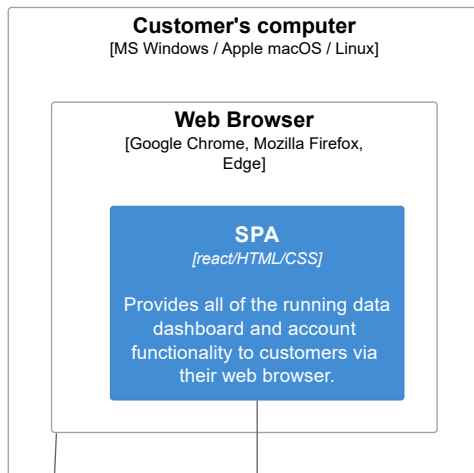
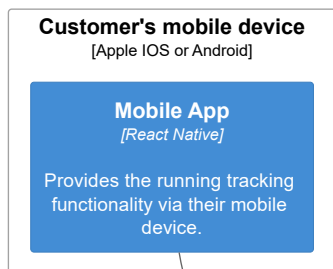
There are two VM instances of 200GB RAM and 32vCPUs. On the first instance , we have:

- 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 of the same size , and it runs a single MySQL database instance that is managed by our cloud services provider.

3 One Month Plan

\3 One Month Plan



Makes API calls to
[json/HTTPS]

Serves
[json/HTTPS]

Delivers to the customer's
web browser

Cloud
[The cloud provider boundary]

Content Delivery Network
[Will deliver the SPA and static content to the customer's web browser.]

Web Application
[Java and Spring MVC]

Provides static content and SPA to customer web browser.

Azure Load Balancer
[Apache Tomcat 8.x]

Load Balancer
[Java and Spring MVC]

Will provide the routing for new VMs that are being scaled up or down.

Serves
[json/HTTPS]

Serves
[json/HTTPS]

Serves
[json/HTTPS]

Dynamically Scaled VMs

api-vm1
[16GB Ram - 4Core - Ubuntu 18.04 LTS]

Apache Tomcat
[Apache Tomcat 8.x]

API
[Java and Spring MVC]

Provides all running app functionality to both Mobile and Web App via a JSON/HTTPS API.

api-vm2
[16GB Ram - 4Core - Ubuntu 18.04 LTS]

Apache Tomcat
[Apache Tomcat 8.x]

API
[Java and Spring MVC]

Provides all running app functionality to both Mobile and Web App via a JSON/HTTPS API.

api-vmN
[16GB Ram - 4Core - Ubuntu 18.04 LTS]

Apache Tomcat
[Apache Tomcat 8.x]

API
[Java and Spring MVC]

Provides all running app functionality to both Mobile and Web App via a JSON/HTTPS API.

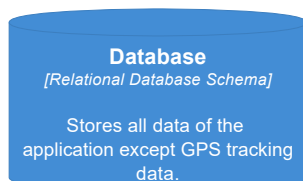
Reads/write data except tracking data
[json/HTTPS]

Reads/write data except tracking data
[json/HTTPS]

Reads/write data except tracking data
[json/HTTPS]

db-vm
[200GB Ram - 32Core - Ubuntu 18.04 LTS]

