Report: Building a FaaS Platform on Kubernetes

Done by:-
Akanksha Dadhich(23m0830)
Prerna Priyadarshini (23m2115)
~Team name :- **CubeFlash**

Introduction

The goal of this project was to develop a Function-as-a-Service (FaaS) platform as a wrapper around Kubernetes (K8s). The FaaS platform needed to support function registration, trigger registration, trigger dispatch, metrics, and more. The scope included building the FaaS platform, demonstrating its correctness and feature set, and conducting performance benchmarking.

Function-as-a-Service (FaaS) is a cloud computing model where developers can deploy individual functions or pieces of code and run them in response to events or triggers without managing the underlying infrastructure. Here are some key aspects and benefits of FaaS:

1. Event-Driven Architecture:
FaaS platforms are designed around the concept of event-driven architecture, where functions are executed in response to events or triggers such as HTTP requests, database changes, file uploads, or scheduled events.
This allows developers to build applications as a series of stateless functions that respond to specific events, enabling a more modular and scalable architecture.

2. Pay-Per-Use Billing Model:
FaaS platforms typically operate on a pay-per-use billing model, where users are charged only for the compute resources consumed by their functions during execution.
This provides cost efficiency by eliminating the need to provision and pay for idle resources, as resources are dynamically allocated and released based on demand.

3. Scalability and Elasticity:
FaaS platforms automatically scale functions based on incoming request volume, allowing applications to handle varying workloads and spikes in traffic without manual intervention.
Functions are instantiated and executed in parallel, providing elasticity to scale up or down in real-time based on demand.

4. Simplified Deployment and Management:

Developers can deploy functions to FaaS platforms without worrying about managing the underlying infrastructure, operating system, or runtime environment.
FaaS platforms handle tasks such as provisioning, scaling, monitoring, logging, and security, allowing developers to focus on writing and deploying code.

5. Language and Runtime Flexibility:
FaaS platforms support multiple programming languages and runtimes, allowing developers to write functions in their preferred language, such as JavaScript, Python, Java, Go, or Ruby.
This flexibility enables teams to leverage existing skills and choose the best language for each use case, improving developer productivity and code maintainability.

6. Microservices and Serverless Architecture:
FaaS is often used in conjunction with microservices architecture and serverless computing paradigms, where applications are built as a collection of loosely coupled and independently deployable functions.
This enables developers to break down monolithic applications into smaller, more manageable components, leading to better scalability, agility, and maintainability.

7. Rapid Development and Deployment:
FaaS platforms facilitate rapid development and deployment cycles by providing fast iteration times and seamless deployment workflows.
Developers can quickly prototype, test, and deploy functions in production environments, accelerating time-to-market and enabling continuous delivery practices.

8. Event-Driven Integration and Orchestration:
FaaS platforms integrate seamlessly with various event sources and services, enabling event-driven integration and orchestration workflows.
Functions can respond to events from external services, trigger actions in other services, and coordinate complex workflows through event-driven messaging and orchestration.
Challenges

Challenges Faced During Deployment and Resolution

During the development phase, we encountered several challenges while attempting to create a new container within an existing pod. One significant obstacle arose due to permission-denied issues within Minikube, our local Kubernetes environment. Despite our efforts to modify Role-Based Access Control (RBAC) roles to grant necessary permissions, we encountered limitations that prevented successful container creation.

