Final Project: Deployable Chat Service

Agenda

01. **Project Overview**

02. **Chat Service**

03. **Containerization**

04. **Orchestration**

05. **Package Management**

06. **Demo**

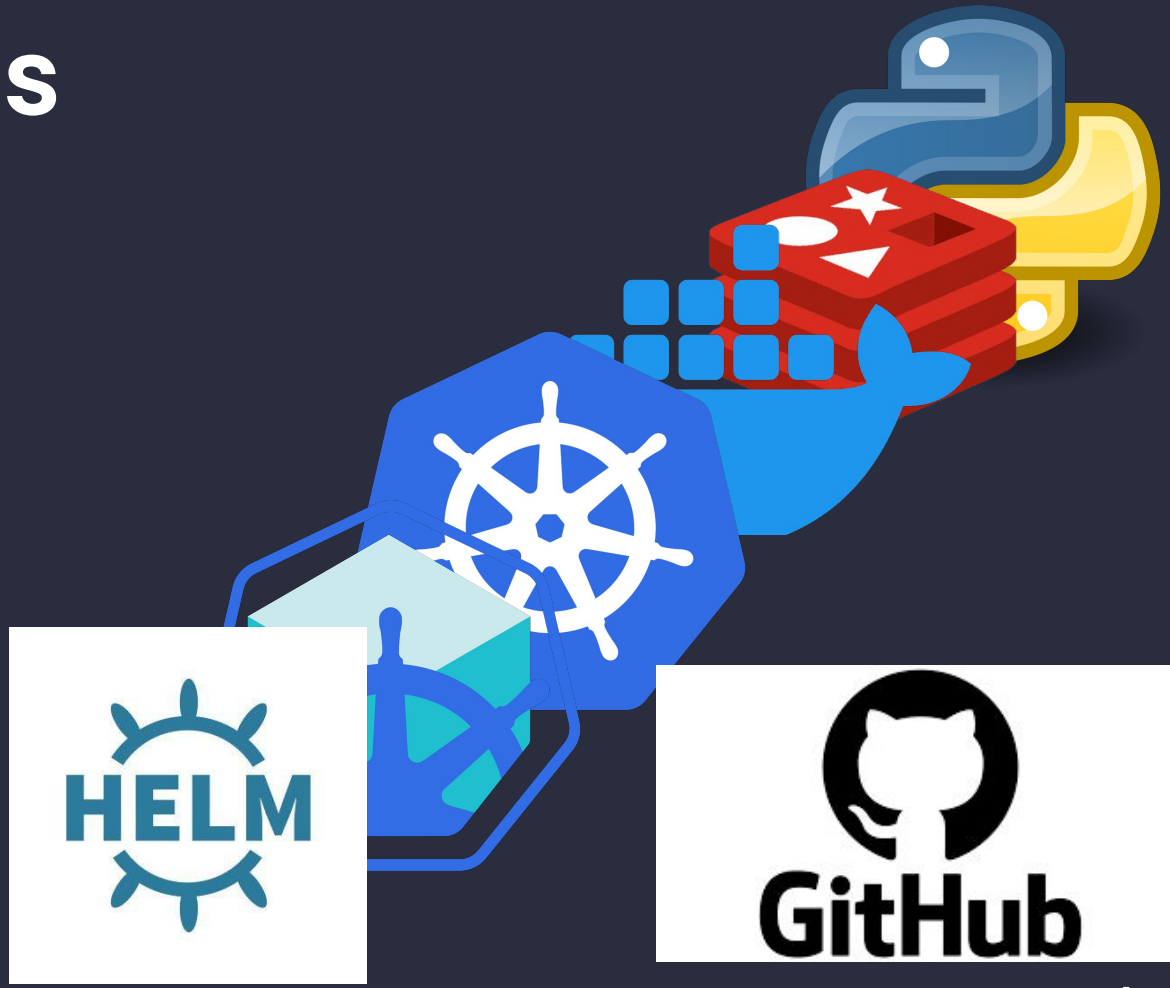
Project Overview

Goals

- Create a simple chat service
  - Central chat server that handles multiple clients
  - Persistent chat history between individual clients and server broadcasts
- Automate management of chat server
  - Containerize the application
  - Orchestration solution
    - Deploys application
    - Scales application
- Load generation to verify success

