Docker Compose Basics

Definition

Docker Compose is a tool for defining and running multi-container Docker applications. With Docker Compose, you use a YAML file to configure your application's services, networks, and volumes. This allows you to easily manage multi-container setups.

Key features:

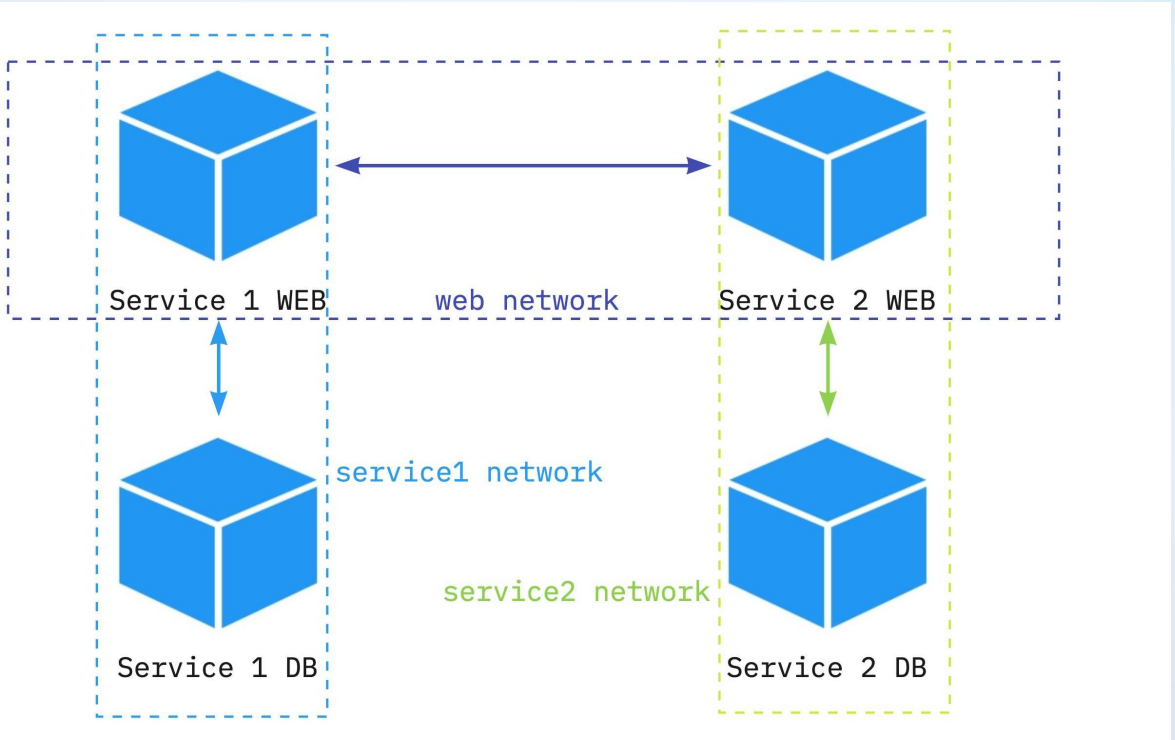
- Simplifies orchestration of services in development and testing environments.
- Centralized configuration through a docker-compose.yml file.
- Supports networking, secrets, and persistent storage configuration

```yaml
version: '3.8'

services:
  app:
    image: my-app:latest
    build:
      context: ./app
      dockerfile: Dockerfile
    networks:
      - backend
    environment:
      - APP_ENV=production
    depends_on:
      - db
    ports:
      - "8080:8080"
    volumes:
      - ./data:/app/data

  db:
    image: postgres:13
    secrets:
      - db_password
    environment:
      POSTGRES_PASSWORD_FILE: /run/secrets/db_password
    networks:
      - backend

networks:
  backend:

secrets:
  db_password:
    file: ./secrets/db_password.txt
```

Networking

Docker Compose provides built-in networking capabilities to allow services to communicate with each other

- **Automatic Networking**: All services are added to a default network, enabling communication via service names.
- **Custom Networks**: You can define custom networks in the docker-compose.yml file for isolation and control.
- **Aliases**: Custom names for services in a network can be configured.
- **Bridge Network Mode**: This is the default network mode for containers in Docker Compose.



Networking

Configuration example.

