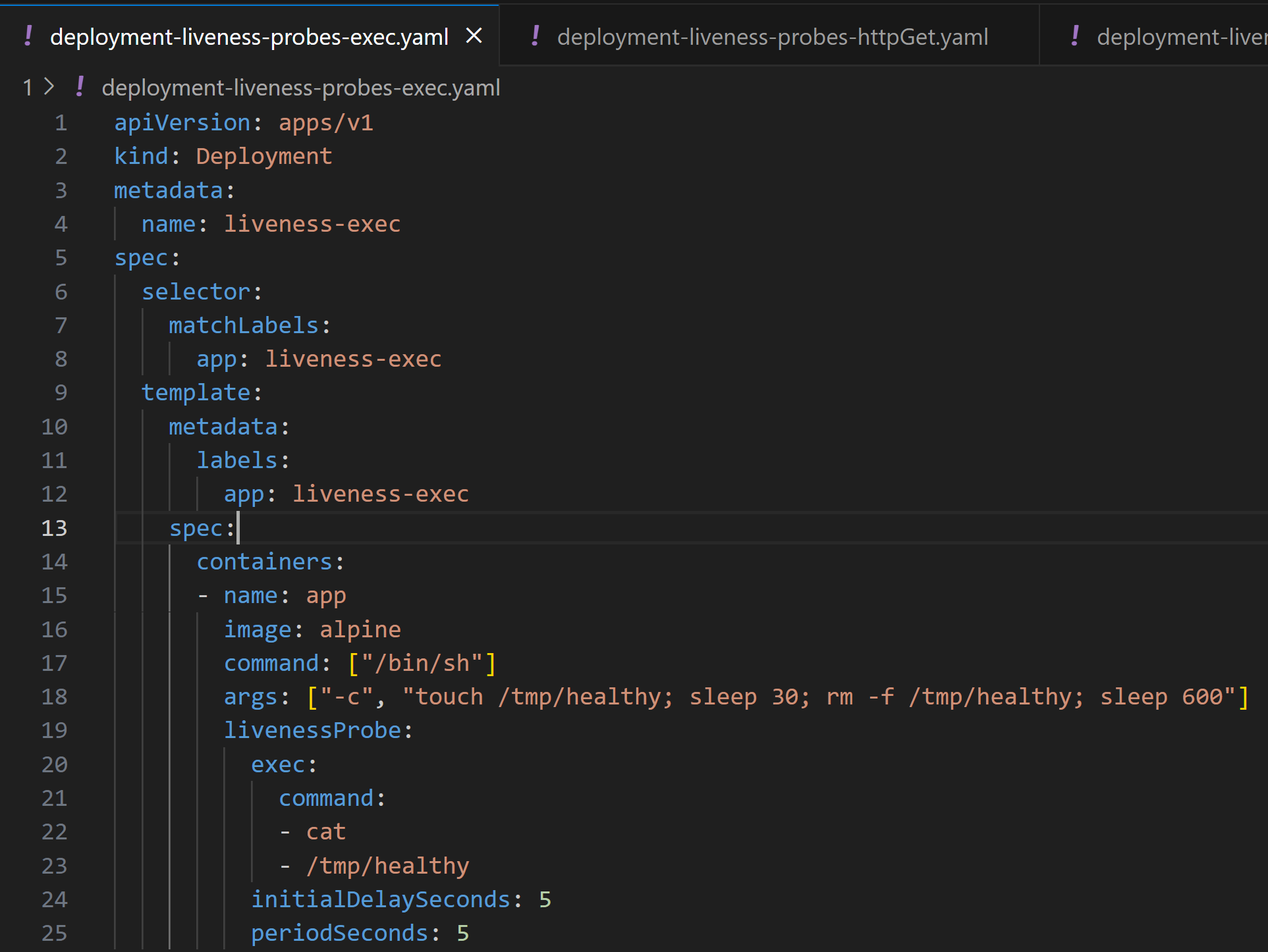
