University of Pretoria
Software Engineering - COS 301

Amazon Dash Software Requirements Specification

Contrapositives
May 2019

Authors:

Brendan Bath	**u16023359**
Musa Mathe	**u15048030**
Jessica da Silva	**u16045816**
Natasha Draper	**u16081758**

Contents

0.1 Glossary . . . 1

1 Introduction **2**
1.1 Objectives . . . 2
1.2 Scope . . . 2

2 Domain model **3**

3 User characteristics **4**

4 Functional requirements **5**
4.1 Use cases . . . 5
4.2 Requirements . . . 9
4.3 Subsystems . . . 10
4.3.1 Metrics Subsystem Requirements . . . 10
4.3.2 Alert subsystem . . . 10
4.3.3 Services subsystem . . . 10
4.3.4 OperationCalculation subsystem . . . 11

5 Quality requirements **12**
5.1 Usability . . . 12
5.2 Security . . . 12
5.3 Performance . . . 12
5.4 Testability . . . 12

6 Improved Quality requirements **13**
6.1 Q1. Usability . . . 13
6.2 Q2. Performance . . . 13
6.3 Q3. Security . . . 13
6.4 Q4. Testability . . . 13

7 Trace-ability matrix **14**

8 Architectural Design Requirements **15**
8.1 3-tier . . . 15
8.1.1 Presentation layer . . . 15
8.1.2 Logic layer . . . 15
8.1.3 Data layer . . . 15
8.2 Component-based . . . 15
8.3 Plug-in . . . 15

9 Deployement Diagram **17**

10 Constraints **18**

11 Technology decisions **19**
11.1 Vue 19
11.2 Vuetify 19
11.3 Python(backend) 19

0.1 Glossary

AWS	Amazon web service
AWSD	Amazon Web Service Dashboard
client(s)	The user using the amazon web services dashboard
system	The Amazon web service dashboard
Vuetify	is a semantic development framework for Vue.js

1 Introduction

This document outlines the Software Requirements and Design Specication for the Amazon Dash project.The vision of the project is to improve upon the existing AWSD to allow a client to easily view and manage the Amazon Web services, and then receiving an alert feedback when the costs of using the services has been exceeded.

1.1 Objectives

The AWSD that is provided by Amazon is cluttered and cumbersome to navigate. This may pose a challenge to novice cloud-hosting clients who may not know what services are best for their requirements and may not deploy services in the best manner which minimises the costs and maximises throughput. Another issue which the client encounters is tracking down an existing service and the client can also be unsure of which region a particular service is deployed at. The main objectives of the project is to provide a global view of all services currently running on the client's AWS account, and to allow the client to record metrics which will be used to ensure that the client is within their budget and to ensure that client's servers are always running optimally without overspending on bulky servers which are barely doing any work.

1.2 Scope

The system will provide an interface for deploying, configuring and maintaining AWS services. The system will visually report on all services running across all AWS regions. Security and AWS Identity and Access Management will be safeguarded as much as possible. The system shall also analyse computing resources, and report instances that are over- or under-utilised.
The system will not alter the user's current AWS service configuration without their approval, but merely make recommendations that may increase throughput and reduce costs. The client can then choose to implement these recommendations.
The benefits of the system is that it will bridge the division between novice cloud computing users and the prolific and complex range of services that AWS provides. The system is meant to increase throughput and reduce costs of services running on the user's AWS account by monitoring metrics and recommending more optimal solutions. The system will also simplify the AWSD interface for ease of use in order to make AWSD more user-friendly and clear.

