Cemalhan Alptekin

Documentation for Honeycomb's Microservices Demo: Online Boutique

1. Introduction

The Online Boutique is a cloud-native microservices demo application. It showcases a typical 10-tier microservices architecture and is built using multiple languages, including Go, Java, .NET, Node, and Python. The application simulates a web-based e-commerce platform where users can browse items, add them to their cart, and make purchases. Honeycomb uses this application to demonstrate key technologies such as Kubernetes, gRPC, and OpenTelemetry to highlight cloud-native monitoring and observability best practices.
This application is designed to run on any Kubernetes cluster, making it easy to deploy with minimal configuration.

2. Tools and Technologies Used

a. Docker

- Why it's used: Docker is utilized to containerize the application's microservices, ensuring that each service runs in its own isolated environment. This guarantees consistent behavior across various deployment environments.
- Description: Docker simplifies the packaging of microservices along with their dependencies into containers, allowing for easy scaling, deployment, and management in the cloud.

b. Kubernetes

- Why it's used: Kubernetes is employed as an orchestration platform to manage the deployment, scaling, and maintenance of the microservices. It allows Online Boutique to operate seamlessly, handling automated load balancing and service discovery.
- Description: Kubernetes organizes the containers and ensures they run efficiently in a distributed system, providing automated recovery and scaling based on resource needs.

c. Helm

- Why it's used: Helm is used to streamline the installation of the application's Kubernetes resources, offering a predefined template (Helm charts) for ease of deployment.

- Description: Helm simplifies Kubernetes management by packaging related Kubernetes resources together, making the setup of complex applications straightforward and reusable.

d. gRPC

- Why it's used: gRPC is a high-performance, open-source RPC framework designed for efficient communication between the microservices in the Online Boutique application.
- Description: gRPC is used to connect microservices, enabling fast, low-latency, and scalable communication. Its support for multiple languages and built-in features like load balancing and service discovery make it ideal for a microservices architecture.

e. Honeycomb

- Why it's used: Honeycomb is the primary observability tool used for in-depth debugging and performance analysis. It provides high-cardinality event-level insights, crucial for understanding system behavior.
- Description: Honeycomb offers real-time insights into the functioning of microservices through event-based tracing and performance visualization. It helps developers detect and resolve complex issues that may arise in the microservices.

f. OpenTelemetry

- Why it's used: OpenTelemetry is used to instrument the Online Boutique application, providing the foundation for collecting traces, metrics, and logs across the microservices.
- Description: OpenTelemetry offers both simple and advanced instrumentation techniques to trace and monitor the entire system. Each microservice in the application has specific code examples in the src folder, explaining how OpenTelemetry is integrated to enhance observability.

g. Minikube

- Why it's used: Minikube is used to run a local Kubernetes cluster for development and testing purposes. It simplifies the setup process and allows developers to test their applications in an environment that closely resembles a production Kubernetes cluster without needing cloud resources.
- Description: Minikube provides a lightweight Kubernetes environment that runs locally on a developer's machine. It supports multiple container runtimes and is configured to mimic a full Kubernetes cluster, enabling easy development, testing, and debugging of applications. The use of Minikube allows for rapid iteration and testing of the Online Boutique application before deploying to larger, more complex environments.

3. Installation Overview

The installation process for the Online Boutique demo application involves the following steps:

1. **Setting up Kubernetes:** Deploy the application on any Kubernetes cluster.
2. **Using Helm:** Helm is used to install the Kubernetes resources required to run the microservices.
3. **Deploying Monitoring and Tracing:** Prometheus and Jaeger are set up to collect metrics and provide distributed tracing for the application.
4. **Enabling Observability:** OpenTelemetry is utilized to instrument the services, while Honeycomb integrates for in-depth observability.

4.Why These Tools?

- **Containerization (Docker):** Docker ensures each microservice is deployed with consistent dependencies, improving reliability and reducing environment-specific issues.
- **Orchestration (Kubernetes):** Kubernetes provides scalability, resilience, and efficient resource management for the microservices.
- **Communication (gRPC):** gRPC offers an efficient, language-agnostic protocol for communication between services, allowing high-performance data exchange.
- **Observability (OpenTelemetry, Honeycomb):** These tools together provide real-time insights into the application's performance, making it easier to monitor, trace, and resolve system issues in a distributed architecture.