Each container can look up each container's name in the same network. For example, the app's application code could now connect to db using `postgres://db:5432` link.

```yaml
networks:
  backend:
  frontend:

services:
  app:
    image: my-app
    networks:
      - backend
  db:
    image: postgres
    networks:
      - backend
  web:
    image: nginx
    networks:
      - frontend
```

Managing secrets

Secrets provide a secure way to pass sensitive information, such as passwords, API keys, or configuration details, to services.

How secrets work:

- Secrets are defined in the docker-compose.yml file or as external files.
- Secrets are mounted as files in the container at runtime and are available at `/run/secrets/<secret_name>` in Linux and `C:\ProgramData\Docker\secrets` in Windows containers.
- Access is restricted to services that explicitly request the secrets.

```yaml
services:
  db:
    image: mysql:latest
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_root_password
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD_FILE: /run/secrets/db_password
    secrets:
      - db_root_password
      - db_password

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD_FILE: /run/secrets/db_password
    secrets:
      - db_password


secrets:
  db_password:
    file: db_password.txt
  db_root_password:
    file: db_root_password.txt

volumes:
    db_data:
```

Main CLI commands

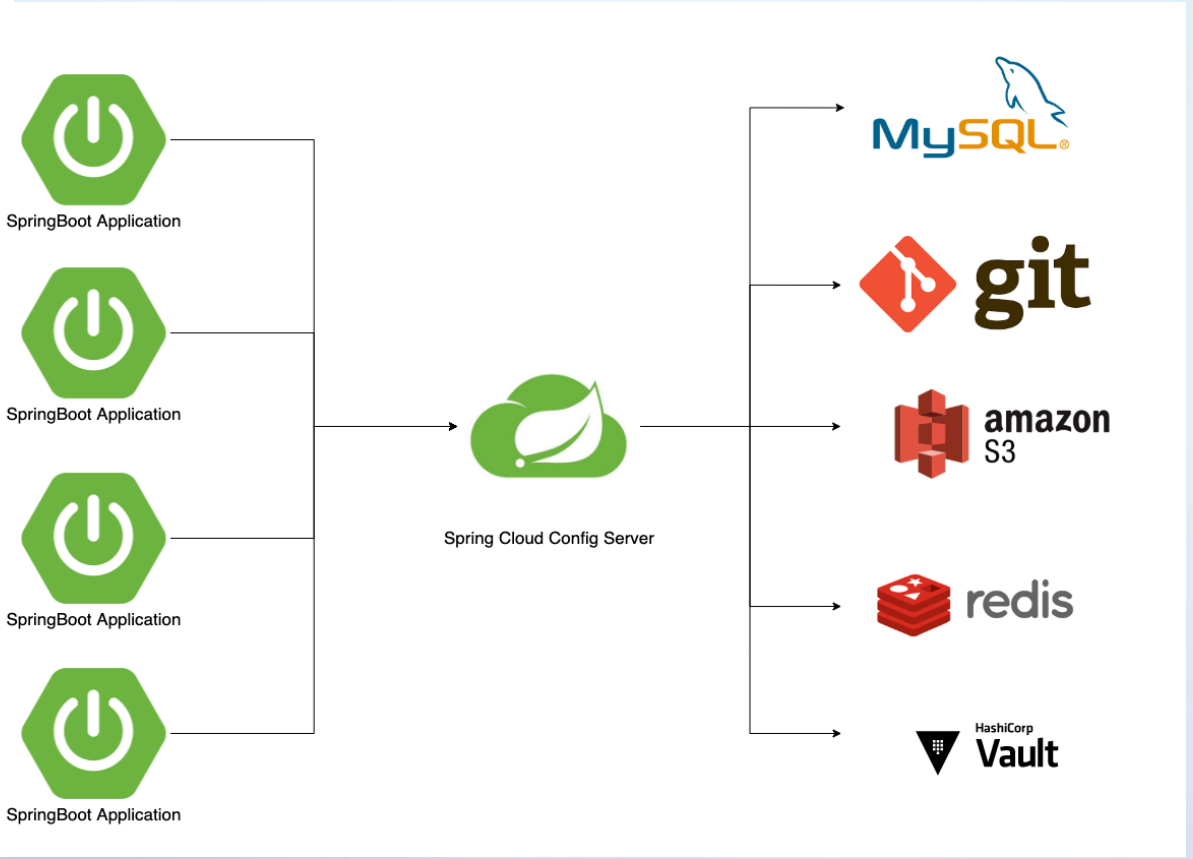
Command	Description
`docker-compose up`	Creates and starts containers defined in the YAML file.
`docker-compose down`	Stops and removes containers, networks, and volumes.
`docker-compose ps`	Lists running containers.
`docker-compose logs`	Displays logs for running containers.
`docker-compose exec [service] [command]`	Runs a command inside a running container.
`docker-compose build`	Builds or rebuilds the services.
`docker-compose stop`	Stops running containers without removing them.
`docker-compose start`	Starts existing containers.
`docker-compose restart`	Restarts services.

