**Irish Senthilkumar 16342613 CT417 Assignment 1**

This report details the software architecture for Deliveroo. Deliveroo is an online food delivery company based in London but have operations around the world. Deliveroo works by first receiving the food order request from the customer. This order is forwarded to the restaurant, who prepare the order. Once the order has been prepared, a Deliveroo rider picks up the order and delivers it to the customer. To achieve this order lifecycle, Deliveroo uses a distributed, event driven software architecture.

Before using event driven architecture, Deliveroo were using a monolithic approach to their software architecture. They were using Ruby on Rails (a server-side web application framework) with PostgreSQL (an open source relational database management system) and Redis (an in-memory data structure store), as shown in Figure 1. For their initial growth phase, this was a good choice of architecture, since monolithic applications have good ease of deployment since they are packaged as a single, tightly coupled unit. This also means that the application’s source code is in a single location, which brings about a good ease of development.

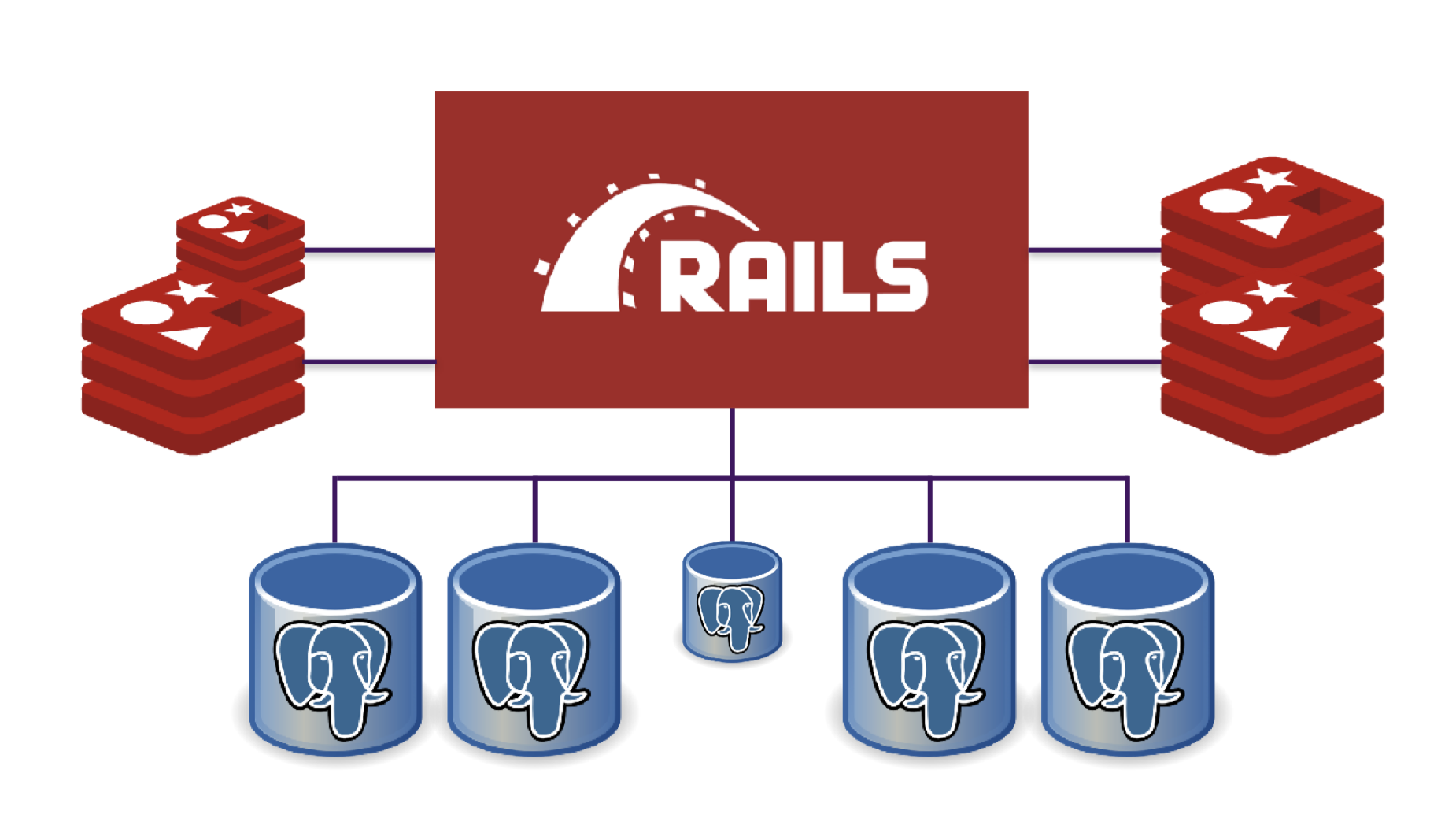


Figure 1: The monolithic software architecture previously employed by Deliveroo.

As Deliveroo started expanding their market share and spreading their services to more countries, this monolithic architecture was becoming obsolete. The previous architecture did not scale well with the addition of new application servers, which reduced performance. As a DevOps engineer during my work placement, I can also say that using continuous integration with monolithic applications brings about a considerable difficulty to deployment, since merge conflicts can arise, build/deploy times will significantly increase, and a single fault can bring the whole system down. These issues are illustrated in Figure 2. All of these issues compounded to decrease the development velocity to such an extent that Deliveroo were left with no choice but to switch over to an event driven architecture.

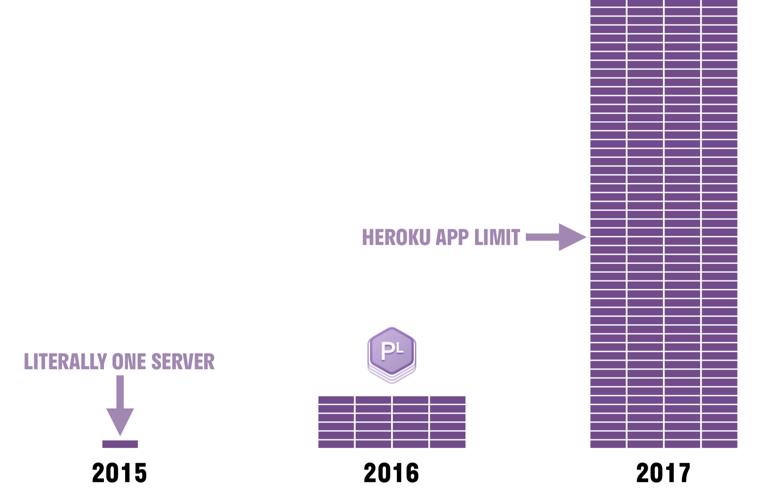
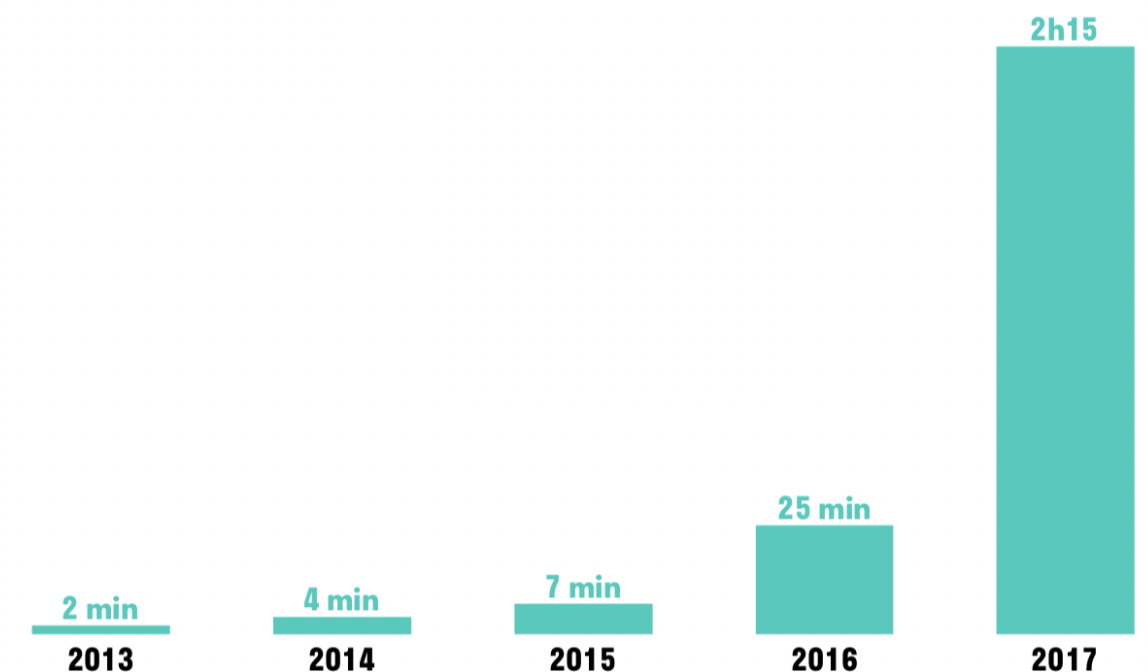