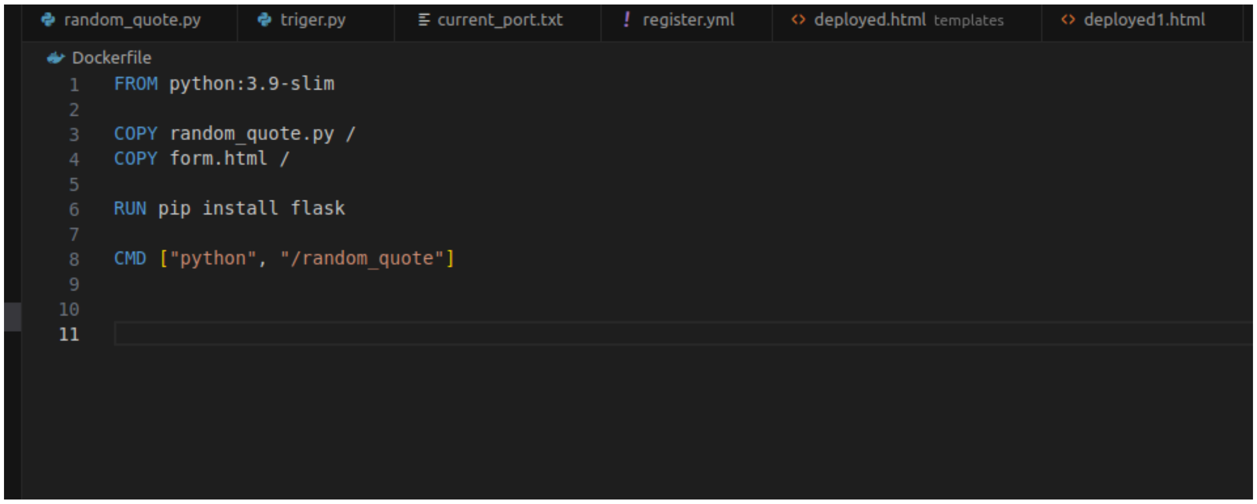
Additionally, while we were successful in configuring port forwarding and creating a container within a single pod, our attempts to add a new container to an existing pod proved unsuccessful. This posed a major challenge to our deployment strategy.

In response to these obstacles, we opted for a revised approach, prioritizing a user-centric design. We transitioned to providing each user with a dedicated pod for each functionality. This approach ensured better scalability, flexibility, and resource isolation, addressing the challenges encountered with container creation within existing pods. By focusing on one pod per functionality, we were able to overcome the limitations posed by Minikube permissions and RBAC configurations, enabling smoother deployment and improved user experience.

Correctness and Feature Set

1. Creating Docker Image and Uploading to Docker Hub

In this project, a Dockerfile was created to containerize a Flask application. The Dockerfile includes instructions to build the application's environment and dependencies. Once the Docker image was successfully built, it was uploaded to Docker Hub, a cloud-based repository for Docker images.



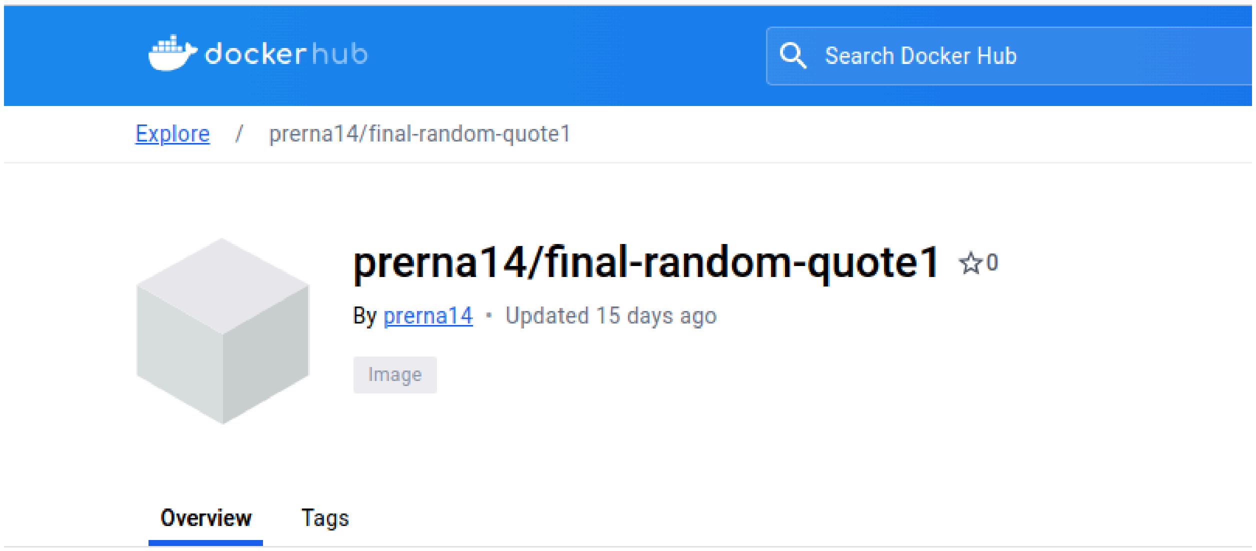
```
FROM python:3.9-slim

COPY random_quote.py /
COPY form.html /

RUN pip install flask

CMD ["python", "/random_quote"]