Main keywords

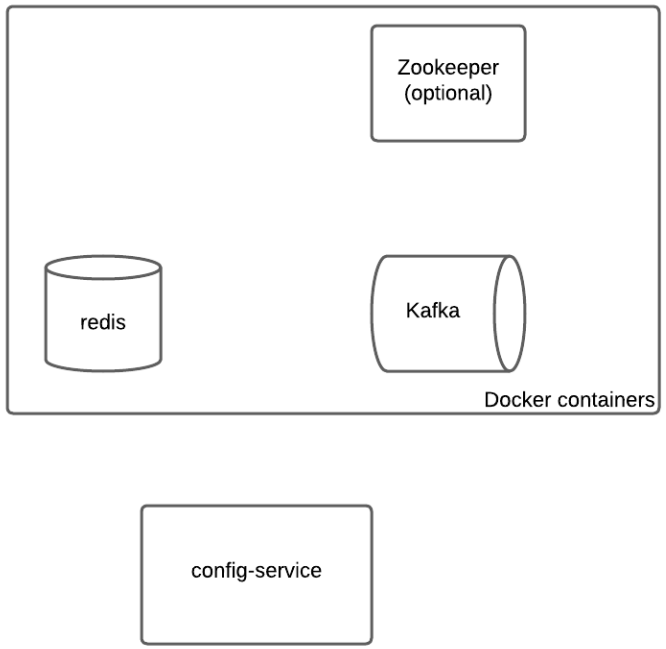
Key	Description
`image`	Specifies the Docker image to use for the container.
`build`	Defines build instructions for a custom image (e.g., `context` and `dockerfile`).
`networks`	Specifies the networks the service should connect to.
`environment`	Sets environment variables for the container.
`depends_on`	Specifies dependencies between services, ensuring order of startup.
`ports`	Maps container ports to host ports.
`volumes`	Mounts directories or files from the host into the container.
`secrets`	Provides secure access to sensitive data for the service.
`command`	Overrides the default command for the container.
`restart`	Configures the restart policy (e.g., `always`, `on-failure`).