5.Conclusion

The Online Boutique application showcases a modern microservices architecture using multiple languages and cloud-native technologies like Kubernetes, gRPC, and OpenTelemetry. Each tool used plays a crucial role in ensuring the application is reliable, scalable, and observable, making it an ideal demonstration for understanding microservices in action.

Development

Prerequisites:

Before getting started with the installation, ensure you have the following tools installed and configured on your development machine:

- Docker for Desktop: Provides containerization for the application.
- kubectl: Command-line tool to interact with the Kubernetes cluster.
- skaffold: Automates the building, pushing, and deployment of Docker images for Kubernetes.
- Helm: Kubernetes package manager to install and manage Kubernetes applications.

1. Kubernetes Quickstart

a. Launch Kubernetes Cluster

You can use Docker Desktop or Minikube to launch your local Kubernetes cluster.

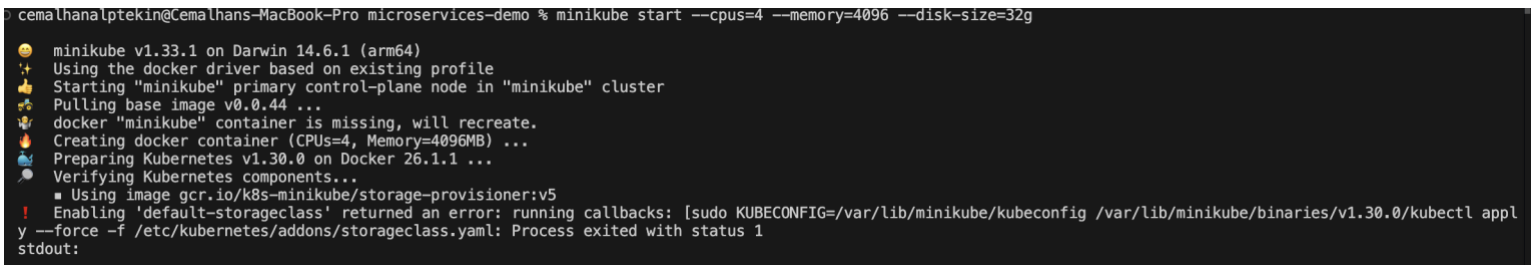
Using Docker Desktop:

1. Open Docker Desktop and navigate to Preferences.
2. Enable Kubernetes under the preferences.
3. Set the following resource requirements for the cluster:
    - CPUs: At least 3
    - Memory: At least 6.0 GiB
    - Disk Space: At least 32 GB (Set under the "Disk" tab)

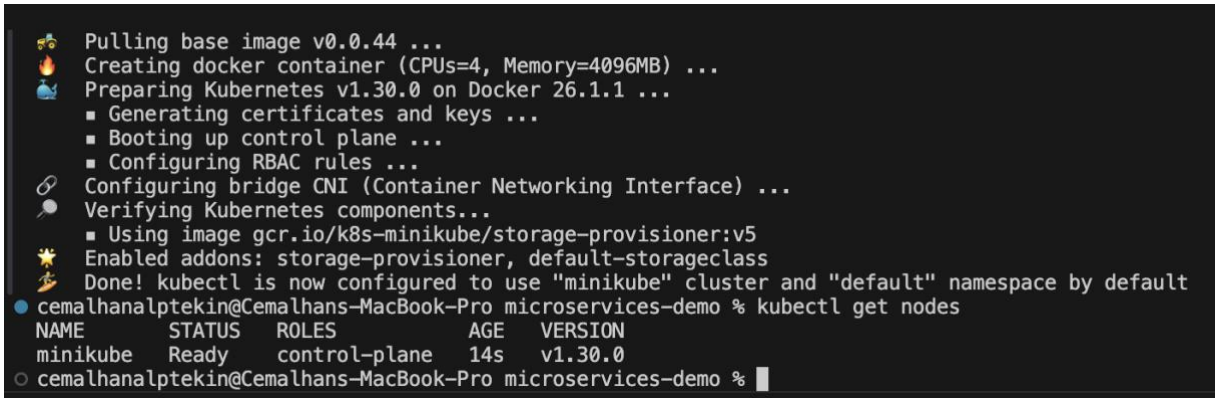
Using Minikube (Tested on Ubuntu Linux):

