Three-Tier Web Application using Docker and Kubernetes

Table of Contents

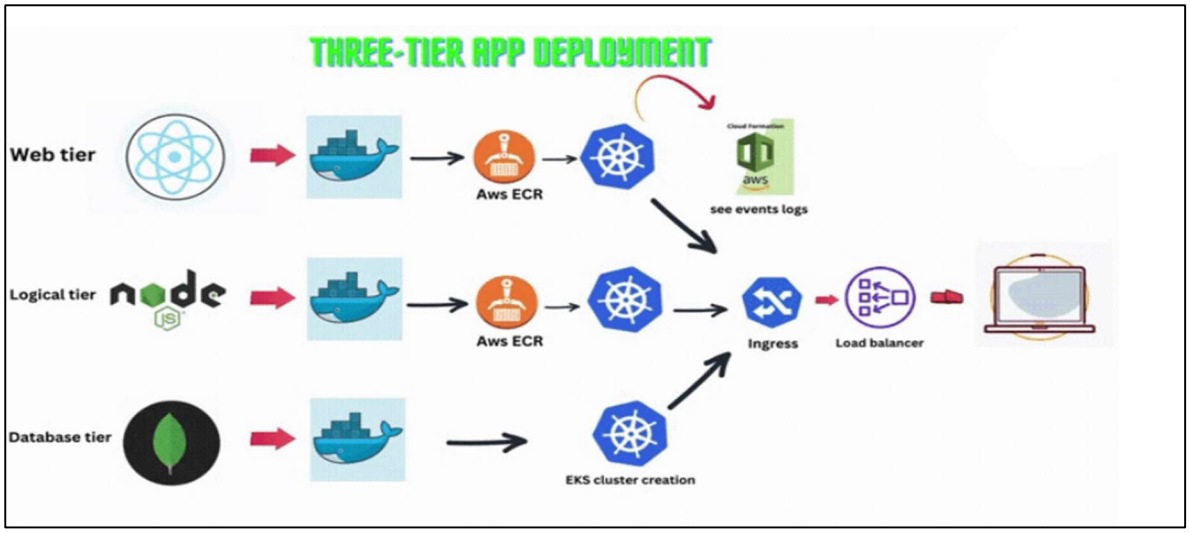
Introduction:

In this case study, we'll explore the setup and deployment of a three-tier web application using Docker and Kubernetes. The application consists of a Fast API backend serving as the API layer and a modern frontend built with Tailwind CSS. Docker will be used for containerization, enabling consistent deployment across different environments, while Kubernetes will handle orchestration for scaling and managing the application components.

Architecture Overview:

The architecture of the three-tier web application is as follows:

- **Presentation Tier (Frontend):** The frontend layer built with Tailwind CSS provides the user interface for interacting with the application.
- **Logic Tier (Backend):** FastAPI serves as the backend logic tier, providing RESTful APIs to communicate with the frontend and interact with the data tier.
- **Data Tier (Database):** The data tier stores and manages the application's data. For simplicity, we'll use a lightweight database such as SQLite, but this can be easily replaced with more robust solutions like PostgreSQL or MySQL.



Prerequisites:

Before proceeding with the setup, ensure you have the following prerequisites installed:

- Docker: Containerization platform for packaging the application components.
- Kubernetes: Container orchestration platform for managing and scaling containerized applications.
- kubectl: Kubernetes command-line tool for interacting with Kubernetes clusters.

Setup Instructions:

Docker Configuration:

- Containerize Backend and Frontend
- Create Dockerfiles for both the backend (FastAPI) and frontend (Tailwind CSS) components.
- Build Docker images for each component using the Dockerfiles.
- Ensure that the Docker images expose the necessary ports for communication.
- Compose Docker Compose File
- Create a Docker Compose file to define the multi-container application.
- Specify services for the backend and frontend, along with any required configurations such as environment variables or volume mounts.

Kubernetes Configuration:

- Create Kubernetes Deployment YAMLs
- Write Kubernetes Deployment YAML files for deploying the backend and frontend components.
- Define the container specifications, resource requirements, and any other configurations.
- Set Up Kubernetes Services:
- Create Kubernetes Service YAML files to expose the backend service internally within the cluster.
- Optionally, configure an Ingress resource to expose the frontend service externally for external access.

Deployment

- Deploy to Local Kubernetes Cluster
- Deploy the application to a local Kubernetes cluster using Minikube or Docker Desktop Kubernetes.
- Apply the Kubernetes YAML files using kubectl apply -f <file.yaml> command
- Monitor Deployment
- Monitor the deployment status using kubectl get pods and kubectl get services commands.
- Ensure that all pods are running and services are accessible.

Conclusion:

In this case study, we've successfully set up and deployed a three-tier web application using Docker for containerization and Kubernetes for orchestration. By containerizing each component and leveraging Kubernetes for managing the deployment, we've achieved a scalable and resilient architecture capable of handling varying workloads.

Appendix

Additional Resources

- Docker Documentation: [link]
- Kubernetes Documentation: [link]

This documentation provides a comprehensive guide for setting up and deploying a three-tier web application using Docker and Kubernetes. For further customization or optimization, refer to the provided resources and sample code.