User Microservice - Complete Guide

# Step 1: User Module - Java Code Structure

Below is a basic Java Spring Boot microservice for a User module:

## User Entity

package com.example.user.model;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
  
@Entity  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String username;  
 private String email;  
  
 // Getters and Setters  
}

## User Repository

package com.example.user.repository;  
  
import com.example.user.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface UserRepository extends JpaRepository<User, Long> {}

## User Service

package com.example.user.service;  
  
import com.example.user.model.User;  
import com.example.user.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class UserService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 public List<User> getAllUsers() {  
 return userRepository.findAll();  
 }  
  
 public User saveUser(User user) {  
 return userRepository.save(user);  
 }  
}

## User Controller

package com.example.user.controller;  
  
import com.example.user.model.User;  
import com.example.user.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/users")  
public class UserController {  
  
 @Autowired  
 private UserService userService;  
  
 @GetMapping  
 public List<User> getUsers() {  
 return userService.getAllUsers();  
 }  
  
 @PostMapping  
 public User createUser(@RequestBody User user) {  
 return userService.saveUser(user);  
 }  
}

## Application Properties

spring.datasource.url=jdbc:h2:mem:userdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true

## Main Class

package com.example.user;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class UserSvcApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(UserSvcApplication.class, args);  
 }  
}

# Step 2: CI/CD with Jenkins

Follow these steps to set up CI/CD with Jenkins:

1. Install Jenkins on your Linux machine.  
2. Configure Jenkins:  
 - Install required plugins (Git, Docker, Maven).  
 - Create a Freestyle Job.  
 - Connect GitHub repo (add credentials and repo URL).  
 - Under Build Steps, add:

mvn clean package  
docker build -t user-svc .  
docker stop user-svc || true && docker rm user-svc || true  
docker run -d -p 8080:8080 --name user-svc user-svc

3. Add a webhook from GitHub (or poll SCM) to auto-trigger the build on push.

# Step 3: Hosting in Docker

Build and run the app using Docker:

## Dockerfile

FROM openjdk:17-jdk-alpine  
WORKDIR /app  
COPY target/user-svc-0.0.1-SNAPSHOT.jar user-svc.jar  
EXPOSE 8080  
ENTRYPOINT ["java", "-jar", "user-svc.jar"]

Commands to run:  
1. mvn clean package  
2. docker build -t user-svc .  
3. docker run -p 8080:8080 user-svc  
  
Access the app: http://<your-ip>:8080/users