Adapting General Purpose Platform-as-a-Service for Customized Product Deployment

Damith A. Senanayake
Department of Computer Science and Engineering
University of Moratuwa, Sri Lanka
Email: damith.10@cse.mrt.ac.lk

***Abstract*— The web services industry has evolved to a point where a lot of services are provided for free or a free tier of services is offered, most of the times. An attempt of trying to exploit this situation is to try and implement a game server based on completely free and open-source technologies or at least not exceeding the free tiers of commercial technologies.**

***Index Terms*— Game Server, Platform as a Service (PaaS), Open-Source-Software**

I. Introduction

Platform-as-a-Service (PaaS) has become a common place concept in modern computing due to the advent of the web. Pioneered by commercial service providers such as Amazon, PaaS has been quickly gaining popularity as an alternative to restricted web hosting technologies provided by hosting servers. PaaS along with more general counterpart IaaS (Infrastructure as a Service), account for more flexible and versatile variations of web services.

The project discussed herewith has made use of PaaS to implement a game server. The game is a simple turn-based implementation of the infamous game "tanks". However this version is meant to be played by automated clients rather than human players. The server was originally designed as a part of a course module [1] where students were expected to develop clients that could play the game as an introductory module for artificial intelligence and networking. The original version of the server was written in C#.NET [1] and set up on a local area network to which multiple players were to connect and play the game. However, the limitations of this version were evident due to the predominantly local specifications of the game. That is, players could not connect from behind network address translations, routers or proxy servers to this game server. This project was done as an attempt to solve these problems and eliminate those limitations by porting the server to be hosted in a commercial PaaS.

II. Related Work

The combination of technologies used in this project was a unique one. Due to this, little or no formal study of the particular technologies that is of relevance, exist in the form of scholarly articles. The study needed for this project was therefore done in the form of collecting unsorted knowledge and compiling useful information out of it. The major aspects in which an extensive study was needed will be briefly cited in the following sections.

A. Platform as a Service

Platform as a Service is a concept that has gained popularity over the recent few years. In PaaS one can develop, host and use web applications with relative ease. However, most of the design decisions of the architecture of these technologies are decided by the providers of these platforms. For instance the platform operating system, the application programmer interfaces (APIs) etc. are fixed to the choice of the service provider [2]. Also they are becoming quite affordable and formidably more powerful due to the sharing of expensive resources [3].

B. Bidirectional Communications over the Web

The conventional methods of communications in web applications to emulate bidirectional data transfer was a usage of high-frequency polling. This is a very inefficient method to employ. Therefore with the advent of HTML5 the Websocket standards have provided a realization of actual two-way data transmission [4]. There are many commercial push-notifications services that make use of the Websocket standard.

III. Technology Overview

The following sections will discuss the technologies used in the development of this project.

A. Openshift by Redhat

The PaaS of choice for this project was Openshift, a commercial PaaS provided by Redhat[5]. Openshift comes with different *cartridges* – a collection of coherent technologies widely used in web applications. Note that these are especially aimed at web applications as opposed to web *sites*. An instance would be a cartridge consisting of a Python "django" implementation – a widely used python based web application framework. And there are other such popular cartridges such as phpMyAdmin, MySQL, PHP, JBOSS and several others. For the purpose of this project however, the do-it-yourself (DIY) cartridge was used [6]. The DIY comes only with a

Linux installation to which you can login remotely using SSH and use this to your advantage.

Before choosing Openshift, the original consideration was to use the Google app-angine (GAE) for development. However, due to the restrictions in thread support in the GAE this was abandoned [7]. Also some other alternatives such as Heroku, App-Fog, CloudControl were considered (and tried) but none of them provided the ease of implementation, the variety of available technologies and seamless integration between development and deployment environments that Openshift provided. As this was done solely for academic purposes, an attempt was made to make full use of the free-tier of these technologies. Due to this fact, Amazon elastic cloud (EC2) was not considered. However, it is note-worthy that App-fog and CloudControl use the Aamazon EC2 infrastructure underneath their platform [8]. However, the free-tier of Openshift was reasonably sufficient for the scope of this project.

B. *Java*

As mentioned above, the original server was implemented in C#.NET. The need of porting this into another language arose with the initial plan of using the Google App-Engine, as it only provided the use of a limited number of languages. Initially, Python was chosen as the go-to language. The reason for this was based on the following facts.