Figure 2: The growth of the number of application servers (left) and the application build times (right).

The event driven architecture is distributed, allowing for highly decoupled components to function together as a single system. To do this, the previous monolithic architecture was split into three classes of applications; domain services, edge services, and client-side user interfaces. These classes abide by the ‘twelve-factor’ rule, which is a set of twelve design constraints that includes having a single codebase with revision control which allows for many deployments, maximising robustness with fast start-ups and graceful shutdowns, and more rules which help to design software-as-a-service applications. These classes are connected using an event bus which aids in decoupling, because this allows the connected components to not depend on each other since they will only have to send the event along the event bus and not worry about the components that will respond to the event. This layout is illustrated in Figure 2.

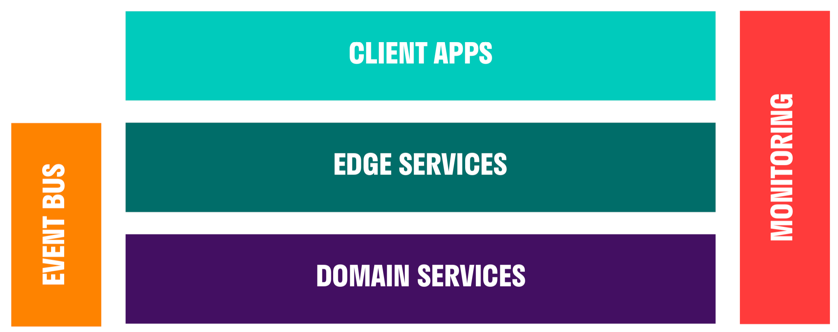


Figure 3: The overall layout of the EDA pattern used by Deliveroo.

Domain services handle the business logic. These services own a large part of the domain, and expose internal REST APIs with hypermedia. This just means that the REST API call will also provide hyperlinks with each response that link to related resources. These domain services can also use other domain service APIs.

Without edge services, the client needs to call the domain services multiple times due to the tight coupling between the client and back end services. This can bring about many problems, including the return of duplicate data due to multiple calls, and the error handling is up to the client. Using edge services eliminates these problems, by introducing a gateway service. This gateway service is essentially an API server which links to the back end microservices using the event bus. Now clients will only need a single call to the microservice, and the gateway takes care of error handling. The topology of an edge service is shown in Figure 4.