2 Domain model

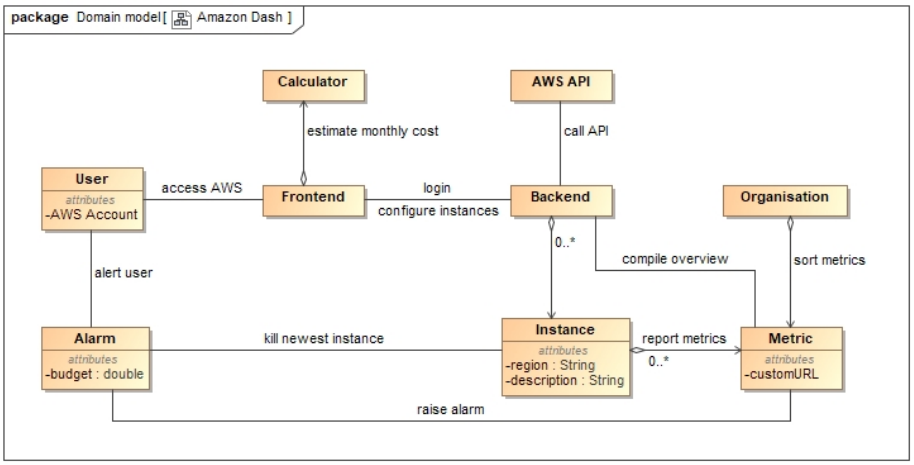

Figure 1: Amazon Dash domain model

3 User characteristics

The client can be an individual, company or organization who will use the Amazon Web Services dashboard to access the cloud-based services offered by Amazon such as global compute, storage, database, analytics, application, and deployment services that will help to manage their entire account, setup and manage their own services as well as to discover new services and build new applications . AWSD will make it easy for the client to find new AWS services, configure services, view service usage, and also to update user groups to building applications to troubleshooting issues and so much more it.
The client is assumed to have rudimentary cloud computing knowledge and experience, primarily regarding the type of services they require. The client is assumed to have technical skills which will allow them to deploy and maintain AWS services which may include rudimentary networking, database and development skills.

4 Functional requirements

4.1 Use cases

UC 1 Provide optimal services to the region the service should be deployed.

UC 2 Integrate existing instances and services on the client's AWS account.

UC 3 Provide a 'kill-all' switch to terminate all running services.

UC 4 Provide an administrative overview of AWS services to display metrics for each orgranisation.

UC 5 Client define their own metrics URL and pull from them.

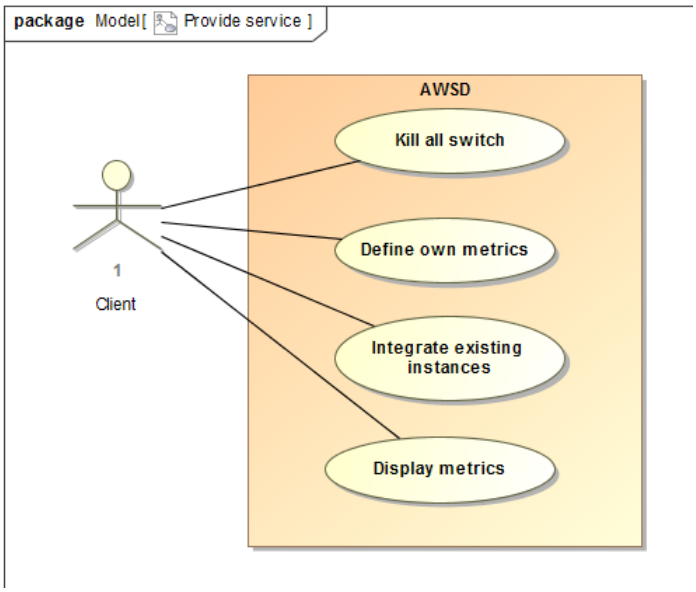

Figure 2: Product service use case

UC 6 Alert the client when the total cost of AWS services exceed the client's budget.

UC 7 Alert the client of configurations affecting throughput and cost of running services.