- Python has been the most mature language used in Google App-Engine development, meaning more community support
- Conceptual overhead of coding in python is quite low compared to other scripting languages and provides a good object orientation framework
- Accounts for light-weight programs, although efficiency may be at somewhat of a loss

However, as the development progressed, it was made evident that without a mature object model for specific aspects of this project, the implementation would prove to be more than a trivial task. For instance, the lack of a proper efficient and optimized model for handling events and object-dependency made it difficult to port the game server which was by large event-driven.

Because of this, the use of Python had to be abandoned and Java was chosen as the language for implementation. Java has an extensive library base which can eliminate the work by a large amount if one knows where to look. And due to the similarities in the languages C# and Java, the porting process was rendered a trivial task. However, the communication component of the project had to be completely reinvented and redesigned due to the requirement of communicating over the internet and thus an additional set of technologies and thereby dependencies were added to the program implementation. Because of this Maven was used as a build tool to help manage dependencies and to make the build process more convenient.

C. *Pubnub*

Part of the major challenge of implementing the new server was due to the difficulties in communications. The structure of the Openshift infrastructure used is largely responsible for this.

Openshift usually allows its users to bind their applications to the port 8080 for HTTP traffic. And this is bound through their load-balancing servers to the port 80 of the public URL of their applications. Refer to Fig. 1 for a clarification of the concept. Due to this continuation of a TCP socket connection for bidirectional data transmission was an impossibility. A method was needed to push notifications to the clients from the server. This plainly means that the server has to trigger an event in the client over the Internet through possible proxies or sub-networks.

Keeping in mind that this is already made possible with the use of web-sockets standard provided by the HTML5 framework, an easier implementation was needed and preferred. Therefore several commercial *push-notification services* were considered. Among them, most important were *Pusher* and *Pubnub*. But when it came to the ease of implementation and the community support, Pubnub was by far the rational choice.

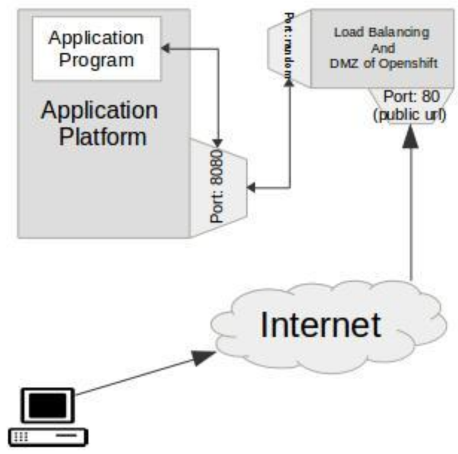

Fig. 1. The architecture of an Openshift Application with respect to network

For the Java implementation of the server, the Pubnub Java libraries were added as a dependency and used to push the notifications to the clients.

Although it is of trivial importance, a mention should be made as to the development environment itself.

D. Development Environment

1) Eclipse IDE-Openshift Integration

The Eclipse IDE supports Openshift development to a great extent. One only needs to make the necessary changes in the code in their own computers using the eclipse IDE, and when they commit the changes to the remote repositories, the build process and deployment process can be automatically completed. The use of IDEs for making the web application deployment an easy task has been extended in the notion of special IDEs such as the JBOSS Developer Studio(JBDevStudio) provided by Redhat where a direct integration exists between the JBOSS cartridges of Openshift and the JBDevStudio.

2) Version Control

Openshift uses Git repositories by default for version control. Git can be easily integrated into an IDE such as Eclipse which was used in the development of this project. With the combination of these two development environments the deployment was made astronomically easy.

IV. DESIGN AND IMPLEMENTATION

The architecture of the system is more or less the same as the original design of C# implementation. The major alterations have been done to the event handling model to make it simplistic and the communications model to allow for extended communication capabilities newly added to it.

The primary components of the design are as follows.

A. Game Engine

The Game Engine is the component responsible for keeping track of the players' status, positions, health and as such. It is also responsible of generating events, decoding and validating incoming messages from players, updating status according to messages, and also generating messages to be sent to the players. The messages will then be sent at regular intervals (1 second) to all clients.

B. Communicator

The communicator component is responsible of receiving, and sending messages from and to clients. The implementation of it is vastly different from the original design, whereas the functionality remains roughly the same. It should be able to broadcast data, and send individual messages to individual clients as well. Also it must be able to receive clients' join requests and handle them, as well as the registered clients' updates.

