CAB432 – Cloud Computing

Assignment Two:

Individual Task

The Queensland University of Technology

Semester Two, 2020

Student: Gavin Smith

Student Number: n10138196

25th October, 2020

Part One – Statelessness, Persistence and Scaling:

'Assignment Two' required us to develop a stateless application of our choosing. By building a 'Twitter feed', a user inputs a search query, which then utilizes the Twitter API and Natural API to listen for tweets containing the query and rendering the tweets and sentiment analysis to a front-end application for the user to view. This application was designed to be stateless, as it doesn't require, nor store, user data for future uses. Simply, the application receives a request from the user (query), and then provides the user with a response (tweets, and other data). The applications architecture can be seen below in Figure One;

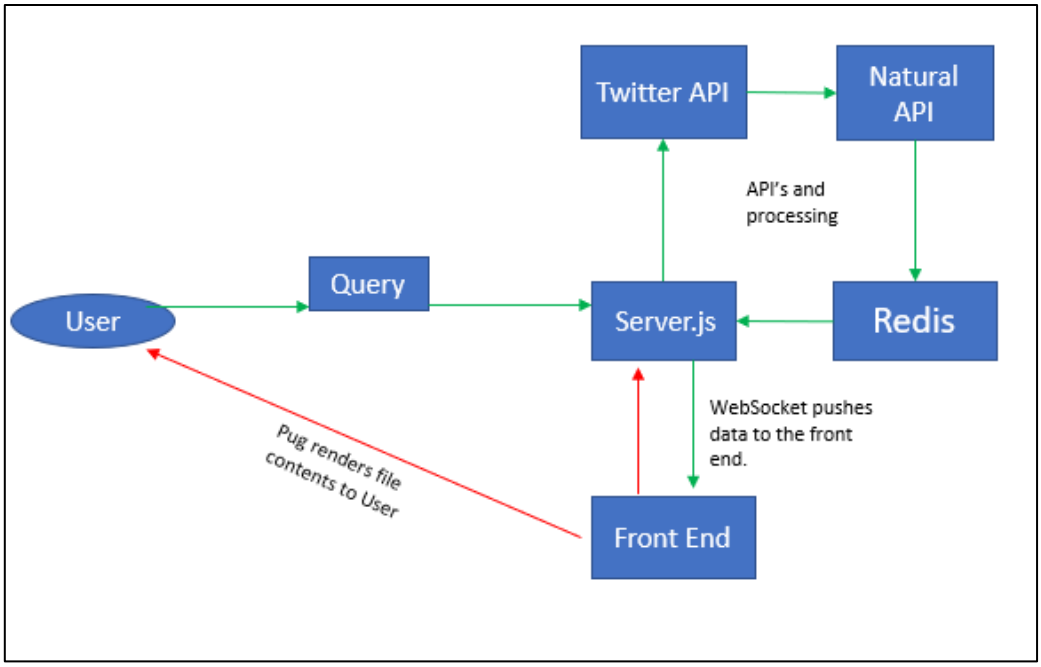

Figure One – Architecture of the Application, Local Environment

The choice to utilize Redis as the method for caching tweet data is an effective way of implementing the 'Persistence' component for this application. We can query Redis to display the data collected in a session by a user and presents the cached data in a format like that rendered in our applications front end.

Scalability provides an application with the resources needed to seamlessly serve the application when web traffic and computing demand grows. By running a 'T2 Micro Ubuntu' server to host our application, our hardware provisions are quite basic, allowing us to demonstrate our application can scale with ease when hit by a wave of internet traffic. By utilizing Postman as a means of initiating web traffic to our server in hopes of essentially *overwhelming* our hardware, this will trigger the scaling component of our server. Figure Two below further demonstrates the applications architecture at a broader level, and demonstrates the transition from Local to Cloud Environment;

Because I followed the 'Scaling' practical in setting up our Assignment Two environment, an alternative setup would include a lower threshold, and perhaps a trigger responding to an increase in 'Network Traffic' rather than 'Computational Thresholds' as the 'Computational Thresholds' were difficult to trigger due to our applications logical simplicity.

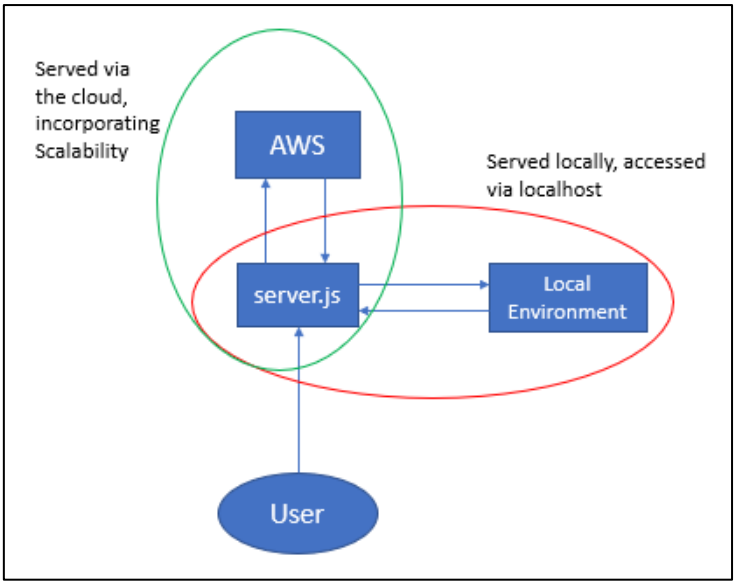

Figure Two – Broader View of the Application Architecture, and the Two Environments.