```

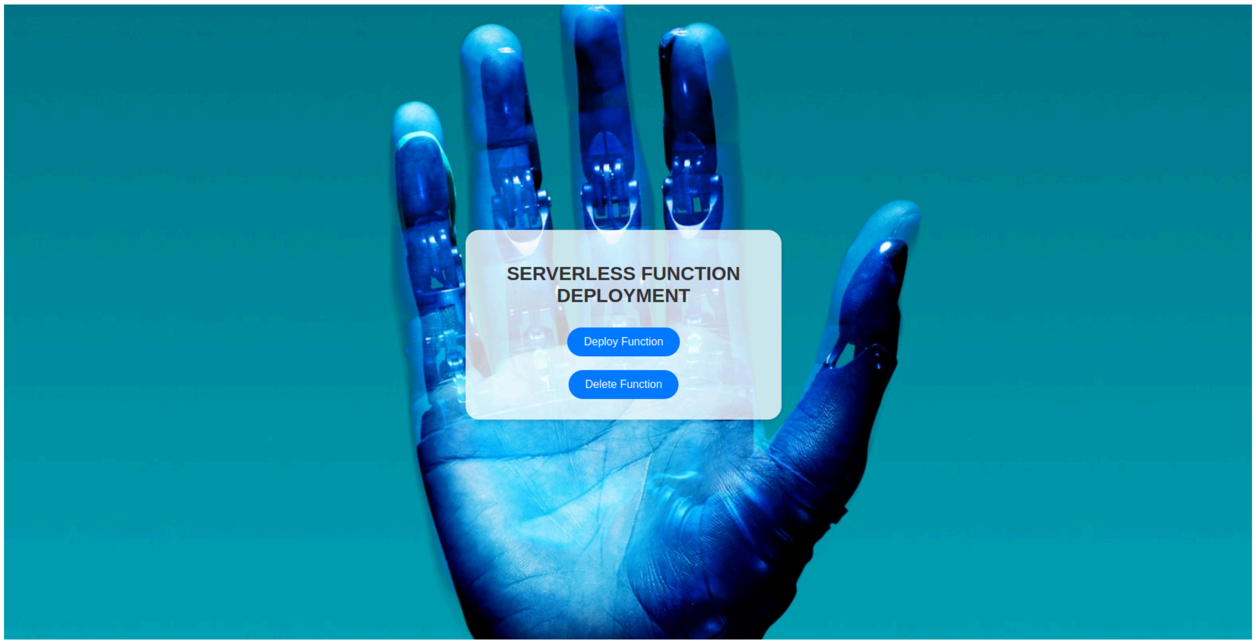


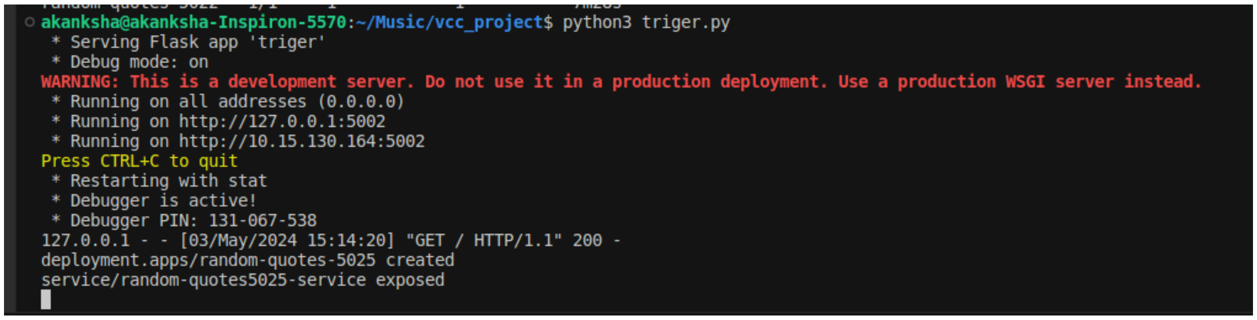
prerna14/final-random-quote1 ☆0

By prerna14 • Updated 15 days ago

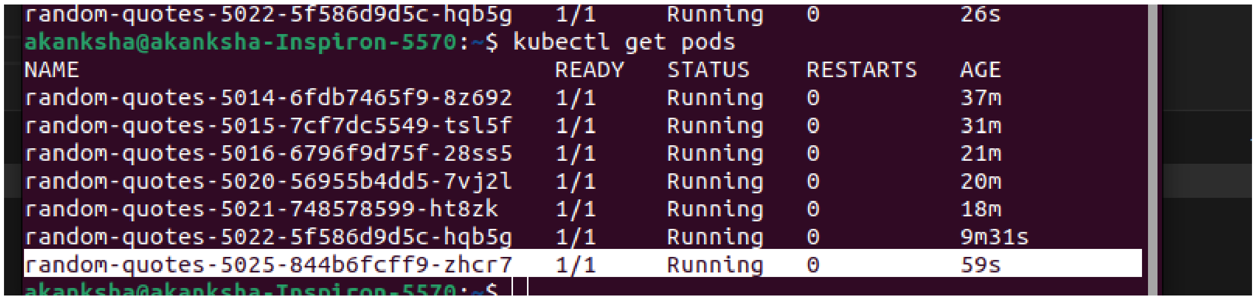
2. User Interface Interaction for Pod Creation

A user-friendly interface (UI) was developed to facilitate user interaction. By clicking on the designated button, the UI triggers the creation of a Kubernetes pod. This pod is responsible for encapsulating the application's containerized environment. Subsequently, the pod retrieves the Docker image from