What is Spring Cloud Config

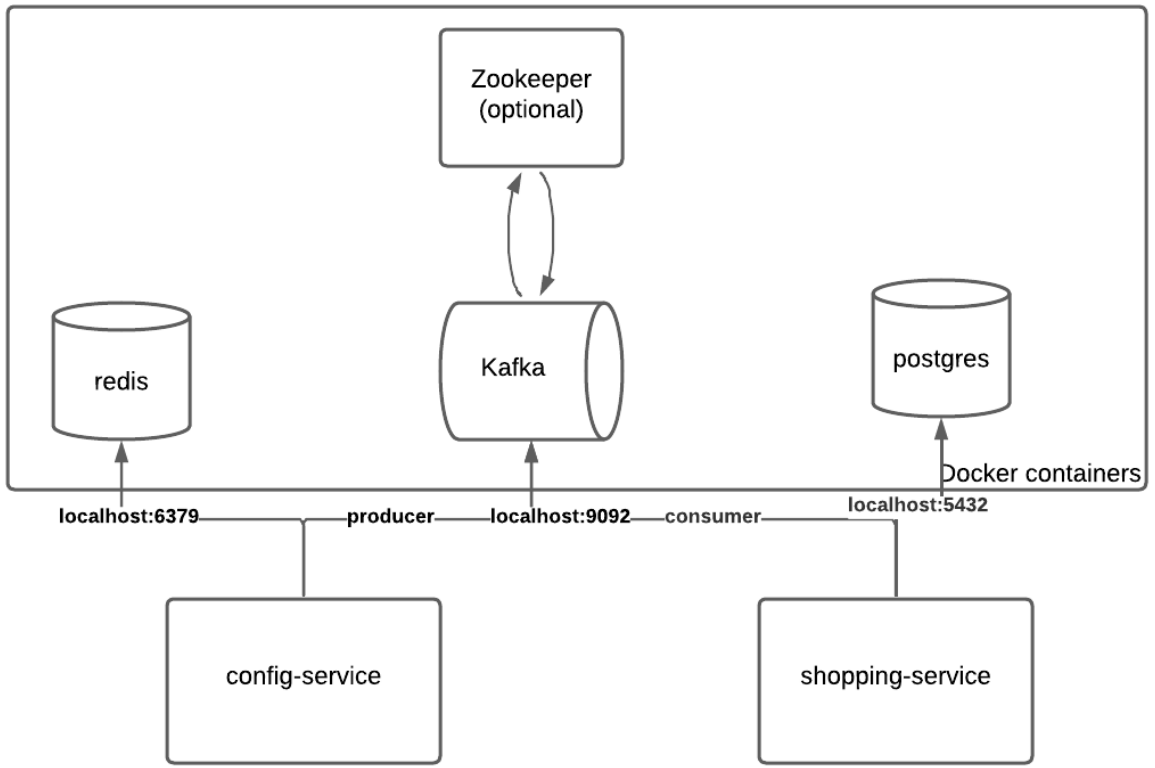
Spring Cloud Config provides server-side and client-side support for **externalized configuration** in a distributed system.
With the Config Server, you have a **central place to manage external properties** for applications across all environments