1. Make sure your Minikube setup has the following minimum resources:
    - CPUs: 4
    - Memory: 4.0 GiB
    - Disk Space: 32 GB
2. Start Minikube with the following command:

    minikube start --cpus=4 --memory=4096 --disk-size=32g



3. Run **kubectl get nodes** to verify the cluster is up and running, and you're connected to the Kubernetes control plane.



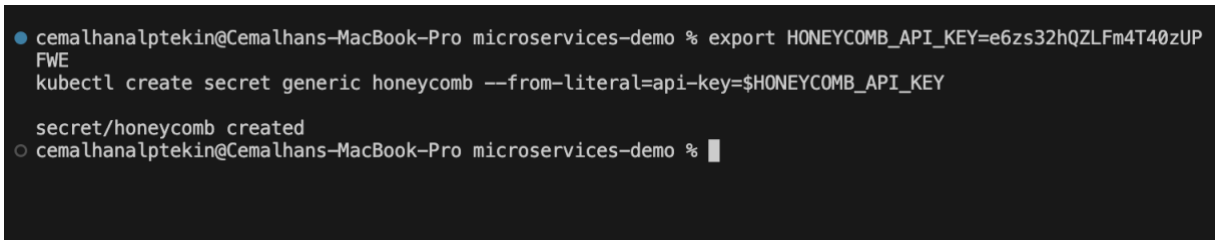
2. Configure Honeycomb API Key

The Honeycomb API key is required for observability and tracing. Add it to Kubernetes as a secret:

Replace $HONEYCOMB_API_KEY with your actual Honeycomb API key. For example, if your API key is abc123, run the following command:

export HONEYCOMB_API_KEY=abc123

kubectl create secret generic honeycomb --from-literal=api-key=$HONEYCOMB_API_KEY

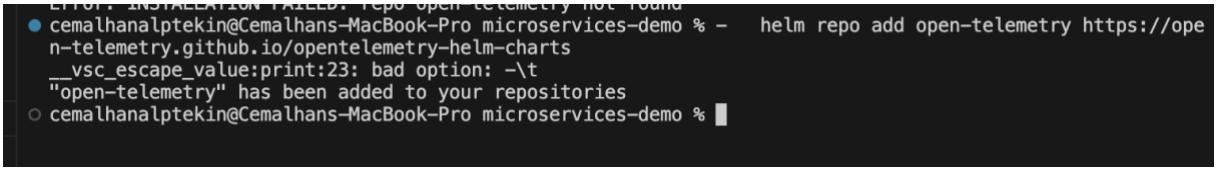


3. Install OpenTelemetry Collector

To collect observability data, you need to install the OpenTelemetry Collector using Helm:

1. Add the OpenTelemetry Helm chart repository:

- **helm repo add open-telemetry** https://open-telemetry.github.io/opentelemetry-helm-charts