Docker Hub, ensuring that the latest version of the application is always deployed.

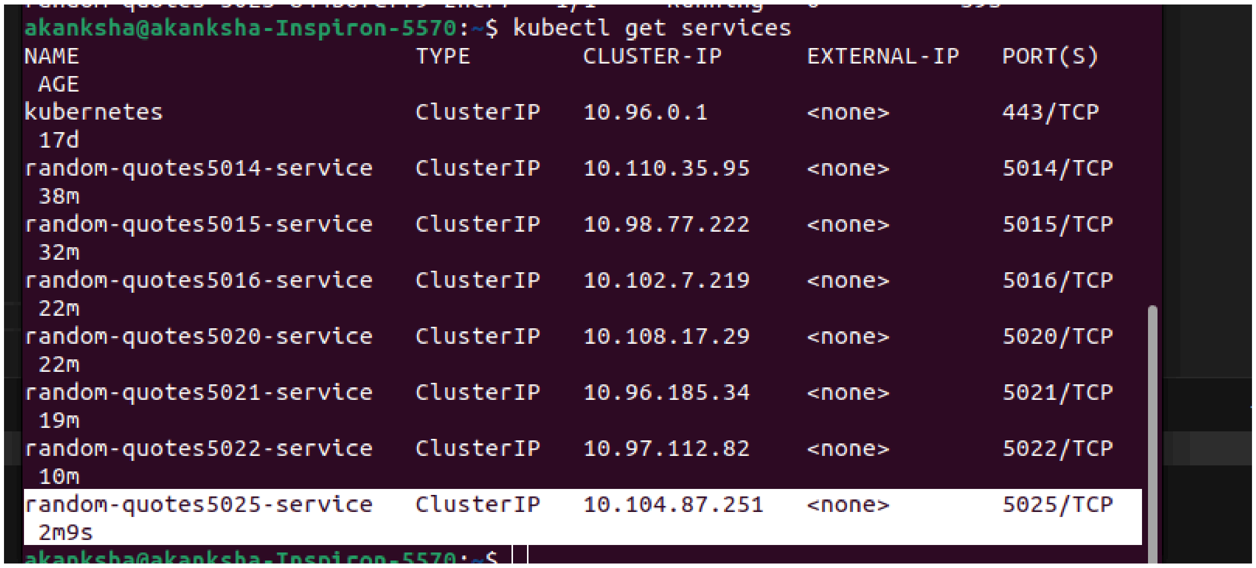




```
akanksha@akanksha-Inspiron-5570:~/Music/vcc_project$ python3 triger.py
 * Serving Flask app 'triger'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5002
 * Running on http://10.15.130.164:5002
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 131-067-538
127.0.0.1 - - [03/May/2024 15:14:20] "GET / HTTP/1.1" 200 -
deployment.apps/random-quotes-5025 created
service/random-quotes5025-service exposed
```