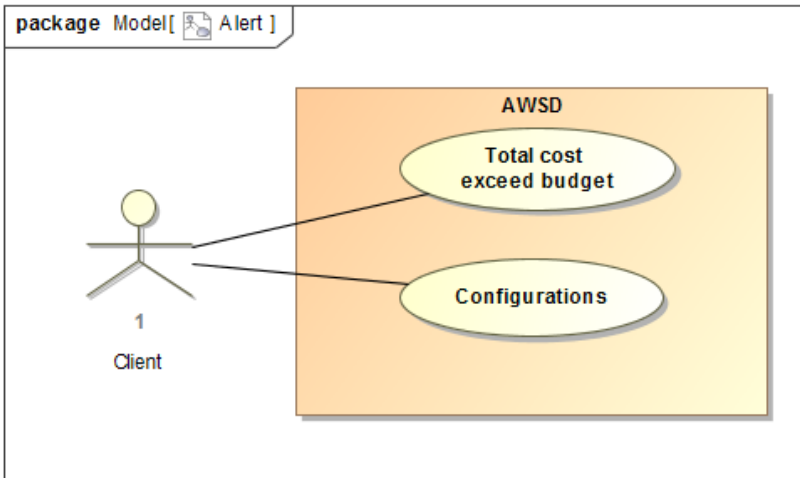

Figure 3: Alert usecase

UC 8 Collect different metrics.

UC 8.1 Collect the metrics of service downtime.

UC 8.2 Collect the metrics of network activity.

UC 8.3 Collect the metrics of CPU usage.

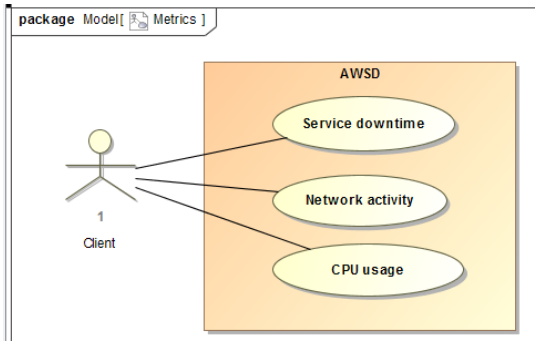

Figure 4: Metrics usecase

UC 9 Calculator to generate an estimated monthly cost.

UC 10 Calculate estimated cost based on previous service usage.z

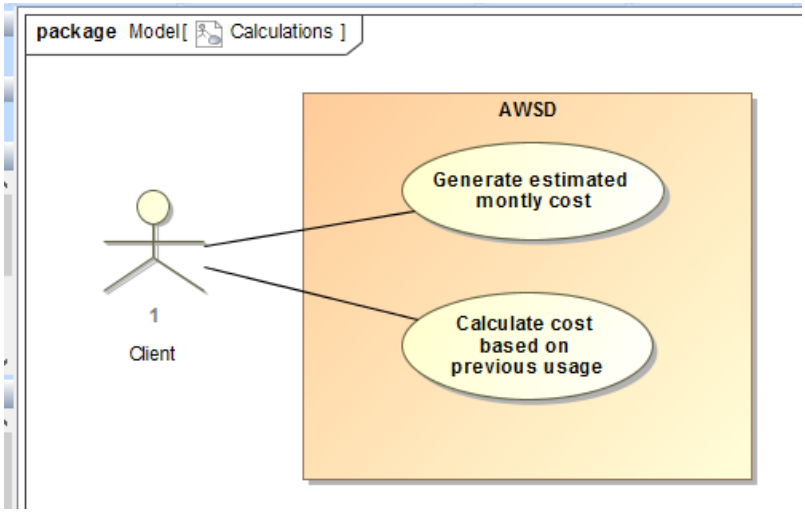

Figure 5: Calculations usecase

4.2 Requirements

R1 The system will provide a service to a client.

R1.1: The system will recommend the most optimal services with regards to the region the service should be deployed.

R1.2: The system will provide a way of integrating existing instances and services on the client's AWS account.

R1.3: The system will provide the user with a 'kill-all' switch, which when invoked will terminate all running services on the user's entire AWS account.

R1.4: The system will provide an administrative overview of AWS services, which will display summarised metrics for each orgranisation.

R1.5: The system will allow a client to define their own metrics URL and pull from them.

R2 The system will collect different metrics of services.

R2.1: The system will collect the metrics of service downtime.

R2.2: The system will collect the metrics of network activity.

