sudo usermod -aG docker $USER && newgrp docker

вот текст моего дз:

Домашнее задание: Развёртывание распределённой системы логирования и хранения с резервным копированием

1. **Создать пользовательское веб-приложение (API)**
   Приложение должно реализовать следующие REST-эндпоинты:

   - `GET /` — возвращает строку `"Welcome to the custom app"`
   - `GET /status` — возвращает JSON `{"status": "ok"}`
   - `POST /log` — принимает JSON `{"message": "some log"}` и записывает его в файл `/app/logs/app.log`
   - `GET /logs` — возвращает содержимое файла `/app/logs/app.log`

   Приложение должно:

   - Писать логи в `/app/logs/app.log`
   - Использовать конфигурационные параметры (например, уровень логирования, порт, заголовок приветствия) из ConfigMap

2. **Развернуть приложение как Pod для начального теста**

   - Написать Dockerfile для приложения
   - Создать Pod, монтирующий:
     - `emptyDir` volume в `/app/logs`
     - ConfigMap с настройками в `/app/config` (или через переменные окружения)

3. **Развернуть приложение как Deployment**

   - Создать Deployment с 3 репликами
   - Настроить монтирование `emptyDir` для логов
   - Обновить Deployment, чтобы изменения в ConfigMap автоматически применялись
   - Проверить через Service и `kubectl port-forward`, что API работает

4. **Создать Service для балансировки нагрузки**

   - ClusterIP-сервис, направляющий трафик на поды приложения
   - Проверить: `curl http://<service-name>/logs` и `curl -X POST http://<service-name>/log -d '{"message": "test"}'`
   - Убедиться, что запросы распределяются между подами

5. **Развернуть DaemonSet с log-agent**

   - DaemonSet должен:
     - Быть запущен на каждом узле
     - Собирать логи приложения из подов (через `hostPath` или `emptyDir`, при наличии доступа)
     - Перенаправлять логи во stdout или сохранять локально на узле

PROF

- Проверить, что `kubectl logs <log-agent-pod>` содержит записи из `app.log`

6. **Развернуть CronJob для архивирования логов**
   - CronJob должен запускаться раз в 10 минут
   - Команда: `tar -czf /tmp/app-logs-<timestamp>.tar.gz /app/logs/`
   - Логи берутся с сервисов приложения через HTTP API `/logs` (например, `curl`) или из общей директории, если доступна
   - Результат сохраняется в контейнере в `/tmp` (внутри пода CronJob)

7. **Создать единый bash-скрипт `deploy.sh` для автоматического развёртывания всей системы**
   - Скрипт должен:
     - Создавать все необходимые **ConfigMap**, **Pod**, **Deployment**, **Service**, **DaemonSet**, **StatefulSet**, **CronJob** и другие объекты
     - Использовать команды `kubectl apply -f` с заранее подготовленными YAML-файлами
     - Ожидать готовности ключевых компонентов
   - В **README.md** проекта добавьте команду для запуска скрипта из терминала

помоги мне сделать дз
у меня UBUNTU, minikube (через docker)
давай делать постепенно, я буду проверять тебе файл, ты будешь проверять его на соответствие дз и на противочия с другими файлами, если его нужно изменить, то пиши полностью новый

первый файл app.py (Исправленный):

PROF

```
from flask import Flask, request, jsonify
import os
import logging
from datetime import datetime

app = Flask(__name__)

# Configuration from environment variables
welcome_message = os.environ.get('WELCOME_MSG', 'Welcome to the custom
app')
log_level = os.environ.get('LOG_LEVEL', 'INFO')
port = int(os.environ.get('PORT', 5000))

# Configure application logging
logging.basicConfig(
    level=log_level,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/app.log'),
        logging.StreamHandler()
    ]
)
```

```
# Ensure logs directory exists
logs_dir = '/app/logs'
os.makedirs(logs_dir, exist_ok=True)

@app.route('/')
def home():
    return welcome_message

@app.route('/status')
def status():
    return jsonify({"status": "ok"})

@app.route('/log', methods=['POST'])
def log_message():
    data = request.get_json()
    if not data or 'message' not in data:
        return jsonify({"error": "Invalid data"}), 400

    try:
        logging.info(data['message'])
        return jsonify({"status": "logged"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/logs')
def get_logs():
    try:
        with open('/app/logs/app.log', 'r') as f:
            logs = f.read()
        return logs.replace('\n', '<br>')  # Сохраняем форматирование
для браузера
    except FileNotFoundError:
        return jsonify({"error": "Logs not found"}), 404

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=port)
```

PROF

второй файл Dockerfile (Исправленный):

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app.py .

