Учитывая, что OpenShift платный и сложный в настройке, эти лабораторные работы построены на использовании **Minikube** (или K3s) для Kubernetes и **Docker Swarm**. Это позволит понять основные принципы без лишней головной боли с инфраструктурой.

Лабораторная работа №1: Задача масштабирования (Docker -> Swarm)

Цель: Изучить разницу между простым запуском контейнера и оркестрацией сервиса. Эта работа закрывает пробел между "работает на моем ноутбуке" и "работает в кластере".

- **Сценарий:** Вы выступаете в роли DevOps-инженеров простого интернет-магазина, которому нужно справиться с наплывом трафика (например, в "Черную пятницу").
- **Приложение:** Простое двухуровневое приложение:
  - **Frontend:** Веб-страница на Python/Flask, отображающая счетчик посещений.
  - **Backend:** База данных Redis для хранения количества хитов.
- **Ход работы:**
  1. **Контейнеризация:** Написать Dockerfile для Python-приложения.
  2. **Docker Compose:** Написать docker-compose.yml для локального запуска связки App + Redis. Убедиться, что они "видят" друг друга.
  3. **Инициализация Swarm:** Перевести Docker в режим Swarm (docker swarm init).
  4. **Развертывание стека (Deploy Stack):** Конвертировать файл compose в деплой стека (Stack Deploy).
  5. **Тест на отказ (Self-healing):** Вы вручную "убиваете" (docker kill) активный контейнер и наблюдают, как Swarm автоматически пересоздает его.
  6. **Масштабирование:** Выполнить команду docker service scale frontend=5.
- **Ключевые знания:** Обнаружение сервисов (Service Discovery — как приложение находит Redis по имени), реплики и самовосстановление системы.

Лабораторная работа №2: Миграция в Kubernetes (Переход к Cloud-Native)

Цель: Перенести нагрузку из Лабораторной №1 в Kubernetes. Так как мы избегаем OpenShift, используйте **Minikube** или **Kind** (Kubernetes in Docker). Это научит работать с примитивами (Pods, Deployments, Services), которые OpenShift обычно скрывает за абстракциями.

- **Инструменты:** Minikube (или Kind), kubectl.
- **Ход работы:**
  1. **Трансляция (Translation):** Вы должны вручную "перевести" логику Docker Compose в манифесты K8s:
     - deployment.yaml для приложения Python (определение образа и реплик).
     - deployment.yaml для Redis.
     - service.yaml (тип ClusterIP) для внутреннего доступа к Redis.
     - service.yaml (тип NodePort или LoadBalancer) для внешнего доступа к приложению.
  2. **Развертывание:** Применить манифесты к локальному кластеру (kubectl apply).
  3. **Rolling Update (Обновление без простоя):** Вы меняете код Python (например, цвет фона), собирают образ версии v2, обновляют YAML-файл и наблюдают, как Kubernetes выполняет постепенное обновление подов без остановки сервиса.
- **Ключевые знания:** Жизненный цикл Pod'а, разница между императивным и декларативным подходом, внутренняя сеть кластера.

Лабораторная работа №3: Шлюз «Умной фабрики» (Симуляция Edge/Fog)

Цель: Смоделировать среду Edge/Fog, где данные обрабатываются *локально* (на "туманном" уровне) перед отправкой в Облако. Это критически важно для вашего курса, так как демонстрирует работу шлюзов (Gateways).

- **Сценарий:** На заводе установлены датчики температуры. Нужен "Туманный узел" (Шлюз), который фильтрует нормальные показатели, но немедленно оповещает "Облако", если температура превышает норму.
- **Архитектура (все запускается в Docker):**
  - **Edge Device (Датчик):** Простой скрипт (контейнер Python), который генерирует случайную температуру и публикует её в топик MQTT каждую секунду.
  - **Fog Node (Шлюз):** Контейнер с **Node-RED**. Он подписывается на данные датчика.
    - *Логика:* Если Темп < 80, ничего не делать (или писать в локальный лог). Если Темп > 80, переслать в Облако.
  - **Cloud (Центр):** Контейнер **Mosquitto** (MQTT брокер) или простой контейнер-дашборд, который принимает только критические алерты.
- **Ход работы:**
  1. **Настройка инфраструктуры:** Создать специализированную сеть в Docker (factory-net).
  2. **Настройка Шлюза:** Вы открываете визуальный редактор Node-RED (localhost:1880). Они перетаскивают блоки (nodes) для создания логики: *Вход*

MQTT -> Switch Node (Если > 80) -> Выход MQTT (Топик Облака).
3. **Запуск симуляции:** Запуск контейнера "Датчика".
4. **Наблюдение:** Вы видите, что контейнер "Облако" получает лишь малую часть трафика. Это демонстрирует экономию пропускной способности (Bandwidth savings) — ключевой принцип туманных вычислений.

- **Ключевые знания:** Протокол MQTT, фильтрация на краю (Edge Filtering), оптимизация трафика, логика работы шлюзов.

Почему это заменяет OpenShift:

1. **OpenShift** — это, по сути, Kubernetes с надстройками безопасности и удобной консолью разработчика.
2. Используя **Minikube** (Лаб. 2), вы учитесь работать с "движком", на котором работает OpenShift, без сложностей с лицензиями.
3. Используя **Node-RED** (Лаб. 3), вы визуализируете "поток" данных (Flow), что гораздо понятнее для изучения концепции Fog Computing, чем написание сложного кода на C++ или Go.