R2.3: The system will collect the metrics of CPU usage.

R3 The system will alert the client.

R3.1: The system will alert the client when the total cost of AWS services exceed the client's budget.

R3.2: The system will alert the client of any configurations that adversely affect throughput and cost of running services.

R4 The system will perform different calculating operations.

R4.1: The system will include a user-friendly calculator which will allow users to generate an estimated monthly cost based on currently running services and propose new services to indicate to the user if their budget may possibly be exceeded.

R4.2: The system will utilise a smart cost exploration tool which will provide an estimated cost based on previous service usage. The application will make recommendations based on previous usage to reduce costs.

4.3 Subsystems

4.3.1 Metrics Subsystem Requirements

The metrics subsystem will be responsible for different types of metrics.

R2 The system will collect different metrics of services.

R2.1 The system will collect the metrics of service downtime.

R2.2 The system will collect the metrics of network activity.

R2.3 The system will collect the metrics of CPU usage.

4.3.2 Alert subsystem

The alert subsystem is responsible for alerting and sending notifications to the client.

R3 The system will alert the client.

R3.1 The system will alert the client when the total cost of AWS services exceed the client's budget.

R3.2 The system will alert the client of any configurations that adversely affect throughput and cost of running services.

4.3.3 Services subsystem

The services subsystem will have sub-services which will be used by the client.

R1 The system will provide a service to a client.

R1.1 The system will recommend the most optimal services with regards to the region the service should be deployed.

R1.2 The system will provide a way of integrating existing instances and services on the client's AWS account.

R1.3 The system will provide the user with a 'kill-all' switch, which when invoked will terminate all running services on the user's entire AWS account.

R1.4 The system will provide an administrative overview of AWS services, which will display summarised metrics for each orgranisation.

R1.5 The system will allow a client to define their own metrics URL and pull from them.

4.3.4 OperationCalculation subsystem

The OperationCalculation subsystem is responsible for performing arithmetic calculations for the calculating tools which will be used by the client.

R4 The system will perform different calculating operations.

R4.1 The system will include a user-friendly calculator which will allow users to generate an estimated monthly cost based on currently running services and propose new services to indicate to the user if their budget may possibly be exceeded.

R4.2 The system will utilise a smart cost exploration tool which will provide an estimated cost based on previous service usage. The application will make recommendations based on previous usage to reduce costs.

5 Quality requirements

5.1 Usability

- The system will provide a user friendly interface which will display all the services in a more simple and less confusing manner by grouping services to make it easy for the client to locate the services they need . We will make use of Vuetify material design framework to design the interface since it provides graphical elements which will improve the design of the system so that the client can easily interact with the services without having to struggle to navigate through the system for a successful ease of use.

5.2 Security

- Two-factor authentication will be used for the client to get access to their account .

- To ensure that the data which we send through the network is also protected a combination of the AES and RSA cryptography algorithms will be used because RSA will be used for authentication but since it is not efficient in encrypting data we will then add the AES for maximum speed for encrypting data. No sensitive data will be in plain text, both on the Back End and Front End.

5.3 Performance

- We will use Gatling in order to measure the performance of variety of running services.

- The time to generate the estimated monthly cost of service will not exceed 4s thus we will use efficient computations algorithms to test the cost of time to calculate in order to ensure that the calculator operates efficiently.

- We will use Silk Performer to ensure that applications and server maintained when client has many services running.

5.4 Testability

- We will make use of automated tests to test that each of the services implemented are tested along through development cycle by making use of LINT , JEST for unit testing the pre-conditions criteria which must be met. For integration testing we will make use of the travis ci for testing the dependencies among the services which need to work with each other.

6 Improved Quality requirements

6.1 Q1. Usability

Q1.1 The application should provide an easy graphic user interface which is easy to navigate through for the client to understand the flow of the application.

Q1.2 The application should provide a user friendly calculator which will allow client of the system to generate an estimated monthly cost.

Q1.3 The application should only display relevant services specific to that particular client's account.

Q1.4 The application should provide a notification when client exceeds the total costing of all services.