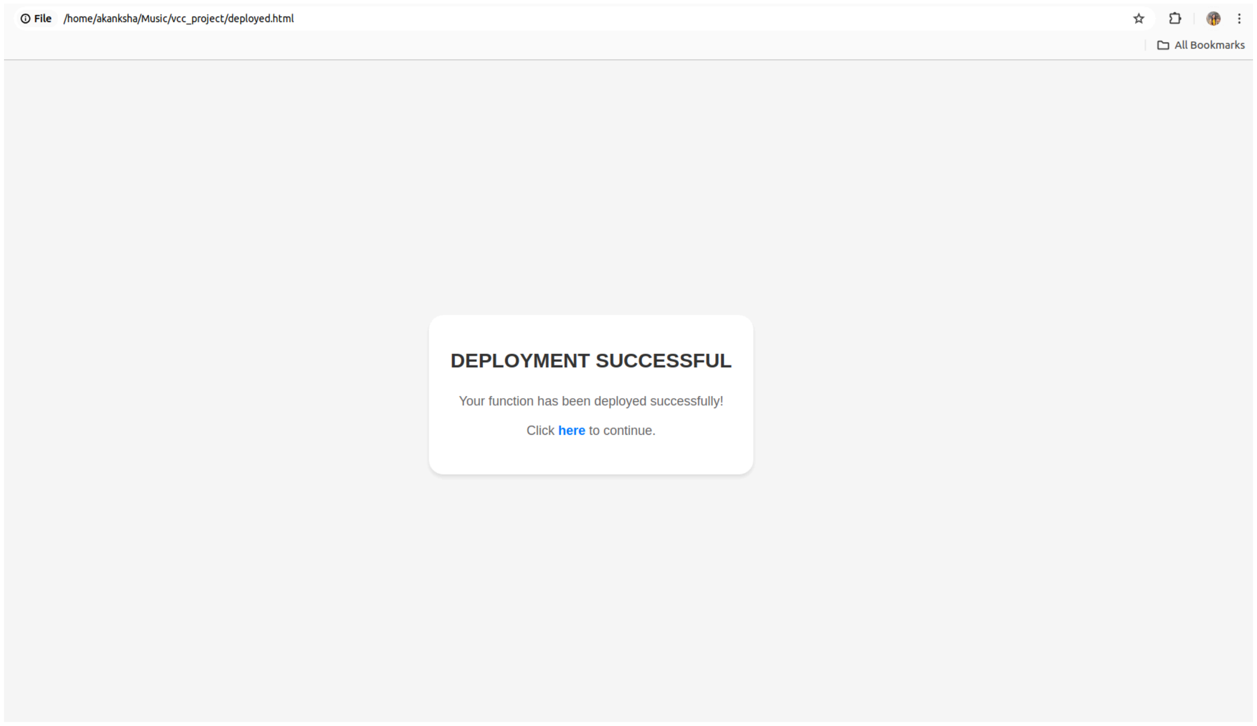


```
random-quotes-5022-5f586d9d5c-hqb5g   1/1     Running   0          26s
akanksha@akanksha-Inspiron-5570:~$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
random-quotes-5014-6fdb7465f9-8z692   1/1     Running   0          37m
random-quotes-5015-7cf7dc5549-tsl5f   1/1     Running   0          31m
random-quotes-5016-6796f9d75f-28ss5   1/1     Running   0          21m
random-quotes-5020-56955b4dd5-7vj2l   1/1     Running   0          20m
random-quotes-5021-748578599-ht8zk    1/1     Running   0          18m
random-quotes-5022-5f586d9d5c-hqb5g   1/1     Running   0          9m31s
random-quotes-5025-844b6fcff9-zhcr7   1/1     Running   0          59s
akanksha@akanksha-Inspiron-5570:~$
```



```
akanksha@akanksha-Inspiron-5570:~$ kubectl get services
NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
 AGE
kubernetes                     ClusterIP   10.96.0.1        <none>        443/TCP
 17d
random-quotes5014-service      ClusterIP   10.110.35.95     <none>        5014/TCP
 38m
random-quotes5015-service      ClusterIP   10.98.77.222     <none>        5015/TCP
 32m
random-quotes5016-service      ClusterIP   10.102.7.219     <none>        5016/TCP
 22m
random-quotes5020-service      ClusterIP   10.108.17.29     <none>        5020/TCP
 22m
random-quotes5021-service      ClusterIP   10.96.185.34     <none>        5021/TCP
 19m
random-quotes5022-service      ClusterIP   10.97.112.82     <none>        5022/TCP
 10m
random-quotes5025-service      ClusterIP   10.104.87.251    <none>        5025/TCP
 2m9s
akanksha@akanksha-Inspiron-5570:~$
```