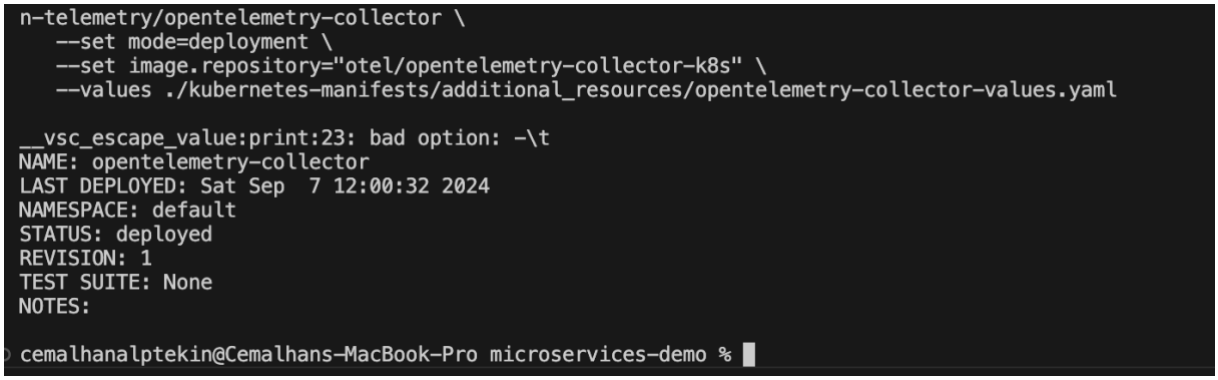


- **helm install opentelemetry-collector open-telemetry/opentelemetry-collector **

**--set mode=deployment **

**--set image.repository="otel/opentelemetry-collector-k8s" **

--values ./kubernetes-manifests/additional_resources/opentelemetry-collector-values.yaml



4. Deploy the Application with Skaffold

To build and deploy the Online Boutique microservices, run the following:

1. Use Skaffold to build and deploy the Docker images to Kubernetes:

   skaffold run

***Note*: The first deployment may take around 20 minutes, depending on the size of the Docker images and network speed.**

2. If you want to automatically rebuild and redeploy the images as you make changes to the code, use the following command:

   skaffold dev

3. Verify the application is running by checking the status of your Kubernetes pods:

   kubectl get pods

```
cemalhanalptekin@Cemalhans-MacBook-Pro microservices-demo % kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
adservice-8549d478cd-hb4h8               1/1     Running   0          95s
cartservice-7fc84fc8b8-x722x             1/1     Running   0          95s
checkoutservice-5d7ddccdc-5f42m          1/1     Running   0          95s
currencyservice-6d99577875-qgpm6         1/1     Running   0          95s
emailservice-86ff7885-ckq9d              1/1     Running   0          95s
frontend-68f989845c-8mm9r                1/1     Running   0          95s
loadgenerator-67cff85574-r7csn           1/1     Running   0          95s
opentelemetry-collector-7f84fff78-2n968  1/1     Running   0          18m
paymentservice-7954d7bb45-pwgfh          1/1     Running   0          95s
productcatalogservice-7b6574d6fc-4cj68   1/1     Running   0          94s
recommendationservice-68b9896746-s8m4q   1/1     Running   0          94s
redis-cart-5948d57c8-lnpgq               1/1     Running   0          94s
shippingservice-7c44cfd49b-f2rtr         1/1     Running   0          93s
```

5. Access the Application

Once the pods are running, access the web frontend of the application:

Docker for Desktop: The frontend should be automatically available at:

http://localhost:80

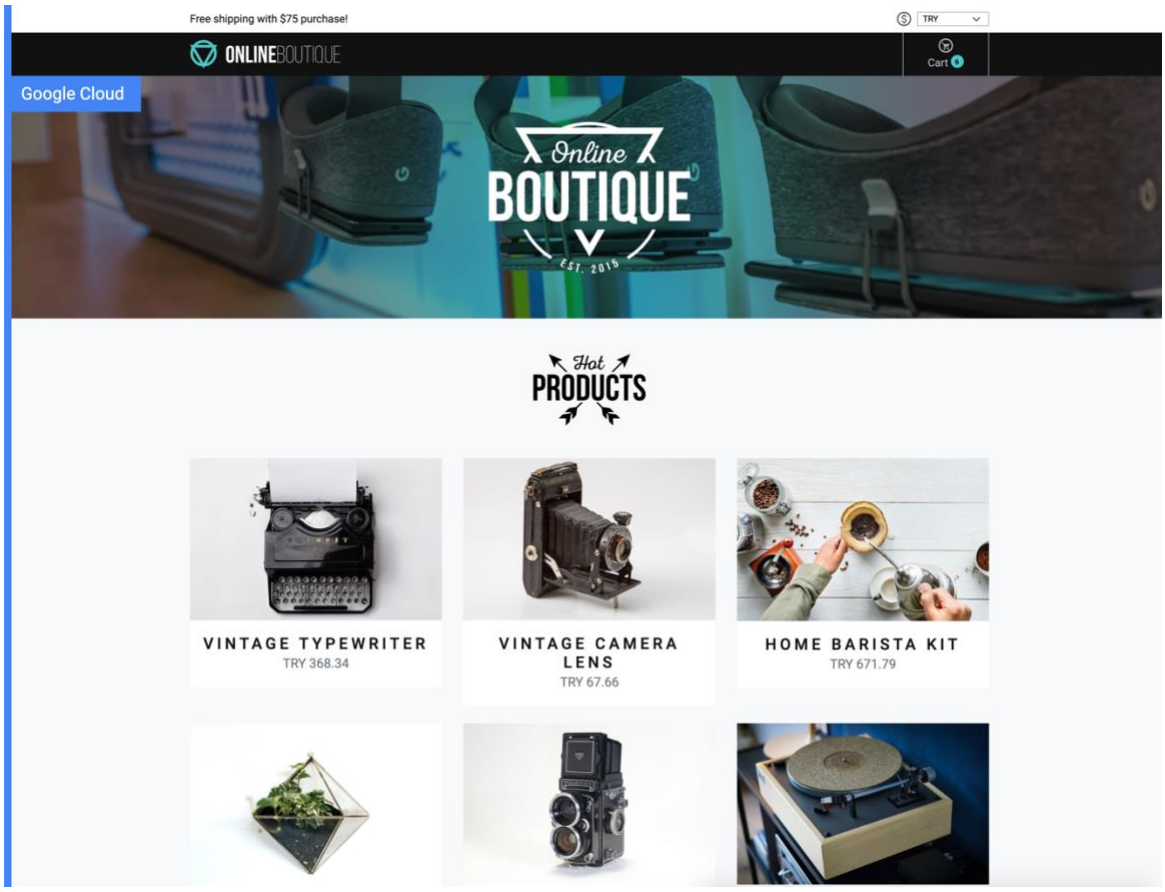
Minikube: Run the following command to access the frontend service:

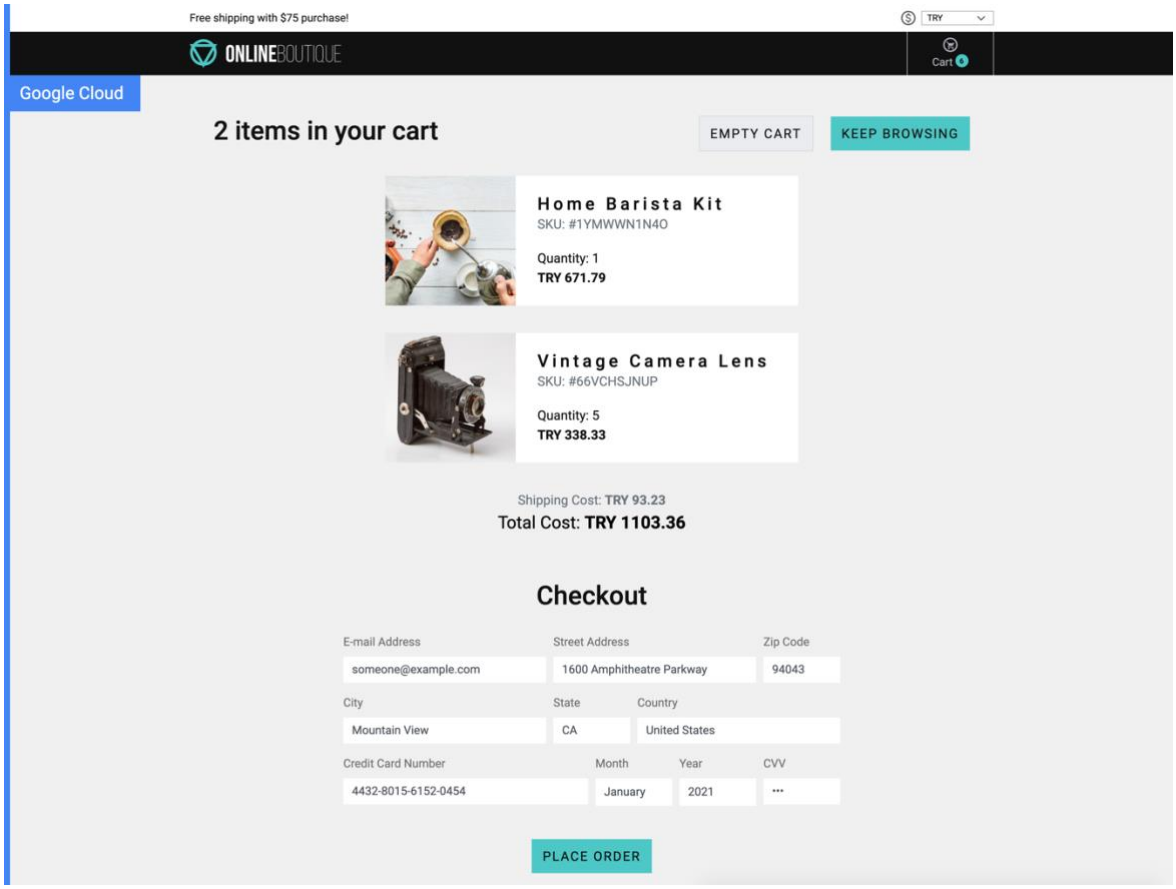
minikube service frontend-external

```
cemalhanalptekin@Cemalhans-MacBook-Pro microservices-demo % minikube service frontend-external
|-----------|-------------------|-------------|---------------------------|
| NAMESPACE |       NAME        | TARGET PORT |            URL            |
|-----------|-------------------|-------------|---------------------------|
| default   | frontend-external | http/80     | http://192.168.49.2:32501 |
|-----------|-------------------|-------------|---------------------------|
🏃  Starting tunnel for service frontend-external.
|-----------|-------------------|-------------|------------------------|
| NAMESPACE |       NAME        | TARGET PORT |          URL           |
|-----------|-------------------|-------------|------------------------|
| default   | frontend-external |             | http://127.0.0.1:50895 |
|-----------|-------------------|-------------|------------------------|
🎉  Opening service default/frontend-external in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

Cemalhan Alptekin

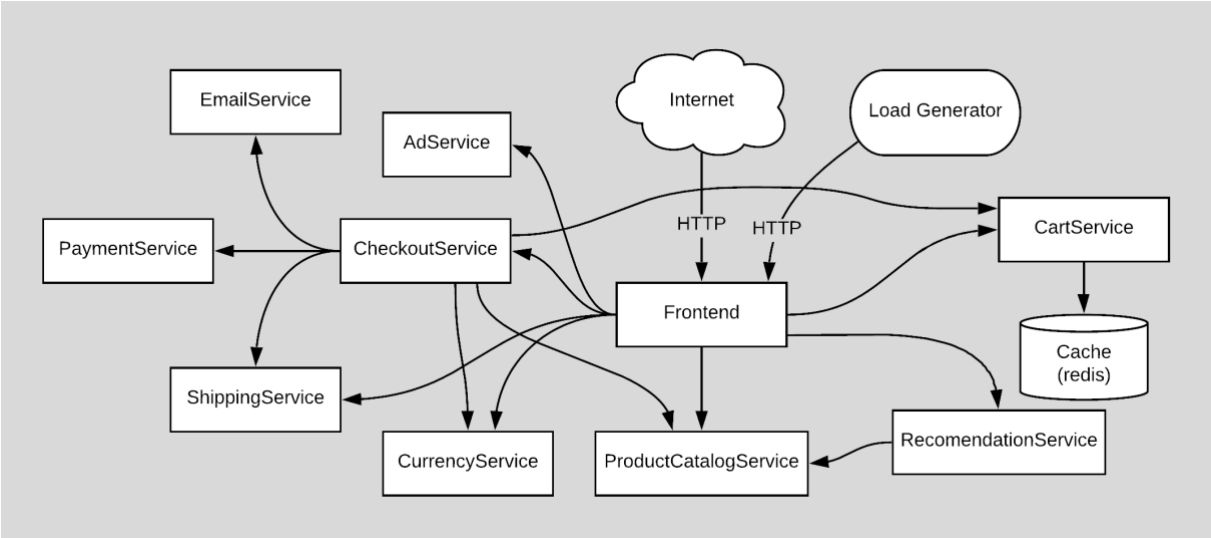
Application Images:





Architecture

Online Boutique is composed of 10 microservices (plus a load generator) written in 5 different languages that communicate with each other over gRPC.



Service	Language	Description
adservice	Java	Provides text ads based on given context words.
cartservice	C#	Stores the items in the user's shopping cart in Redis and retrieves it.
checkoutservice	Go	Retrieves user cart, prepares order and orchestrates the payment, shipping and the email notification.
currencyservice	Node.js	Converts one money amount to another currency. Uses real values fetched from European Central Bank. It's the highest QPS service.
emailservice	Python	Sends users an order confirmation email (mock).
frontend	Go	Exposes an HTTP server to serve the website. Does not require signup/login and generates session IDs for all users automatically.
loadgenerator	Python/Locust	Continuously sends requests imitating realistic user shopping flows to the frontend.
paymentservice	Node.js	Charges the given credit card info (mock) with the given amount and returns a transaction ID.
productcatalogservice	Go	Provides the list of products from a JSON file and ability to search products and get individual products.
recommendationservice	Python	Recommends other products based on what's given in the cart.
shippingservice	Go	Gives shipping cost estimates based on the shopping cart. Ships items to the given address (mock)

Features

- **Kubernetes:** The app is designed to run on Kubernetes
- **gRPC:** Microservices use a high volume of gRPC calls to communicate to each other.
- **OpenTelemetry Tracing:** Most services are instrumented using OpenTelemetry trace providers for gRPC/HTTP.
- **Skaffold:** Application is deployed to Kubernetes with a single command using Skaffold.