Q1.5 The application should provide an alert should a server not respond.

6.2 Q2. Performance

Q2.1 The application should take min of 60s to activate a running instance.

Q2.2 The application should take min of 60s to deactivate a running instance.

Q2.3 The application should not take more than 1.2 mins to kill all running instances.

Q2.4 The application should take 1min to get all the instances.

Q2.5 The time taken for user authenticating should not take more than 30s

6.3 Q3. Security

Q3.1 The clients passwords should be hashed first before they are stored.

Q3.2 The application should use two-factor authentication for the client to get access to their account .

Q3.3 The application should make use of google authentication to authenticate.

6.4 Q4. Testability

1. Q4.1 The application should provide component services which are testable through unit tests.
2. Q4.2 The application should provide a test for integrating the subsystems of the application by performing an integration test to ensure the subsystems communicate accordingly.

7 Trace-ability matrix

	Metrics subsystem	Services subsystem	OperationCalculation subsystem	Alert subsytem
R1				
R1.1		X		
R1.2		X		
R1.3		X		
R1.4		X		
R1.5		X		
R2				
R2.1	X			
R2.2	X			
R2.3	X			
R3				
R3.1				X
R3.1				X
R4				
R4.1			X	
R4.2			X	
Q1				X
Q2		X		
Q3			X	
Q4	X		X	

Figure 6: Trace-ability matrix

8 Architectural Design Requirements

We are using a 3-tier + Component-based + Plug-in Architecture

8.1 3-tier

8.1.1 Presentation layer

This will be the top level of the application which will provide graphic user interface by displaying information to the user in a more organized and simple manner in which the user can easily understand and navigate through application. This layer will serve as a means to communicate with the user by displaying list of services, performing calculations, and displaying the metrics of the instances.

8.1.2 Logic layer

This layer will serve as a middle dynamic content processing which will be responsible for coordinating the application, and making logic decision and performing calculations. We will be using python for the logic and flask to expose the logic as a restful API.

8.1.3 Data layer

The data layer will provide an API to the application layer that exposes methods of managing the stored data like keys and login details without exposing or creating dependencies on the data storage mechanisms.

8.2 Component-based

This will provide flexibility which will ensure separation of concerns which will help for defining, implementing and composing loosely coupled independent components into our system through out the development cycle to allow the client benefit both for short and long-term process since it will provide us continuous deployment of features since a component(e.g displaying metrics) will be developed as a separate feature on it's own then later will be converted to a service which then will be added to the running application. It also give us an opportunity to substitute a component which needs to be replaced by another component with either an updated version or an alternative without breaking the system in which the component operates.

8.3 Plug-in

This enable us to add features and functionality to our application by just installing a new bundle by adding multiple instances of a particular type of module that provides a well-defined unit of functionality to our application thus allowing for customization. Since we need to pull data from other third-parties

services this will give us an ability to implement plug-in functionality using some form of shared libraries, which we can get data dynamically at run time thus extending our application e.g pulling data via a plugin using boto3. This will also reduce the size of our application.