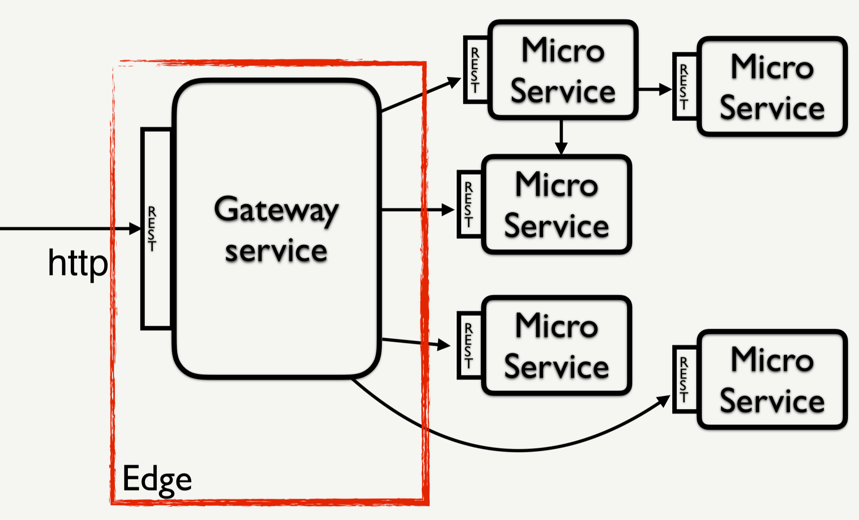


Figure 4: The topology of a simple edge service.

Each application keeps its own data store, which is non-accessible to any other application. All data is exposed as REST APIs with hypermedia. The domain services send events through the event bus when entities are created, updated or deleted. Since no embedded objects are allowed to be contained within these events along the event bus, this prevents the bus from becoming a source of data loss.

Event driven architecture due to its many strengths over the previous monolithic architecture, due to its distributed, asynchronous nature:

* The overall agility is high, due to the very loose coupling and asynchronous nature. Resources can freely move to the next task once their current unit of work is complete. Events can be queued and buffered, which prevents consumers from putting pressure on producers.
* The ease of deployment is very high, due to the autonomous nature of the components by loose coupling. Components do not depend on other components, since all they need are events. These components can therefore be individually updated, tested, and deployed.
* Performance is very high due to the decoupled, parallel processing capabilities of the software components.
* The scalability is high due since the components only perform on task each, so you can scale up just the components that are needed.

The event driven architecture model also has several weaknesses:

* The ease of development is low due to the asynchronous nature. Contracts are needed to specify communication rules between components, and unresponsive event brokers and event processors need to be handled.
* Software components could behave in an unpredictable manner, since a single event could trigger a range of routines, which only increases the more events are added into the system. This makes unit testing, component testing, and debugging difficult.

Monolithic architecture is quite outdated and many potential replacement software architectures exist, but Deliveroo chose event driven architecture for their application due to a variety of reasons. The main reason is due to the ease of deployment of EDA. Since the architecture was no longer tightly coupled and asynchronous, the components no longer depended on each other. This meant that a single fault could not bring the system down, while also dramatically speeding up the development velocity. Since Deliveroo is a rapidly expanding business to this day, the scalability of this architecture was appealing. If more processing power is needed in a certain part of the application, only the software component/s for that task need to be scaled, since the components are loosely coupled. EDA also boasts much better performance than monolithic architectures since the software components have parallel processing capabilities.

**References:**

* Distributed Computing:

<https://www.ibm.com/support/knowledgecenter/en/SSAL2T_8.2.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distd_comptg.html>

* Event driven architecture: <https://www.youtube.com/watch?v=XohG9yQe3Ps>
* 12 factor rule: <https://12factor.net>
* REST with Hypermedia: <https://reflectoring.io/rest-hypermedia/>
* Monolithic to distributed system:

<https://www.infoq.com/news/2017/03/deliveroo-monolith-distributed/>

* Deliveroo presentation:

<https://qconlondon.com/system/files/presentation-slides/qcon_-_the_distributed_pit_of_success_at_deliveroo.pdf>

* Edge services:

<https://cdn.oreillystatic.com/en/assets/1/event/281/Edge%20PaaS_%20A%20Netflix%20case%20study%20about%20edge%20service%20architecture%20Presentation.pdf>