EXPOSE 5000
```

```
CMD ["python", "app.py"]
```

также вот requirements.txt:

```
Flask==2.0.3
Werkzeug==2.0.3  # Явно указываем совместимую версию
```

третий файл (без изменений):
k8s/configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  WELCOME_MSG: "Welcome to the custom app"
  LOG_LEVEL: "INFO"
  PORT: "5000"
```

четвертый файл k8s/deployment.yaml (исправленный):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: custom-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: custom-app
  template:
    metadata:
      labels:
        app: custom-app
    spec:
      containers:
      - name: app
        image: custom-app
        imagePullPolicy: Never
        ports:
        - containerPort: 5000
        envFrom:
        - configMapRef:
            name: app-config
        volumeMounts:
```

```
        - name: logs-volume
          mountPath: /app/logs
      volumes:
      - name: logs-volume
        hostPath:
          path: /var/log/app
          type: DirectoryOrCreate
```

пятый файл (Исправленный):

```
apiVersion: v1
kind: Service
metadata:
  name: custom-app-svc
spec:
  type: ClusterIP
  selector:
    app: custom-app
  ports:
  - name: http
    port: 80
    targetPort: 5000
    protocol: TCP
```

шестой файл:
k8s/daemonset.yaml (исправленный)

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: log-agent
spec:
  selector:
    matchLabels:
      app: log-agent
  template:
    metadata:
      labels:
        app: log-agent
    spec:
      containers:
      - name: log-collector
        image: busybox
        command: ["sh", "-c"]
        args:
          - tail -n +1 -F /host-logs/app.log
        volumeMounts:
        - name: host-logs
```

PROF

```
            mountPath: /host-logs
            readOnly: true
        volumes:
        - name: host-logs
          hostPath:
            path: /var/log/app
            type: DirectoryOrCreate
```

седьмой файл
k8s/cronjob.yaml:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: log-archiver
spec:
  schedule: "*/10 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: archiver
            image: alpine
            command: ["sh", "-c"]
            args:
              - |
                TS=$(date +%s);
                cp /host-logs/app.log /tmp/app.log;
                tar -czf /host-archives/app-logs-${TS}.tar.gz -C /tmp
app.log;
                echo "Archive created: /host-archives/app-
logs-${TS}.tar.gz";
            volumeMounts:
            - name: host-logs
              mountPath: /host-logs
              readOnly: true
            - name: host-archives
              mountPath: /host-archives
          volumes:
          - name: host-logs
            hostPath:
              path: /var/log/app
              type: Directory
          - name: host-archives
            hostPath:
              path: /var/log/app-archives
              type: DirectoryOrCreate
          restartPolicy: OnFailure
```

восьмой файл
deploy.sh (исправленный):

```
#!/bin/bash
set -eo pipefail

# Initialize Minikube
echo "Starting Minikube..."
minikube start --driver=docker
eval $(minikube docker-env)

# Build app image
echo "Building Docker image..."
docker build -t custom-app .

# Create host directories
minikube ssh "sudo mkdir -p /var/log/app /var/log/app-archives && sudo
chmod 777 /var/log/app /var/log/app-archives"

# Apply Kubernetes manifests
declare -a manifests=(
    "k8s/configmap.yaml"
    "k8s/deployment.yaml"
    "k8s/service.yaml"
    "k8s/daemonset.yaml"
    "k8s/cronjob.yaml"
)

echo "Applying manifests..."
for manifest in "${manifests[@]}"; do
    if [ ! -f "$manifest" ]; then
        echo "Error: Missing $manifest"
        exit 1
    fi
    minikube kubectl -- apply -f "$manifest"
done

# Wait for components
echo "Waiting for deployment rollout..."
minikube kubectl -- rollout status deployment/custom-app --timeout=180s

echo "Checking DaemonSet..."
minikube kubectl -- rollout status daemonset/log-agent --timeout=120s

echo -e "\nDeployment complete!"
echo -e "Access endpoints with:"
echo -e "1. kubectl port-forward service/custom-app-svc 8080:80"
echo -e "2. curl http://localhost:8080"
echo -e "\nView agent logs with:"
echo -e "kubectl logs -l app=log-agent"
```

```
echo -e "\nView archives in Minikube:"
echo -e "minikube ssh 'ls -lh /var/log/app-archives'"
```

Помоги мне сделать второе дз:

Добавление Istio в существующую Kubernetes-систему

1. Настроить Istio Gateway и обеспечить внешний доступ

- Настройте объект `Gateway`, принимающий HTTP-трафик на порт 80.
- Настройте `VirtualService`, который подключён к этому Gateway.

2. Настроить маршруты в VirtualService

Создайте объект `VirtualService`, который:

- Обрабатывает все внешние запросы, поступающие через Gateway.
- Делает маршрутизацию на основное приложение
- Все неизвестные маршруты (например, `/wrong`) должны возвращать 404 ошибку

3. Настроить DestinationRule для управления соединениями

Для каждого сервиса, на который маршрутизируется трафик (например, `app-service`, `log-service`), настройте объект `DestinationRule` со следующими параметрами:

- Балансировка нагрузки:
    - используйте алгоритм `LEAST_CONN`, чтобы трафик направлялся туда, где меньше всего текущих подключений.
- Ограничение соединений:
    - максимум 3 одновременных TCP-соединений
    - максимум 5 ожидающих HTTP-запросов
- Включите защищённую межсервисную коммуникацию внутри mesh-а (`ISTIO_MUTUAL` TLS-режим)

PROF

4. Настроить отказоустойчивость и политику доставки

Для маршрута `POST /log` (в VirtualService) реализуйте поведение при сбоях:

- Добавьте искусственную задержку ответа — 2 секунды.
- Установите общий таймаут — 1 секунда (чтобы запрос завершался с ошибкой по таймауту).
- Разрешите повторные попытки — до 2 попыток в случае неудачи.

Ожидаемый результат

- Все конфигурации (`Gateway`, `VirtualService`, `DestinationRule`) должны быть оформлены в отдельных YAML-файлах.

- bash-скрипт `deploy.sh` из предыдущего задания должен быть модифицирован и, помимо, применения новых манифестов, должен предварительно настроить istio service mesh в кластере

Давай будем его постепенно выполнять
начни с 1го пункта.

добавленные файлы:
k8s/istio-gateway.yaml

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: custom-app-dr
spec:
  host: custom-app-svc.default.svc.cluster.local
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
    loadBalancer:
      simple: LEAST_CONN
    connectionPool:
      tcp:
        maxConnections: 3
      http:
        http1MaxPendingRequests: 5
```

k8s/istio-virtualservice.yaml

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: custom-app-vs
spec:
  hosts:
    - "*"
  gateways:
    - custom-gateway
  http:
    - match:
        - uri:
            exact: /
      route:
        - destination:
            host: custom-app-svc.default.svc.cluster.local
            port:
              number: 80

    - match:
```

PROF

```
        - uri:
            exact: /status
      route:
        - destination:
            host: custom-app-svc.default.svc.cluster.local
            port:
              number: 80

    - match:
        - uri:
            exact: /log
          method:
            exact: POST
      fault:
        delay:
          percentage:
            value: 100
          fixedDelay: 2s
      timeout: 1s
      retries:
        attempts: 2
        perTryTimeout: 0.5s
      route:
        - destination:
            host: custom-app-svc.default.svc.cluster.local
            port:
              number: 80

    - match:
        - uri:
            exact: /logs
      route:
        - destination:
            host: custom-app-svc.default.svc.cluster.local
            port:
              number: 80

    - match:
        - uri:
            prefix: /
      directResponse:
        status: 404
        body:
          string: "Not Found"
```

PROF

deploy.sh (отредактированный)

```
#!/bin/bash
set -eo pipefail

# Initialize Minikube and install Istio
```

```
echo "Starting Minikube and setting up Istio..."
minikube start --driver=docker
eval $(minikube docker-env)

# Install Istio
ISTIO_VERSION=1.18.0
echo "Downloading Istio ${ISTIO_VERSION}..."
curl -sL
https://github.com/istio/istio/releases/download/${ISTIO_VERSION}/istio-
${ISTIO_VERSION}-linux-amd64.tar.gz | tar xz
export PATH="${PWD}/istio-${ISTIO_VERSION}/bin:$PATH"
istioctl install --set profile=demo -y
minikube kubectl -- label namespace default istio-injection=enabled

# Build app image
echo "Building Docker image..."
docker build -t custom-app . || { echo "Docker build failed"; exit 1; }

# Create host directories
minikube ssh "sudo mkdir -p /var/log/app /var/log/app-archives && sudo
chmod 777 /var/log/app /var/log/app-archives"

# Apply Kubernetes manifests
declare -a manifests=(
    "k8s/configmap.yaml"
    "k8s/deployment.yaml"
    "k8s/service.yaml"
    "k8s/daemonset.yaml"
    "k8s/cronjob.yaml"
    "k8s/istio-gateway.yaml"
    "k8s/istio-virtualservice.yaml"
    "k8s/destination-rule.yaml"
)

echo "Applying manifests..."
for manifest in "${manifests[@]}"; do
    if [ ! -f "$manifest" ]; then
        echo "Error: Missing $manifest"
        exit 1
    fi
    minikube kubectl -- apply -f "$manifest"
done

# Wait for components
echo "Waiting for deployment rollout..."
minikube kubectl -- rollout status deployment/custom-app --timeout=180s

echo "Checking DaemonSet..."
minikube kubectl -- rollout status daemonset/log-agent --timeout=120s

echo -e "\nDeployment complete!"
echo -e "Access endpoints through Istio Gateway:"
echo -e "export INGRESS_PORT=\$(kubectl -n istio-system get service
```

```
istio-ingressgateway -o jsonpath='{.spec.ports[?
(@.name==\"http2\")].nodePort}')"
echo -e "export INGRESS_HOST=\$(minikube ip)"
echo -e "curl http://\$INGRESS_HOST:\$INGRESS_PORT"
echo -e "\nView agent logs with:"
echo -e "kubectl logs -l app=log-agent"
echo -e "\nView archives in Minikube:"
echo -e "minikube ssh 'ls -lh /var/log/app-archives'"
```

app.py (отредактированный)

```
from flask import Flask, request, jsonify
import os
import logging
from datetime import datetime
import time

app = Flask(__name__)

# Configuration from environment variables
welcome_message = os.environ.get('WELCOME_MSG', 'Welcome to the custom
app')
log_level = os.environ.get('LOG_LEVEL', 'INFO')
port = int(os.environ.get('PORT', 5000))

# Configure application logging
logging.basicConfig(
    level=log_level,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/app.log'),
        logging.StreamHandler()
    ]
)

# Ensure logs directory exists
logs_dir = '/app/logs'
os.makedirs(logs_dir, exist_ok=True)

@app.route('/')
def home():
    return welcome_message

@app.route('/status')
def status():
    return jsonify({"status": "ok"})

@app.route('/log', methods=['POST'])
def log_message():
    data = request.get_json()
    if not data or 'message' not in data:
```

```
        return jsonify({"error": "Invalid data"}), 400

    try:
        time.sleep(3)  # Имитация долгой обработки
        logging.info(data['message'])
        return jsonify({"status": "logged"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route('/logs')
def get_logs():
    try:
        with open('/app/logs/app.log', 'r') as f:
            logs = f.read()
        return logs.replace('\n', '<br>')  # Сохраняем форматирование
для браузера
    except FileNotFoundError:
        return jsonify({"error": "Logs not found"}), 404

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=port)
```