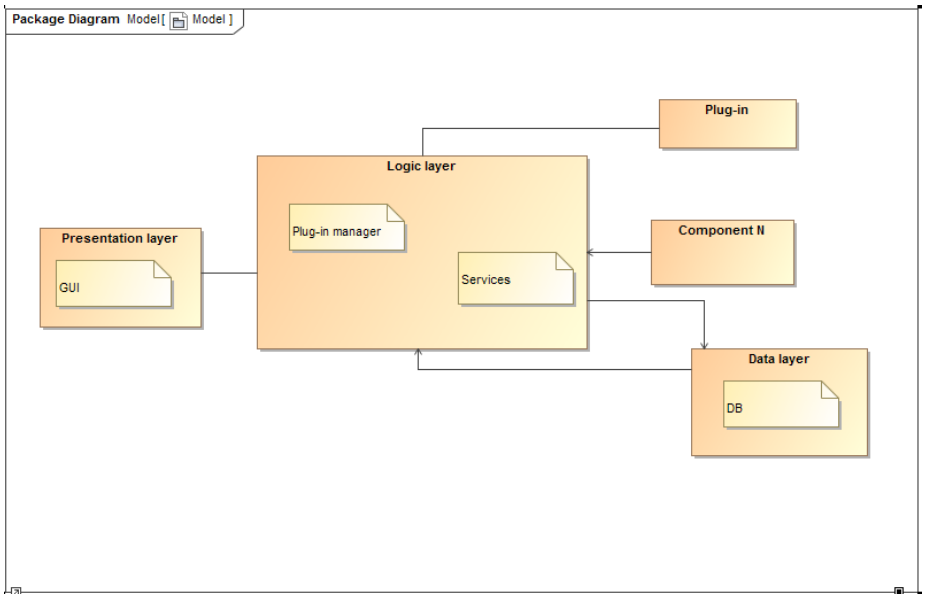

Figure 7: Amazon Dash Architecture consideration

9 Deployement Diagram

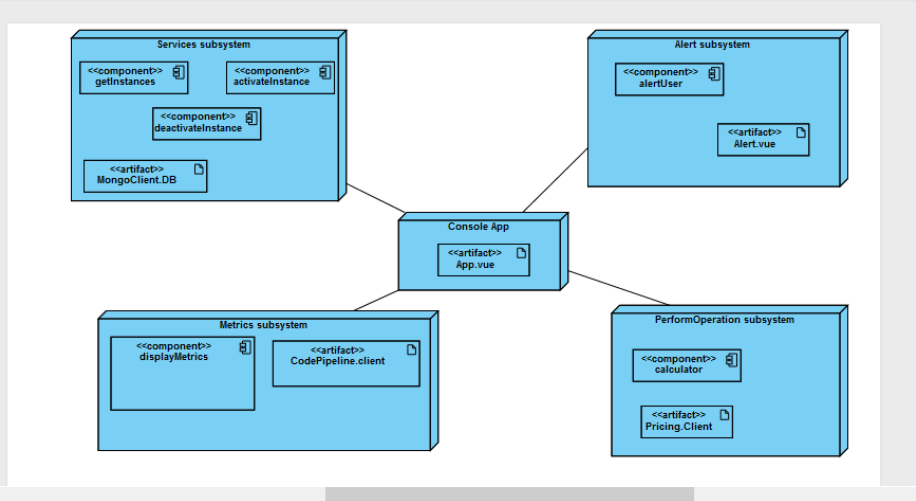

Figure 8: Amazon Dash Deployement diagram

10 Constraints

- The client must be registered with amazon web services in order to use the application.

- Should the total costing of all services exceed a certain amount then new services cannot be activated.

- The application should enforce authentication before a user can perform any action on the application.

11 Technology decisions

11.1 Vue

- We choose to use Vue to build our web console application because it is flexible enough because it accommodates for different ways of writing your code, for example, you can write a template in HTML, or in Javascript, or you can use JSX and also provides freedom to decide the structure of the application in a way that suits us.

- It is easily extendable by giving us the ability to integrate with other companion libraries which are officially supported and kept up to date with the Vue core library e.g Vuex

- It is also sophisticated in building Single-Page Applications and simple to use since it is a component library based on HTML, CSS and JSS.

- It provides us with an essential feature which is routing in order for us to communicate between components.

11.2 Vuetify

- Since Vue.js mainly focuses on the application logic and not on its appearance so we will combine it with the Vuetify.js frond-end UI framework to enrich the user interface.
- We use vuetify because it follows the Material Design standard and the component framework provides clean, semantic and reusable components. It also supports all the modern browsers and it also offers templates which we can use.

11.3 Python(backend)

- Python makes it easy to employ modern programming techniques and has many powerful libraries that speed up development time.

- Rather than manually keeping track of which libraries we are using, it gives us the option of utilizing the package management suite pip which helps to install and uninstall packages.

References

[1] Gouws, M. 2019, 'UP Project Templates-Amazon Dash-V1.2', Project proposal, Advance.