- **Synthetic Load Generation:** The application demo comes with a background job that creates realistic usage patterns on the website using Locust load generator.

Monitoring and Logging with Honeycomb

Honeycomb.io is a powerful observability platform designed to help developers and operations teams understand and debug complex systems. It provides real-time insights into the performance and behavior of distributed applications by collecting, analyzing, and visualizing telemetry data such as traces, logs, and metrics. Honeycomb is especially useful for identifying and resolving issues in microservice architectures and cloud-native environments.

Why Use Honeycomb?

- **Distributed Tracing:** Honeycomb excels in providing deep insights into distributed systems through tracing. This allows developers to follow a request as it moves across services, helping to identify performance bottlenecks and errors.
- **High-Cardinality Data:** Honeycomb is designed to handle high-cardinality data efficiently, making it ideal for environments with many unique identifiers (such as user IDs or request paths).
- **Real-Time Debugging:** Honeycomb provides near real-time querying capabilities, which makes it easy to investigate issues as they occur, reducing mean time to resolution (MTTR).
- **Custom Dashboards and Queries:** Users can create custom dashboards and run flexible queries to visualize key performance indicators and analyze application behavior in granular detail.

Integration with OpenTelemetry

Honeycomb integrates seamlessly with **OpenTelemetry**, an open-source observability framework for instrumenting, generating, collecting, and exporting telemetry data (traces, metrics, logs). By configuring OpenTelemetry to export data to Honeycomb, developers can centralize their observability efforts without vendor lock-in.

Key Features:

1. **End-to-End Tracing:** Honeycomb captures traces from services instrumented with OpenTelemetry, offering a clear view of how data flows through an application.
2. **Event-Based Data Model:** Honeycomb uses an event-based model, which provides high granularity and flexibility in monitoring both individual events and aggregated metrics.
3. **Root Cause Analysis:** By correlating logs, traces, and metrics, Honeycomb helps pinpoint the root cause of performance degradation or failures in complex systems.

How Honeycomb Helps in Development and Operations

- **Performance Optimization:** Identify slow or inefficient code paths in your services using Honeycomb's trace visualization tools.
- **Service Reliability:** Detect and resolve service outages or performance issues before they impact customers.
- **Capacity Planning:** Use historical data collected by Honeycomb to make informed decisions about scaling infrastructure.

Cemalhan Alptekin

Honeycomb Images: