AWS SERVERLESS CONTENT MANAGEMENT SYSTEM

Auckland University of Technology research and development project for Bulletproof (March to November 2016)

Adam Campbell (1311607) Miguel Saavedra (13826904) John Cave (1324776) Chris Treadgold (1399164) Waqar Hussain (Supervisor)

1. RATIONALE

The team at Bulletproof wanted to demonstrate the power and flexibility of the Amazon Web Services (AWS) platform by developing a Content Management System (CMS) that runs entirely on AWS, without depending on any capabilities that aren't native to the platform.

A system such as this has never been attempted before in open source software. Through this demonstration of the strength in the platform, they were hoping to attract potential clients into adopting AWS services, therefore expanding Bulletproof's client base. By working with AUT, Bulletproof could create a Minimal Viable Product (MVP) of the concept without expending money or time.

2. OBJECTIVES

Bulletproof was flexible about the exact form that the content management system should take. Some of the high level requirements stipulated by Bulletproof included:

- The CMS would run only upon Amazon's managed cloud services.
- The CMS would not require a constantly running server to run.
- The CMS would be able to be used at minimal or no cost.
- The CMS should be well documented.

With these objectives, we were tasked with making a CMS to produce a corporate website or personal blog that would fulfil the basic needs of this use case.

3. METHOD

INITIAL REQUIREMENTS

Initially, Bulletproof did not know what they wanted in regards to CMS functionality, nor did they have any specific expectations. In order to create a starting point for our efforts, we conducted several time boxed interviews with Bulletproof, in order to extract functional requirements for the CMS. The interviews had an open structure where we added ideas to a whiteboard, and Bulletproof accepted or rejected them as they saw fit.

METHODOLOGY

In order to produce such a system with little initial direction and projection, we opted to go with an Agile approach, so we could adapt to frequently changing requirements and feedback. Our client suggested that we use SCRUM, which was agreed upon between our supervisor and ourselves. We decided on two week sprints so that we could get a reasonable amount of work done in a single iteration, whilst still getting feedback frequently.

Our level of adherence to the specifics of the SCRUM methodology has not been 100%. In particular, at the beginning of semester two we opted to do away with daily stand-ups. We were not finding them useful as we generally could not do them in person due to university schedule constraints, and we were communicating consistently regardless.

We found SCRUM's sprint retrospectives and sprint reviews to be highly valuable (Cockburn, A. 2006). Sprint reviews allowed us to see what Bulletproof liked and disliked in our project progress, as well as letting us constult with the Bulletproof team regarding challenges we were facing using AWS Services. Sprint retrospectives frequently revealed problems with our methods and allowed us to improve them as necessary.

TOOLS

The tools we used consistently throughout the entire project were Facebook Messenger and JIRA. Over the course of the project we also tried using several other tools including Trello and Slack. However we dropped these as keeping up to date with what was going on with each tool was complicated and didn't seem to offer us much value. Facebook provided a convenient platform for the whole team to facilitate live communication.

TESTING

Due to AWS lacking an automated testing framework, our testing practices were limited to manual unit testing. Integration testing also occurred when merging code; this consisted of more unit tests done after a merge. The client also performed end user testing at the end of each sprint. In retrospect we realise that more time and effort should have been put into not only testing as a practice, but also formally defining tests. We also peer reviewed our code often to ensure code quality.

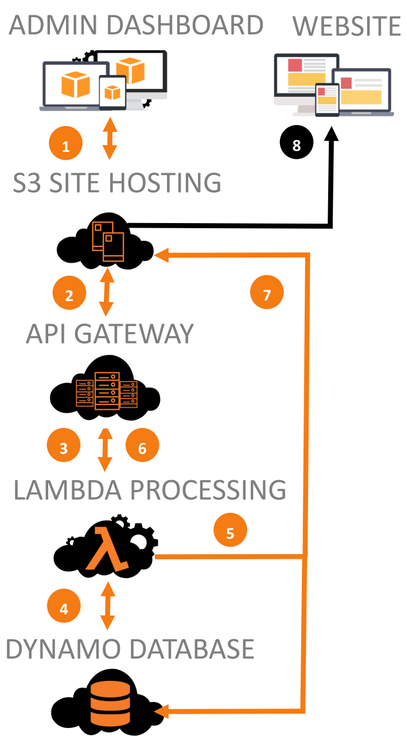
4. ARTIFACTS & DELIVERABLES

ARCHITECTURE MODEL

For the AWS Lambda CMS, it was important to Bulletproof that the CMS be low-cost or free to run, and as such, we produced a system architecture which facilitates said requirement. Much like traditional web applications, we required a backend to manage the processing of requests, data and produce responses. However, unlike traditional web applications, the Lambda backend does not work quite the same.

As Lambda runs only when required, costs associated with Lambda are calculated on 100 millisecond portions of Lambda function run time (Amazon Web Services n.d.). Our intention was to produce one Lambda backend function which handles all incoming requests, data manipulation, and response protocol in order to mitigate the requirement of bulked Lambda functions, Lambda chaining/piping, and concurrency costs.

Our team, understanding the difficulties of Lambda after considering several architecture solutions, eventually modelled our solution with a single Lambda function backend.



PROCESS

1 | Admin accesses the admin dashboard stored on the S3 cloud storage unit

2 | Admin requests are sent to the CMS through the API Gateway

3 | API Gateway receives the request, and routes it to LAMBDA for processing

4 | LAMBDA performs authentication of the user against Dynamo database

5 | Performs the changes requested, taking data from S3 or Dynamo database

6 | A status message about the requested changes are sent back to the admin dashboard

7 | These changes are pushed to the S3 hosted site

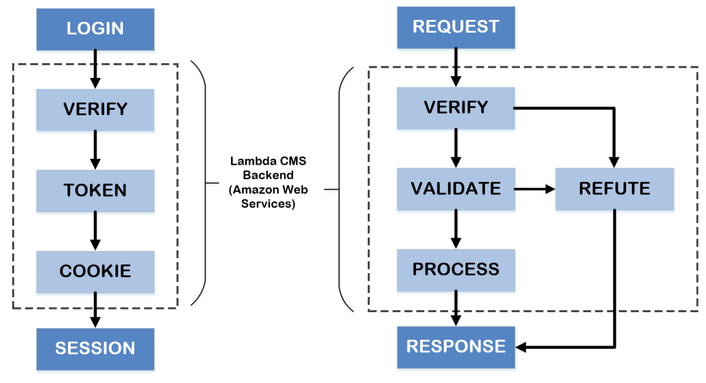
8 | The visitor sees the changes made and appreciates the content

SECURITY MODEL

As the AWS Lambda CMS is a custom built solution, it required us to build a custom built security model. During our training of AWS services, we did find that AWS do provide services for end user logins and session management for applications hosted on AWS, however, these services were not only impossible to implement within our CMS solution (Amazon Web Services n.d.), but completely lacking in the required features and User management functionality required by Bulletproof.

Yet again, our team went to the drawing board to produce a custom Security model which would facilitate several requirements:

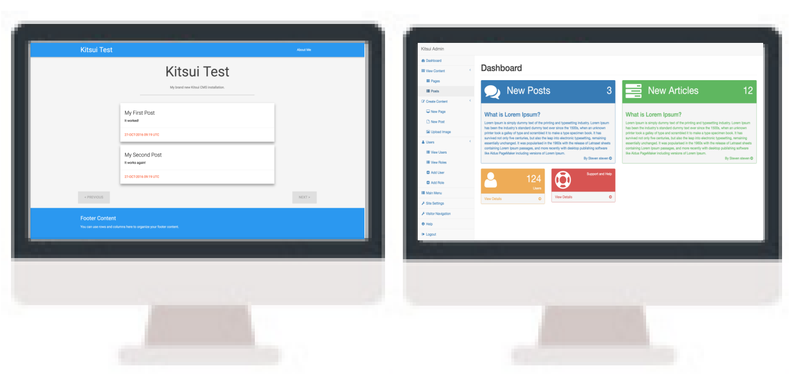
- User sessions
- Token production and evaluation
- Cookie production and evaluation
- User assigned permissions (roles)

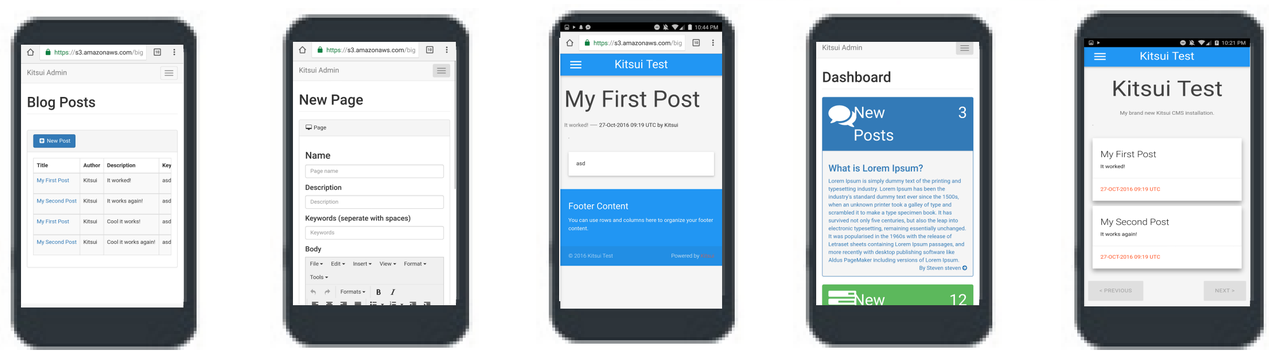


Our custom security model was designed and constructed similarly to the security model of existing CMS solutions that we examined.

PRODUCED DELIVERABLES

- [x] Online user guide
- [x] Online developer documentation
- [x] CMS install script
- [x] CMS uninstall script
- [x] Private web based admin dashboard
- [x] Public blog/corporate website
- [x] CMS source code repository (GitHub)





5. AREAS OF DIFFICULTY

BACKEND PROGRAMMING LANGUAGE

We initially chose Node.js as the language for our backend, as some of our team had experience using it. However, we found that it did not produce useful error messages, and hence would impede the development and integrity of the product. So we were forced to switch languages to Python as it had comparable speeds (GitHub n.d.) at the end of the first sprint. None of the team had much experience developing in Python, so the team reviewed online tutorials for Python.

AMAZON WEB SERVICES

None of the team had any experience working with Amazon Web Services or cloud platforms. Bulletproof provided an online course in AWS basics to help us get started, but we found that the majority of the training was not applicable to our project. This meant we planned to use some AWS services which we discovered did not meet our requirements, and this resulted in wasted resources. With the help of the Bulletproof team we found alternative solutions.

TECHNICAL REQUIREMENTS GATHERING

This project was conceived as a proof-of-concept to demonstrate the viability of a low to no cost Content Management System on the Amazon Web Services platform. This, along with the unique project concept, meant that many of the product design decisions were left to us as a team. Considering the fact that we had no experience with AWS and only one of us had experience with running websites, this meant that we had to define requirements based on research of existing CMS platforms.

REFERENCES

Amazon Web Services. (n.d.). AWS Lambda | Pricing. Retrieved from https://aws.amazon.com/lambda/pricing/

Amazon Web Services. (n.d.). AWS User Authentication & Mobile Data Service | Amazon Cognito. Retrieved from https://aws.amazon.com/cognito/

Cockburn, A. (2006). Agile software development: the cooperative game. Pearson Education.

GitHub. (n.d.). GitHub - berezovskyi/lambda-test: Test (pseudo) AWS runtime startup time. Retrieved from https://github.com/berezovskyi/lambda-test