3. Deployment of Service and Port Forwarding

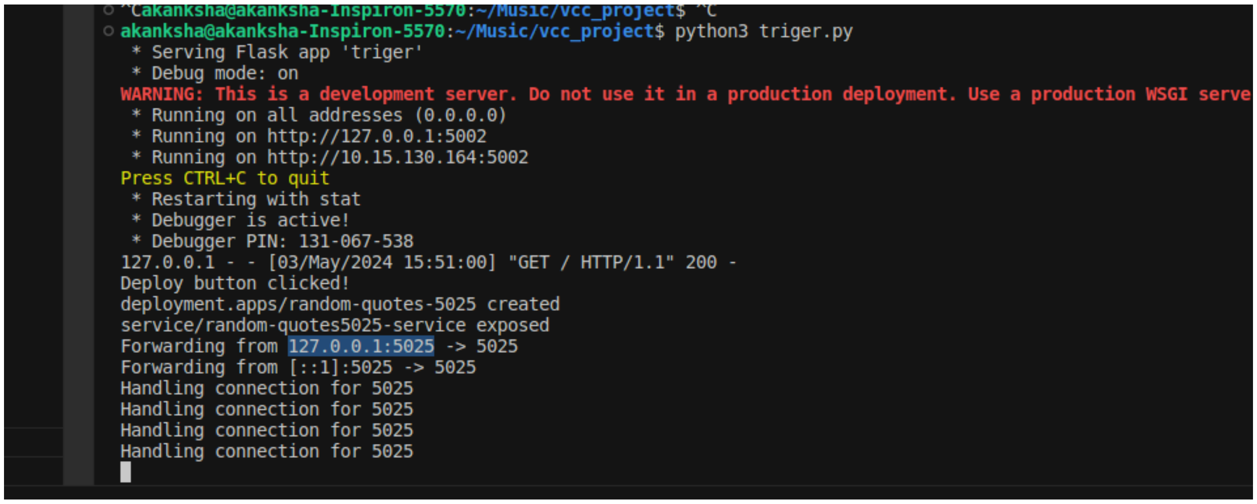


DEPLOYMENT SUCCESSFUL

Your function has been deployed successfully!

Click here to continue.

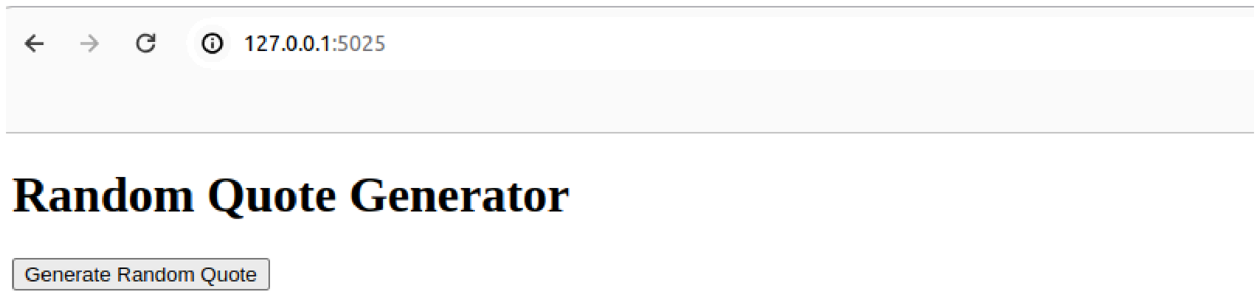
Upon pod creation, the Kubernetes service is deployed to expose the application to external traffic. This service acts as a load balancer, directing incoming requests to the appropriate pod. Additionally, port forwarding is configured to enable communication between the Kubernetes cluster and the application. This ensures seamless access to the site hosted within the containerized environment.



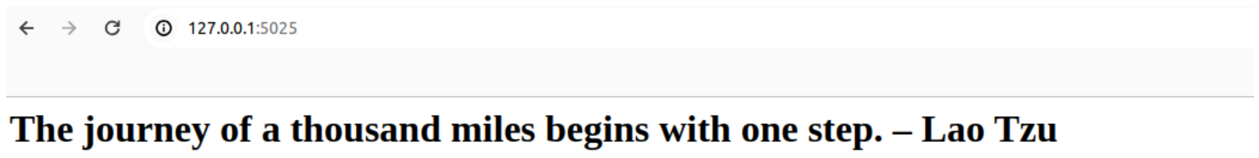
```
^Cakanksha@akanksha-Inspiron-5570:~/Music/vcc_project$ ^C
akanksha@akanksha-Inspiron-5570:~/Music/vcc_project$ python3 triger.py
 * Serving Flask app 'triger'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI serve
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5002
 * Running on http://10.15.130.164:5002
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 131-067-538
127.0.0.1 - - [03/May/2024 15:51:00] "GET / HTTP/1.1" 200 -
Deploy button clicked!
deployment.apps/random-quotes-5025 created
service/random-quotes5025-service exposed
Forwarding from 127.0.0.1:5025 -> 5025
Forwarding from [::1]:5025 -> 5025
Handling connection for 5025
Handling connection for 5025
Handling connection for 5025
Handling connection for 5025
```

4. Accessibility of the Website

Following the successful deployment of the service and port forwarding, the website becomes accessible to users. Users can access the site by navigating to the designated URL, which is made available through the port forwarding configuration. This enables users to interact with the application and access its functionalities effortlessly.



Random Quote Generator



The journey of a thousand miles begins with one step. – Lao Tzu

5. Pod Deletion by User

To enhance the user experience and offer greater flexibility, we are considering the implementation of a delete button feature. This feature would empower users to delete their allocated pods, allowing them to manage their resources more effectively and pay only for the services they use. Implementation of this feature aligns with our commitment to providing a user-friendly and cost-effective solution for our users.

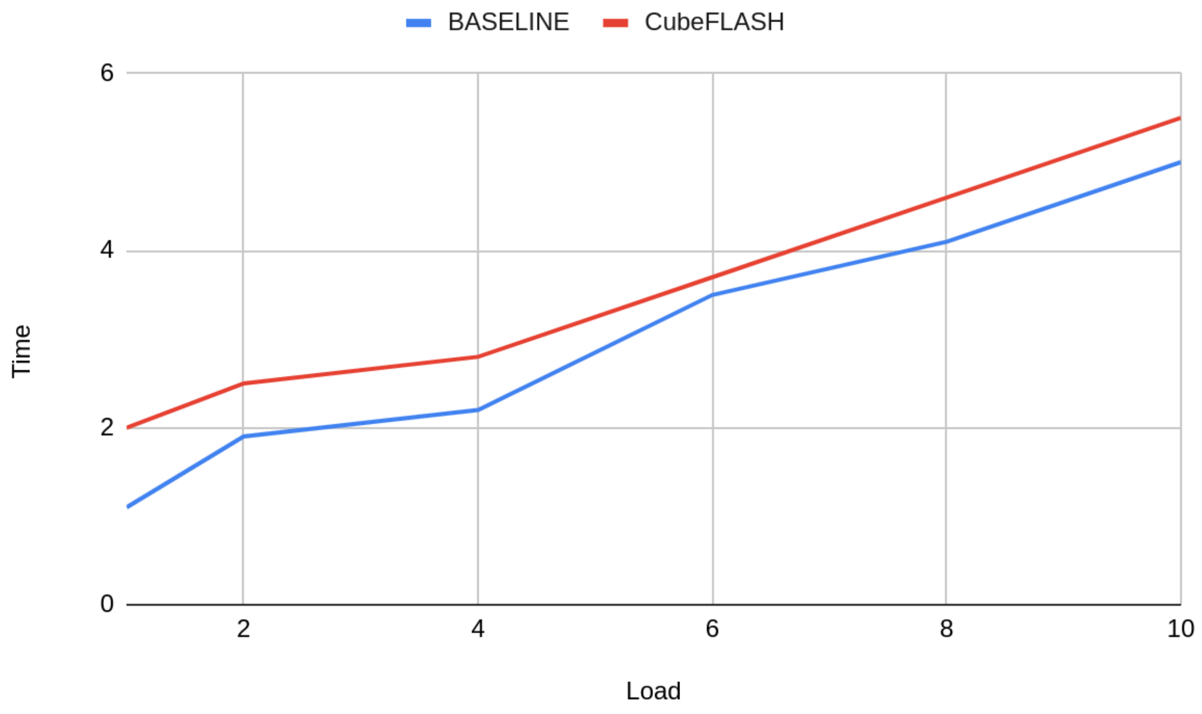
Performance Evaluation

To assess the performance of our proposed serverless infrastructure compared to traditional non-serverless infrastructure, we conducted a comprehensive evaluation under varying load conditions. The load was defined as the number of simultaneous requests sent to both systems. We meticulously recorded the response time for each request in both technologies to discern performance discrepancies.

Our findings reveal that our serverless infrastructure exhibited longer response times compared to the non-serverless baseline. This delay can be primarily attributed to the overhead involved in creating and configuring pods, along with other associated setup processes inherent in serverless deployments. However, our analysis extends beyond mere response times to offer valuable insights into the long-term performance advantages of our serverless approach.

Despite the initial overhead, our serverless infrastructure promises superior performance over time, particularly for individual users. This assertion is underpinned by the fact that our technology allocates a dedicated container for each user, ensuring optimal resource allocation and isolation. In contrast, the non-serverless baseline architecture necessitates bandwidth sharing among multiple users, leading to potential contention and performance degradation, especially during peak usage periods.

From a technical perspective, the observed discrepancy in response times underscores the intricate trade-offs between serverless and non-serverless architectures. While our serverless solution may incur a marginal delay during initial pod creation and setup, it offers unparalleled scalability and resource isolation benefits in the long run. This strategic approach aligns with modern cloud-native principles, emphasizing efficient resource utilization, dynamic scalability, and enhanced user experience.



Conclusion

In conclusion, the challenges we encountered during the development phase led us to reevaluate our deployment strategy and adopt a user-centric approach. Despite facing permission denied issues and limitations with container creation within existing pods, we successfully overcame these obstacles by transitioning to a one-pod-per-functionality model.

This revised approach not only addressed the technical challenges we faced but also enhanced scalability, simplified deployment processes, and improved the overall user experience. By providing users with dedicated pods for their specific functionalities, we empowered them with greater control over their resources and streamlined the management of their applications.

Furthermore, the implementation of a delete function for user-requested pods ensures cost efficiency and transparent resource billing. Users can now easily manage their resources, scaling them up or down as needed and only paying for the resources they actively use.

Overall, our experience highlights the importance of adaptability and user-centric design in overcoming challenges and delivering a robust and user-friendly solution in Kubernetes environment.

References :-

- https://minikube.sigs.k8s.io/docs/start/
- https://flask.palletsprojects.com/en/3.0.x/
- https://kubernetes.io/docs/concepts/containers/