Technologies

- Python
- Redis
- Docker
- Kubernetes
- Minikube
- Helm
- Git/GitHub



Chat Service

Client

- Modeled after example code
- Command line arguments
  - Required: Server IP, Server Port, Username
  - Optional: Recipient
- Execute
  - Connects to server, sends configuration information
  - Gets message history from server
- Two threads
  - Listening for data from the server to show user
  - Accepting input from user to send to server

Server

- Modeled after example code
- Binds to port specified in command line argument
- Accepted connections run in individual threads
- Directs messages accordingly
  - Client to client messages
  - Server broadcast messages
- Connection to Redis database
  - Saves messages with user and recipient as entries
  - Queries and parses responses to send according message history

Containerization

Dockerfile

- Uses **Python 3.13-slim** as a lightweight base image
- Sets `/app` as the working directory inside the container
- Copies and installs dependencies from `requirements.txt` using pip
- Copies the chat server source code into the image (`src/`)
- Exposes port **6000**, where the chat service runs
- Starts the chat server with `python src/chat_server.py 6000` whenever the container is up
- Organized for efficient caching (requirements copied before code)

Docker Compose

Defines a **multi-container setup**: Redis + Chat Server

Automatically handles networking — containers communicate by service name

Redis container:

- Uses official `redis:7` image
- Persists data via a named volume (`redis_data`)
- Exposes port **6379**

Chat server container:

- Built from the provided Dockerfile
- Depends on Redis (ensures Redis starts first)
- Passes environment variables (`REDIS_HOST`, `REDIS_PORT`)
- Exposes port **6000** for external clients

Simplifies startup from scratch with: `docker compose up --build`

Orchestration

Kubernetes

Why Kubernetes? (Shift from Docker → Kubernetes)

- Docker solved packaging (containers). Kubernetes solves orchestration.
- Automated scheduling & scaling of containers (Replica management).
- Self-healing (restart failed pods, reschedule on node failure).
- Rolling updates & rollbacks (zero-downtime deploys).
- Service discovery & stable networking (ClusterIP, DNS).
- Declarative configuration (manifests/Helm) → reproducible infra.

Setup - Docker desktop, Minikube

K8s Resources Used (what we created & why)

- Used Docker Buildx to create and push a multi-architecture container image.
- Namespace: **chat-app** — isolation for project resources.
- Deployment: **chat-server** — manages replica sets and rolling updates.
- Service: chat-server (NodePort: 30060) — connecting to chat server outside the cluster.
- Created a **PersistentVolumeClaim** so Redis stores messages permanently
- PVC ensures chat messages are not lost even if Redis pod crashes.
- Redis service (ClusterIP: 6379) — Internal Discovery
- HPA — autoscale pods based on CPU/memory.

Client → NodePort (30060) → Chat Pods → ClusterIP → Redis + PVC

Package Management

Helm Deployment

- Kubernetes uses many YAMLs → hard to manage at scale
- Helm = package manager for Kubernetes
- Templates + values → avoid repetition
- Versioned releases → easy upgrades & rollbacks
- Shareable charts → teams standardize deployments

Helm Structure

- Created a Helm chart for the chat server + Redis
- Parameterized replicas, image, ports
- Values file manages namespace, NodePort, resource limits
- Redis PVC + service templated for consistency
- One command deploy: helm install chat-service . -n chat-app

Demo

Questions?

References

- Images
  - https://upload.wikimedia.org/wikipedia/commons/thumb/0/0a/Python.svg/1200px-Python.svg.png
  - https://www.jetbrains.com/guide/assets/thumbnail-0d651b12.png
  - https://www.stackhero.io/assets/src/images/servicesLogos/openGraphVersions/docker.png?d87f4381
  - https://upload.wikimedia.org/wikipedia/commons/thumb/3/39/Kubernetes_logo_without_workmark.svg/960px-Kubernetes_logo_without_workmark.svg.png
  - https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEipF4bE7nbr0yO5GUaTp2BBBe4J5QiQF07IZBIRiGrH7uIfNZHmPLKY27EnqC4AU-DUyJzQ5KZYaOgbUnlDL4PgUo2BiMr-pgvcKqFMuHxBRrpb_NOiPkl2nJc_6_N8mn-TskwCMyLTrEg/s1600/m.png
  - https://miro.medium.com/v2/1*YcByGZNVtb6cGO4FQLr0mw.jpeg
  - https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS-tvx2BFjpYmfiIBLV25XlfVZy4KhCYFLB7w&s