Set up a Kubernetes environment to deploy a simple e-commerce application with a frontend, backend, and database. Create a namespace called `ecommerce` and label all resources within this namespace with `app=ecommerce`. The application consists of three components:

1. **Frontend:** Use the Nginx image `nginx:latest` as a web server. Expose it via a Kubernetes Service of type `ClusterIP` named `frontend-service`. Create an Ingress resource that routes external traffic to this service based on the `/` path.
2. **Backend:** Deploy a Python Flask application using the image `tiangolo/uwsgi-nginx-flask:python3.8`. This backend should be exposed by a Kubernetes Service of type `ClusterIP` named `backend-service`. Ensure the backend service is accessible by the frontend using the service name.
3. **Database:** Use the MySQL image `mysql:5.7` for the database. Deploy it using a StatefulSet named `mysql` to maintain persistent storage. Expose it internally with a Service of type `ClusterIP` named `mysql-service`. Securely manage the MySQL root password using a Secret named `ecommerce-secret`, with the key `MYSQL_ROOT_PASSWORD`.

For configuration management, create a ConfigMap named `ecommerce-config` to store non-sensitive data, such as the Flask app's environment variables (`FLASK_ENV=production`). The backend should connect to the MySQL database using the credentials stored in the secret.

Deploy the `frontend` and `backend` using Kubernetes Deployments, each with a replica count of 2 for high availability. Deploy the `database` using the StatefulSet to ensure persistent data storage.

Verify the deployment by accessing the frontend via the Ingress, confirming that it successfully interacts with the backend and retrieves data from the MySQL database. Ensure that all components are correctly placed within the `ecommerce` namespace and properly labeled, and check the logs to confirm there are no errors in the setup.