Part Two – Global Application:

A truly stateless application allows for any application to perfectly scale, meeting demand of any size. A globally successful version of this application would remain stateless, with the revised application architecture taking into account a more specialized persistence module. As applications become more popular and significant growth occurs, so do the computational power and memory needed to host the applications. While my application is relatively simple, if it involved, hypothetically, a software update, an 'Edge Cache' implemented by a software service such as Cloudflare or Amazon CloudFront would drastically lower the computational power needed to host such a large file. An Edge Cache would allow a user to download a large file, such as the software update which would then be Cached on the edge of the network, allowing super-fast download speeds to users in a location that shares the same cached memory. This lowers application latency, and server costs dramatically.

Furthermore, the application must be redesigned with costs in mind. As applications grow and scale, so do the hardware components needed to host such applications. An application that is perfectly efficient in terms of logic and computation would drastically lower hardware costs when scaled up significantly. This in turn lowers 'Server renting' on sites such as AWS dramatically, and thus saves you money. Therefore, with a 'greatly enhanced budget', I would hire a senior Software Engineer who specializes in algorithmic efficiency to further enhance my code and architecture build quality. Moreover, the API's used in our application should be further researched as tested as it is well documented that Twitter, regarding the free developer access, is given access to 500,000 tweets per month. If our application was to gain popularity and scaling increased 1000-fold, then we would need to consider upgrading our Twitter API to a paid plan, granting us access to unlimited tweets.

As costs grow when an application scales, monetizing the application is another change that could be added to recoup the costs over managing the application. Running ads or providing user access to a wider range of analytics or tools such as more complex sentiment analyses could be profit generating method with enormous potential. Also, a subscription service could be another method in recovering costs and provides the developer with a consistent income stream which in turn can be put back into the application.

Another cost factor that needs to be considered with a scaling application is ‘inhouse’ versus ‘outhouse’ server storage. Research indicates that AWS are the gold standard in server storage, however there are alternatives. It has been shown that AWS on-demand instances were far more expensive than traditional server-based infrastructure by nearly 300%. Furthermore, contracting physical servers to meet your individual demands have shown to be roughly 250% cheaper than AWS reserved instances when rented over the same duration (CloudHealth, 2018).

However, while AWS is an arguably expensive service when compared to renting your own physical servers, how do they stack up against the competition? Figure Three below shows the costs associated between AWS, and Microsoft Azure. It is shown that AWS is still significantly more expensive than other cloud-based services.

ON DEMAND AZURE VS AWS PRICING

WINDOWS OS (AUGUST 2018)

TYPE	vCPU	MEM.	AZURE	AWS
General Purpose	2	8GB	$0.4990	$0.6680
	4	16GB	$0.5970	$0.8560
	8	32GB	$1.1940	$1.7120
Compute Optimized	2	4GB	$0.5630	$0.6570
	4	8GB	$0.7260	$0.8340
	8	16GB	$1.4510	$1.6680
Memory Optimized	2	16GB	$0.6250	$0.7160
	4	32GB	$0.8500	$0.9520
	8	64GB	$1.7000	$1.9040

Figure Three – AWS Prices versus Microsoft Azure (CloudHealth, 2018).

However, when referring to our cloud application, it seems as if a physical server may be the most cost-effective method of serving our application. But, when factors such as application longevity and popularity are taken into account, physical servers may not be the best option. If our application was to lose interest amongst users, an AWS server’s ability to scale down along with their prices make it a very attractive solution. Purchasing your own servers can be expensive, particularly if you’re unsure how the application is going to perform in the long run and with AWS inbuilt software, navigation is easy and setup costs are virtually nonexistent. As shown above, it does however make sense to consider alternative solutions, and going forward I would implement an Azure based server to run my Application, as prices are lower and the flexibility unoffered by purchasing physical servers make Azure a highly attractive cloud based service.

Final Questions:

There are many high-level security threats that my application presents. Weak access controls, and application hosting on a public AWS account means that any CAB432 moderator can access my storage, data, and application. Furthermore, because the application is stateless, client data is not an issue we need to consider. Because of our stateless nature, a Denial of Service Attack (DoS, or DDoS) is arguably the biggest threat our application faces, whereby a system, or multiple systems, disrupt an application by flooding the server with traffic. However, cloud services such as AWS offer solutions to various cyber-attacks, such as DDoS, with security measures such as ‘AWS Shield’ protecting applications and their resources.

References:

CloudHealth. (2018, November 19). *A Look At Azure vs AWS Pricing In 2018-2019*. Retrieved from CloudHealth by VMWare: https://www.cloudhealthtech.com/blog/azure-vs-aws-pricing