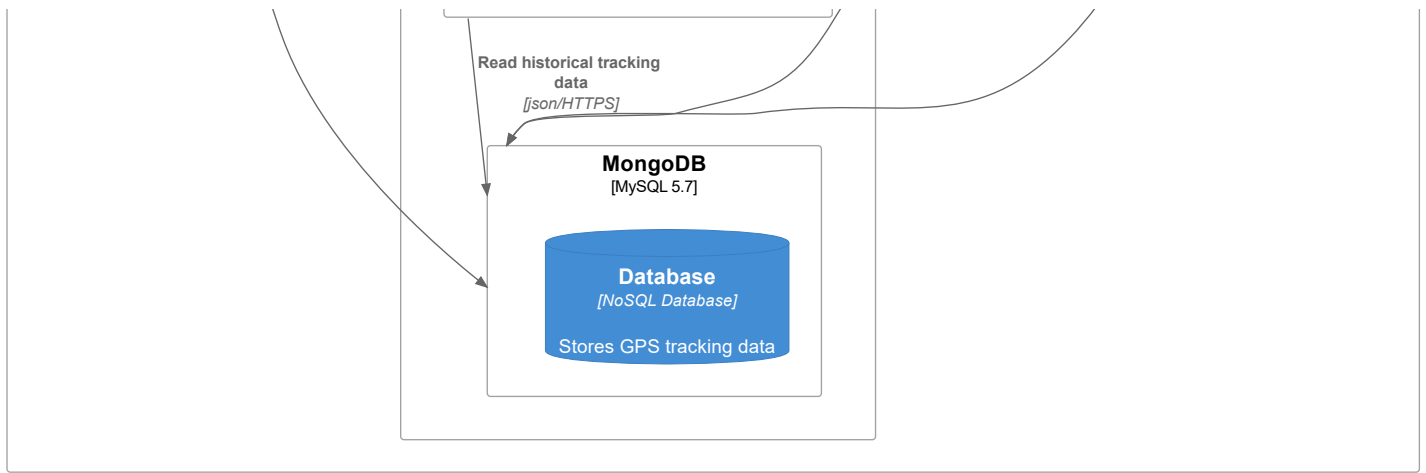
MySQL
[MySQL 5.7]



Write tracking data
[json/HTTPS]

Write tracking data
[json/HTTPS]

Write tracking data
[json/HTTPS]



Legend	
person	
system	
container	
external person	
external system	
external container	

One Month Attack Plan

Analyzing our users habits, we can notice that most of them follow conventional running times of early morning and end of day. This is causing our servers to be overloaded at these times and vastly underutilized the rest of the day, specially at night.

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

In order to maximize the elasticity of the resources, we will add a load balancing layer in front our API services and we will host this services into many more VMs to be able to scale up and down throughout the day. Due to this migration , we will be able to reduce the size of the actual VM Instance 1.

Also, the mobile app will be modified to send each individual point of the tracking data at a smaller intervals, therefore avoiding large chunks of data coming in at once. Therefore the "STOP RUN" functionality of our app will only carry the overall run directly to the dashboard tables processed at the client mobile phone.

For the Database server we will create a second Database responsible for write-intensive GPS tracking data that is provided on each run coming from the mobile apps via API. Each run will then have the entry to the overall data on the dashboard tables but if the client want to view the exact tracking of the run it will be linked to this data. This aims to remove the write overhead from the tables that are currently responsible for the tracking data separating it from the read part of the application.

Proposed Solution for main problems

As listed on our overview page, some of the main problems we have are:

- the app is not responsive during a run, especially trying to start and stop it

We will refactor the Mobile application to send each individual GPS tracking point to a new API endpoint, which in turn will save each individual point at a separate NoSQL database. On this same refactor we will have the mobile app create an overview of the run and when the customer hits STOP RUN , we will save the overview directly on the dashboard tables already processed, with a link to the NoSQL data points of the run GPS trail.

- some customers mentioned that their runs are disappearing from the web dashboard

With the above refactor we will reduce the peaks of CPU processing on both API and Database servers, therefore reducing the overall unavailability of the application. We will also get rid of the processing overnight batches and make the dashboard data faster to read.

- when customers finish a run, it takes more than one day to view on the dashboard

As we mentioned, by refactoring the mobile app to send individual points and an overview of the run at the end, we will no longer have to process the data overnight, therefore saving resources.

