# Part 1: Serverless CQRS in Azure

#### + Event sourcing / Materialized views / Logic Apps / Event Grid / Cosmos DB

## Intro

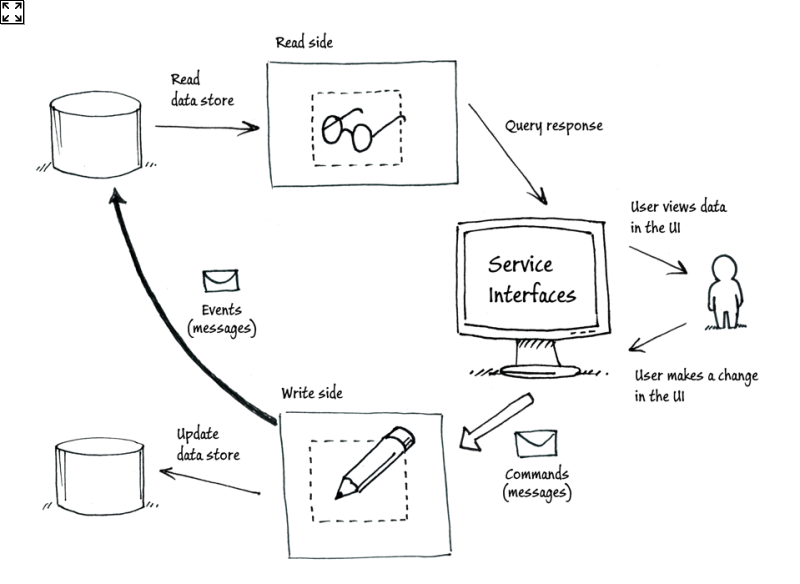
Keen to explore the realms of what is possible with Azure’s Serverless technologies, I decided there was no better way than to attempt to tackle a complex, modern software pattern: Command Query Responsibly Segregation, otherwise known as CQRS (with event sourcing & materialized views thrown in for the mix).

The goal: to be as service-full as possible and attempt to orchestrate the application with Logic Apps and without writing any code.

*I’ll provide a link to a working solution on github when the part 5 is complete*

## CQRS

In a nutshell, CQRS is a pattern that can be used where high throughput is required that would otherwise not be possible for traditional CRUD operations using a repository pattern. As a system, it looks a little something like this:



Many others have gone in to extensive detail on this subject so I suggest getting up to speed by reading the fantastic documentation on Microsoft’s Azure architecture site and Martin Fowler’s bliki.

<https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>

<https://martinfowler.com/bliki/CQRS.html>

## Event sourcing & Materialized views

These are two patterns that work very well with CQRS. The idea for event sourcing is that you store only events and use the event store as a mechanism, or log, of everything that has happened within a system. This event store can then be used to create materialized views (at any point in time) to reconstruct the current state of a query-able view from the read store.

<https://docs.microsoft.com/en-us/azure/architecture/patterns/materialized-view>

<https://docs.microsoft.com/en-us/azure/architecture/patterns/materialized-view>

## The Problem

Imagine a world where someone had the foresight to create some electric cars and a network of places to charge them. Perhaps you might want to call them superchargers, and maybe they would look like this: <https://www.tesla.com/supercharger>. The scenario I set myself was that I’m creating a global database to:

* Help people to find superchargers
* Allow the operations teams see usage
* And prioritise engineer services

The problem being solved isn’t really the point of this exercise and so without putting too much thought into it I came up with the following bounded contexts:



## Architecture

It is in fact possible to build some pretty complex workflows in Logic Apps and hence it should be possible to aim for a no-code target architecture like so:



For the purpose of this exercise I’m going to gloss over the apps, API management and authentication within this system as that is not the topic for this post.

If we extract just what’s required for our CQRS implementation it would look like this:



The next steps are to break down the system into its component parts.

# Part 2: Commands and Event sourcing