Assignment. Task 1

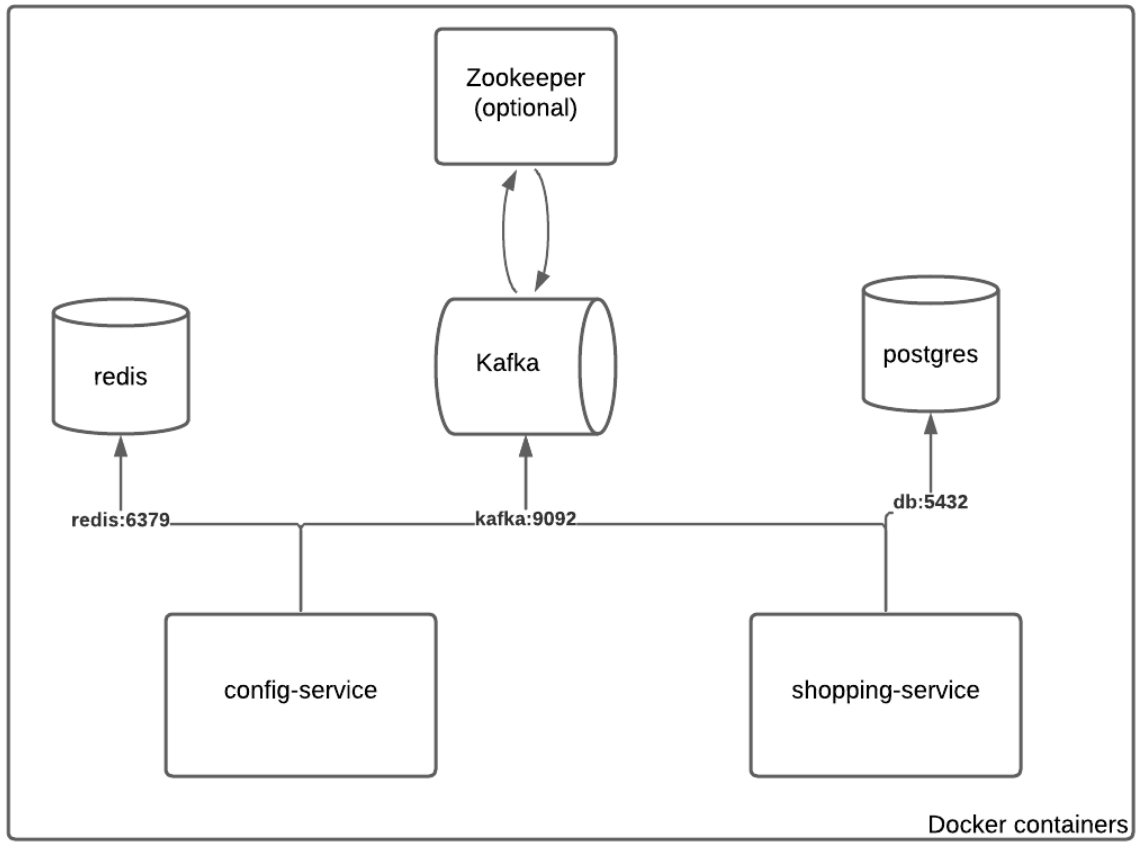


Assignment. Task 2

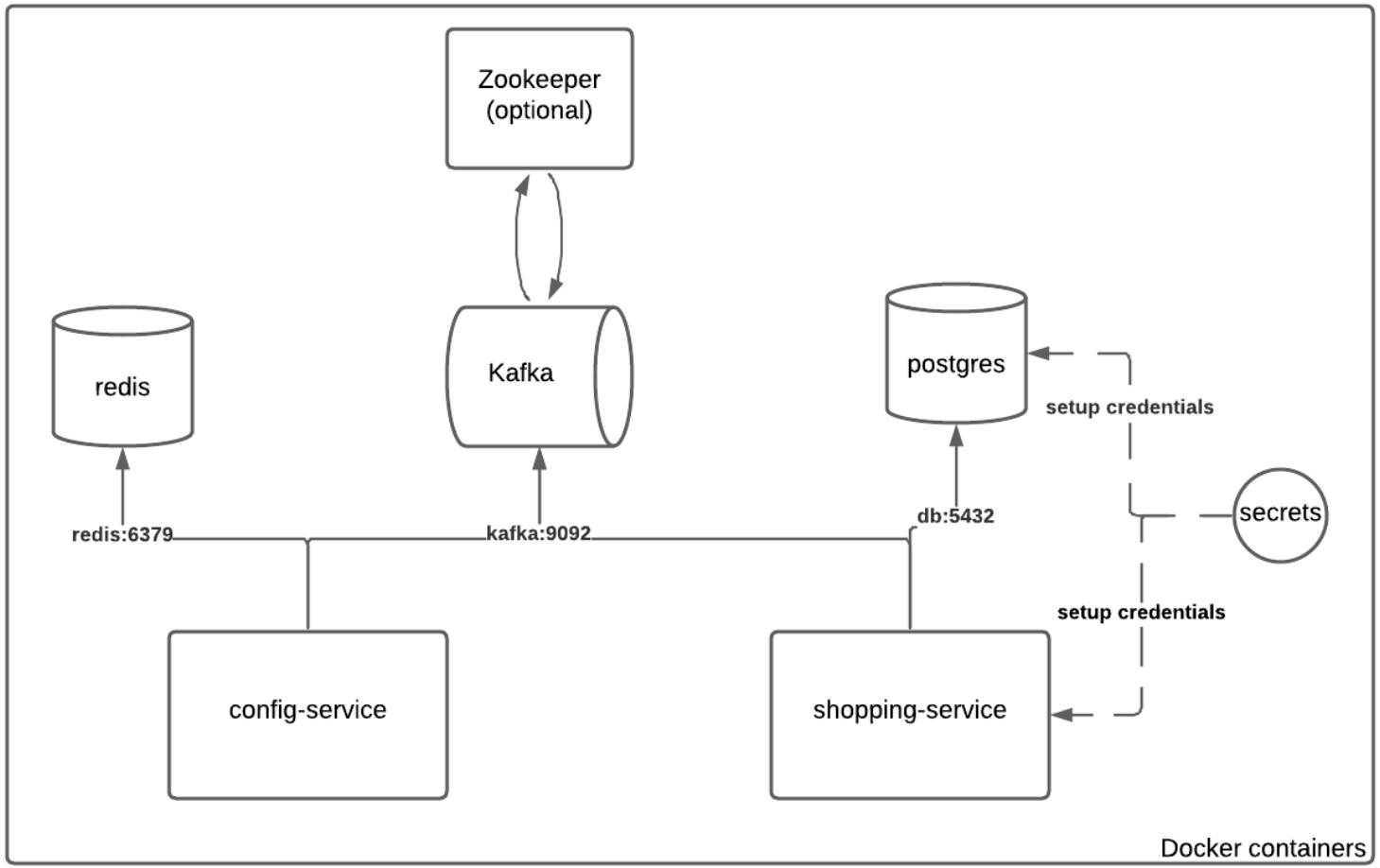


CODEUS_

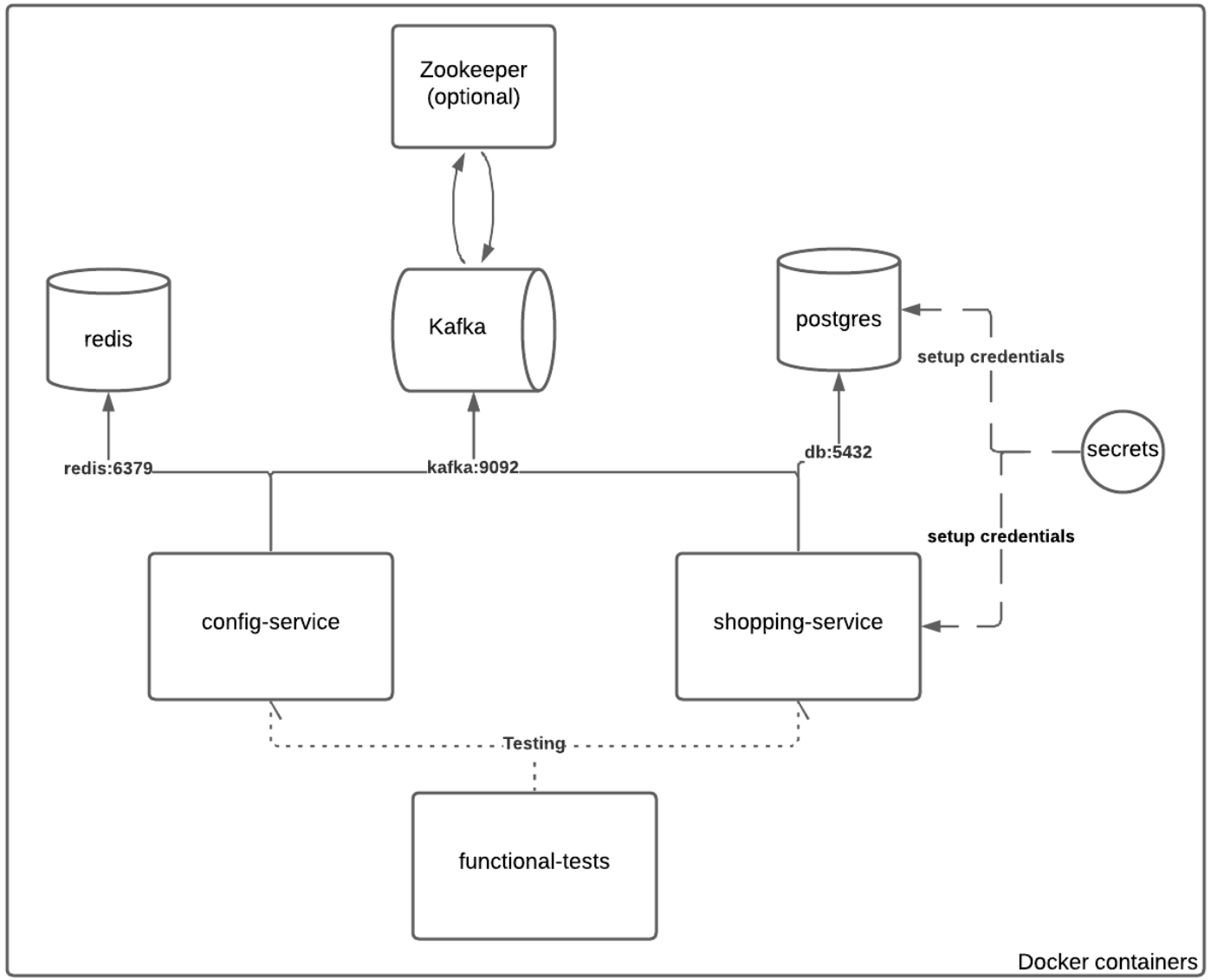