This component encapsulates the functionality of Pubnub as described above.

The communication sequence is as follows.

1. A client sends a join request on the public 'Join-request-channel'.
2. If a position is available, the communicator will inform the player of the secure channel which they must use to send updates to.
3. If no position is available, the client is sent a denial of request.
4. The client, upon receiving the secure channel, uses this to communicate to the server.
5. The client also has to listen on the global-update-channel, to which the server writes all the global regular updates of the game.

Obviously there are security concerns. However, with only a little amount of added effort, one can find a secure public-key scheme to accommodate for all the conceivable communication mishaps.

Fig. 2 shows the overview of the layout of channels between a client and the server. However, the channels, except for the global update channel and join request channel are duplicated for each new player.

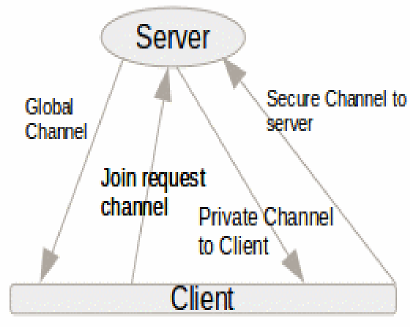

Fig. 2. The communications channel layout between a single client and a server

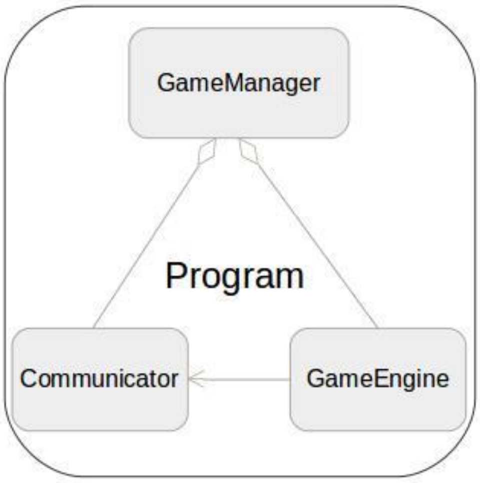

Fig. 3. Components of the main Game Server Program

C. *Game Manager*

The Game Manager component encompasses the two components mentioned above. It is responsible for creating instances of the game and maintaining and terminating them. The following diagram, Fig. 3 shows this relationship.

V. RESULTS

The deployment of the project was done through an automated maven build and the components are currently operational. The minimalistic code accounts for the improved performance, making the program run fast even on the small provisioning of resources offered by the free tier of Openshift.

Security functionality provided by Pubnub is flawed however, and calls for a developer implementation of a proper public-key schema for secure message exchange.

VI. CONCLUSIONS

The current trends in shared computational resources such as Platform as a Service and Infrastructure as a Service provide developers with versatility of developing and deploying versatile, multipurpose and flexible web applications. The architectures of various development strategies can be easily accommodated with modern use of integrated development tools along with the advent of the new technologies and trends in communications, and network technologies.

Acknowledgment

The author thanks Nisansa De Silva, Lecturer of the Department of Computer Science and Engineering of University of Moratuwa for the guidance and support given throughout the development of the project. Also the open-source community has shown great help and promise in completing this project.

References

[1] Nisansa de Silva, "*Classes:Programming Challenge II*" Available: http://www.cse.mrt.ac.lk/~nisansadds/

[2] Lawton, George. "Developing software online with platform-as-a-service technology." *Computer* 41.6 (2008): 13-15.

[3] Keller, Eric, and Jennifer Rexford. "The "Platform as a service" model for networking." *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. Vol. 4. No. 5. USENIX Association, 2010.

[4] P Lubbers, F Greco . "Html5 web sockets: A quantum leap in scalability for the web" *SOA World Magazine*, 2010

[5] OpenShift "*Platform as a Service"* Available:https://www.openshift.com/products

[6] Openshift "*Extending Openshift*" Available:https://www.openshift.com/developers/do-it-yourself

[7] Google App-Engine, "*Dealing with DeadlineExceededErrors*" Available:https://developers.google.com/appengine/articles/deadlineexceedederrors

[8] cloudControl "*Platform Documentation*" Available:https://www.cloudcontrol.com/dev-center/Platform%20Documentation

[9] Rothschild, Jeffrey Jackiel, et al. "Online gaming architecture." U.S. Patent No. 6,152,824. 28 Nov. 2000