Implementing a Secure Microservice Architecture: NGINX & Kubernetes Deployment

Orchestration: Alexander Moomaw
Eastern Washington University

Services: Andre Ramirez
Eastern Washington University

Logging: Brendan Hopkins
Eastern Washington University

Services: Beighlor Martinez
Eastern Washington University

Defense: Cameron Olivier
Eastern Washington University

Frontend: Dillon Pikulik
Eastern Washington University

Project Management: Chelsea Edwards
Eastern Washington University

Abstract

This paper presents a secure and scalable deployment architecture for web server environments using Kubernetes. The deployment incorporates a proxy and web application firewall for enhanced security and modular design. Additionally, we introduce our solution to logging and IP banning mechanisms to monitor and restrict unauthorized access. Our approach ensures high availability, load balancing, and robust defense against threats, offering a comprehensive framework for modern web applications.

1 Introduction

Ensuring the security and scalability of web server environments is a challenge in modern digital infrastructure. As cyber threats become more advanced, it's crucial to implement solutions that protect against unauthorized access while also maintaining high availability and performance.

To address these challenges, we propose a comprehensive deployment architecture by first utilizing Kubernetes, an open-source platform designed for automating deployment, scaling, and management of containerized applications. With this platform, we ensure high-availability to prospective users and a modular environment to build and scale upon as we please.

To enhance our microservice architecture, we introduced an additional abstraction layer using an NGINX proxy hosted locally on the server. This proxy server fetches content from our Kubernetes load balancer, serving as a gateway between our internal services and external communications. To mitigate potential threats, we integrated ModSecurity, a well-known web application firewall (WAF), into our proxy. ModSecurity monitors and blocks common exploitation methods by referencing a comprehensive rule list that addresses the OWASP Top Ten vulnerabilities.

We realize that exploiting our endpoint services isn't the only attack vector within our environment. To address this, we implemented host-level logging to monitor network traffic and identify IP addresses not communicating via SSH or HTTP. Additionally, we applied an automated process to block malicious IP addresses, enhancing our overall security posture.

2 Orchestration

The ultimate challenge of any deployment life cycle is tying all ideas into a single, nicely wrapped package. This is the goal of orchestration. To implement, or *orchestrate* a deployment such as this, we designed a multi-tiered architectural diagram (Figure 1), that outlines how every building block in our environment fits together to form a finalized representation of a working project.

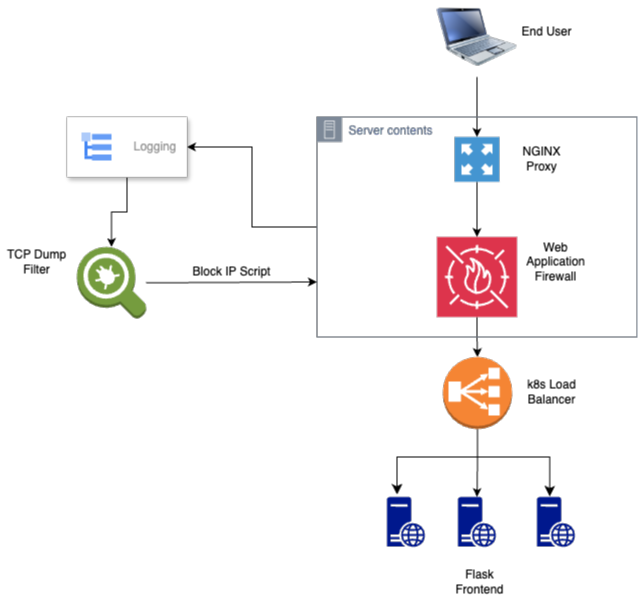

Figure 1: Multi-tiered architectural diagram.

To host our content, we utilized Kubernetes. This method first involved integrating a docker image that was specifically designed to host content created by our frontend developers. We then created service and deployment YAML files to control multiple containers behind a load balancer.

Once we had a functional cluster of orchestrated containers, the next goal was to make our load balancer not the first point of contact between external users and our internal environment. This involved integrating an NGINX reverse proxy on our host machine to facilitate communication without exposing our services directly to the internet.

To setup an NGINX reverse proxy, we had to install the service then modify the global configuration file in the `/etc/nginx/conf.d/` directory to specifically mediate communications between external users and our internal load balancer [1]. The specifics for this step is very convoluted, essentially we had to define server and location blocks with a `proxy pass` flag to enable the proxy feature of NGINX.

To tie our scalable, secure environment together on the orchestration end, we installed a web application firewall (WAF) on our reverse proxy to inspect requests being sent to the load balancer [2]. The requests will be meticulously cross-referenced against a series of rule lists specifically designed to catch exploits matching the OWASP Top Ten vulnerabilities. If a exploit request is detected, the WAF will drop the request and return a HTTP 403 status code to the threat actor.

3 Floating Figures and Lists

A paragraph of text goes here. Lots of text. Plenty of interesting text. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. More fascinating text. Features galore, plethora of promises.

4 Floating Figures and Lists

A paragraph of text goes here. Lots of text. Plenty of interesting text. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. More fascinating text. Features galore, plethora of promises.

5 Floating Figures and Lists

References

[1] Hostinger. How to set up nginx reverse proxy. https://www.hostinger.com/tutorials/how-to-set-up-nginx-reverse-proxy/, 2024. Accessed: 2024-05-21.

[2] Linode. Securing nginx with modsecurity. https://www.linode.com/docs/guides/securing-nginx-with-modsecurity/, 2024. Accessed: 2024-05-02.