Assignment. Task 3. Step 1



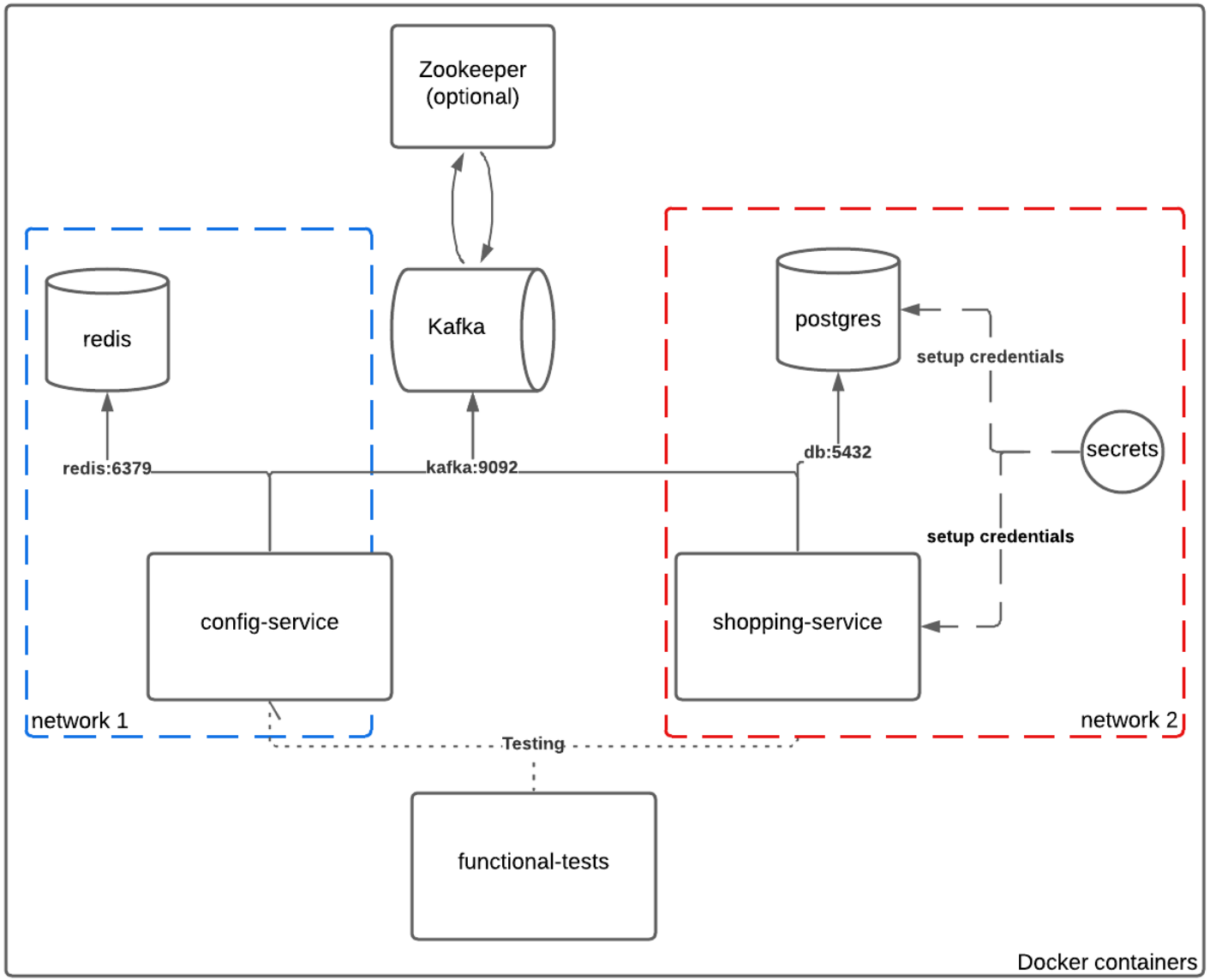
Assignment. Task 3. Step 2



Assignment. Task 3. Step 3



Assignment. Task 3. Step 4



Thank you

- Author: Yevhenii Savonenko
- My LinkedIn: www.linkedin.com/in/yevhenii-savonenko-624a32172
- Date: January 2025
- Join Codeus community in Discord
- Join Codeus community in LinkedIn