Formatting Submissions for a USENIX Conference: An (Incomplete) Example

Nicholas Burlakov
Eastern Washington University

Second Name
Eastern Washington University

Your N. Here
Eastern Washington University

Second Name
Eastern Washington University

Your N. Here
Eastern Washington University

Second Name
Eastern Washington University

Second Name
Eastern Washington University

Abstract

The project entails deploying a robust web service securely and efficiently using Kubernetes (K8s) orchestration. This involves hosting Nginx servers to give users an IP-to-Physical Address service and a weather service. Incorporating a multi-layered defense strategy, including honeypot mechanisms, and utilizing Sysdig for comprehensive logging ensures heightened security and real-time monitoring. Leveraging Kubernetes' scalability and resilience optimizes resource utilization and bolsters fault tolerance, facilitating seamless access and utilization of the critical web service with enhanced efficiency and reliability.

1 Introduction

In today's digital landscape, deploying web services with security and efficiency is paramount. This project focuses on harnessing the power of Kubernetes (K8s) orchestration to achieve these goals. Kubernetes provides an ideal platform for hosting critical web services. In this endeavor, we will deploy an Nginx server to deliver an IP-to-Physical Address service and a weather service, ensuring these services are highly available and performant.

Security is a cornerstone of our approach. We will implement a multi-layered defense strategy, incorporating honeypot mechanisms to detect and deflect potential threats. For real-time monitoring and comprehensive logging, we will leverage Sysdig, a powerful tool that offers deep visibility into the system's behavior. This combination of advanced security measures and monitoring tools will safeguard our services against a wide range of cyber threats.

Furthermore, by utilizing Kubernetes, we aim to optimize resource utilization and enhance fault tolerance. Kubernetes' inherent capabilities in managing containerized applications will allow us to seamlessly scale our services according to demand, ensuring continuous availability and reliability. This project underscores the critical importance of efficient and secure web service deployment in meeting the growing demands of modern users and enterprises.

2 Orchestration

This is the orchestration section (Eric)

3 Front end

The front end of the project was crafted using a combination of HTML, CSS, and JavaScript, which outputted an interactive user interface. HTML was utilized to structure the content, creating the skeleton of the web pages. CSS was used in styling this content and JavaScript was integral for form validations and fetching content from our services.

To bring this front-end framework to life, the application was deployed on a Flask server, utilizing Python to manage the backend operations. Flask, a lightweight and flexible web framework, allowed for the efficient handling of requests and responses, serving the HTML, CSS, and JavaScript files to our orchestration team to implement. Python's integration with Flask facilitated the creation of routes and endpoints, enabling our backend services to interact with the front end. The use of Flask and Python in the backend complemented the front-end technologies, resulting in a cohesive and well-integrated web application.

4 Services

«««< HEAD This is the services section (Spencer) =======
The first service implemented in the project is a weather API designed to provide accurate and detailed weather forecasts by leveraging various web technologies and APIs. Built on a Flask server, this service efficiently handles user requests to deliver timely weather information. Users input a URL, which our system processes to retrieve the corresponding IP address. This IP address is then used to fetch the geographical location through a WHOIS lookup and the U.S. Census Bureau's geocoding service. By converting the address into latitude and longitude coordinates, we can accurately pinpoint the user's location.

Once the geographical coordinates are obtained, the service makes an API call to the National Weather Service (NWS) at weather.gov. The NWS API provides detailed weather forecasts based on the coordinates received. To optimize performance and reduce redundant API calls, the service implements caching mechanisms. This caching stores previously fetched weather data, ensuring that repeat requests for the same location are served quickly and efficiently. The integration of these components within the Flask framework not only streamlines the process but also ensures scalability and reliability, making our weather API service a robust tool for users seeking accurate weather information. »»»> c373d122dbb882a8121d00e642e07ea5aeb22987

5 Logging

This is the logging section (miah)

6 Defense

For the defense of our system we implemented multiple defensive techniques. The first defensive technique is built into our system design and is the load balancer which functions as moving target defense by putting users into one version or

References