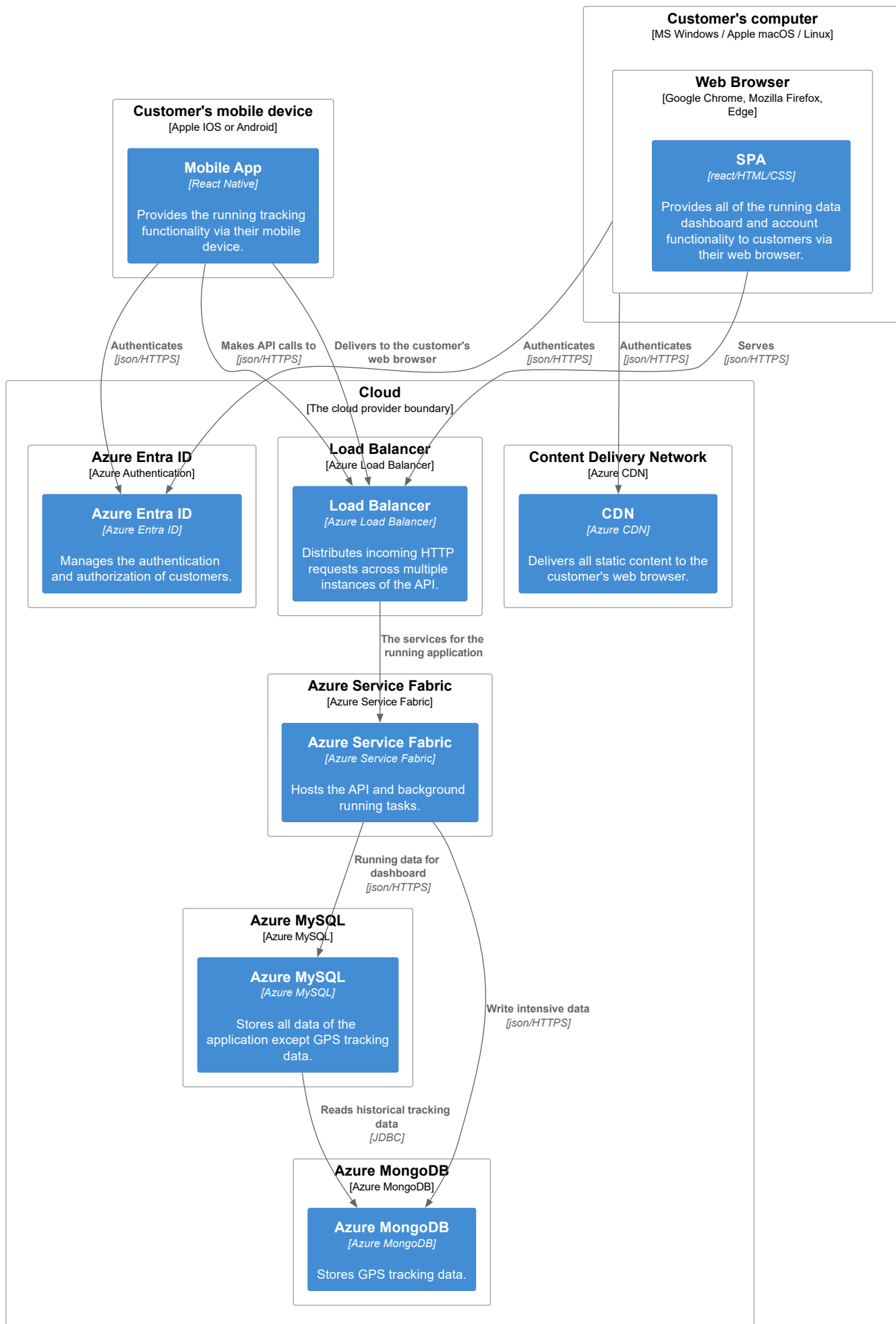
Proposed Solution for secondary problems

- Reduce cost of total cloud account

By adding a load balancer and distributing the service layer we are now able to scale the VMs and fine tune the amount of services provided , scaling up at peak times and down on reduced usage times.

4 Long Term Plan

\4 Long Term Plan



Legend

- person
- system

external person
external system
external container

One Month Attack Plan

Having solved the most immediate problems and having more time to work, we can actually fine tune this architecture and utilize more of the cloud resources. We will use Azure as an example but all cloud providers have the same services , only under different names.

For client authentication we can move all IDs to Azure Entra ID and manage all authentication and authorization directly at the cloud.

In order to maximize the elasticity of the resources, scaling up at peak hours and scaling down at night, we will convert all API layer into Azure Service Fabric services and no longer directly manage VMs. Due to this migration, we will be able to get rid of the overhead of managing the virtual machines and use our service directly on the cloud, while reducing costs even more. This layer will continue to be served behind a load balancer.

We will move the code towards a [CQRS](#) pattern, in which we laid a foundation on the previous step of this migration by introducing a NoSQL database for the run operations. We will continue to move all write operations related to run to the COMMAND part of the application and have all the QUERY part from the MySQL database to serve all the dashboard operations.