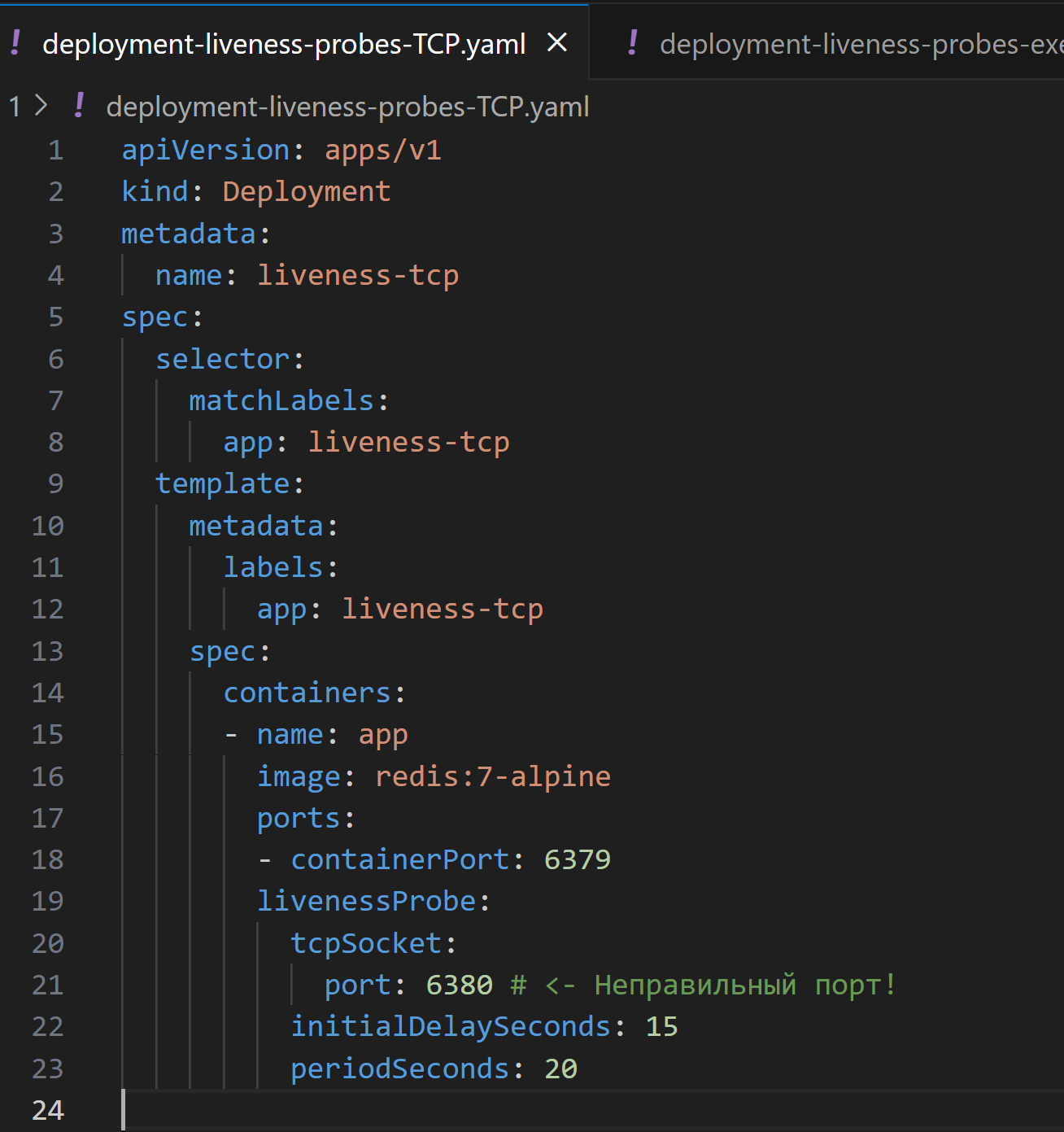
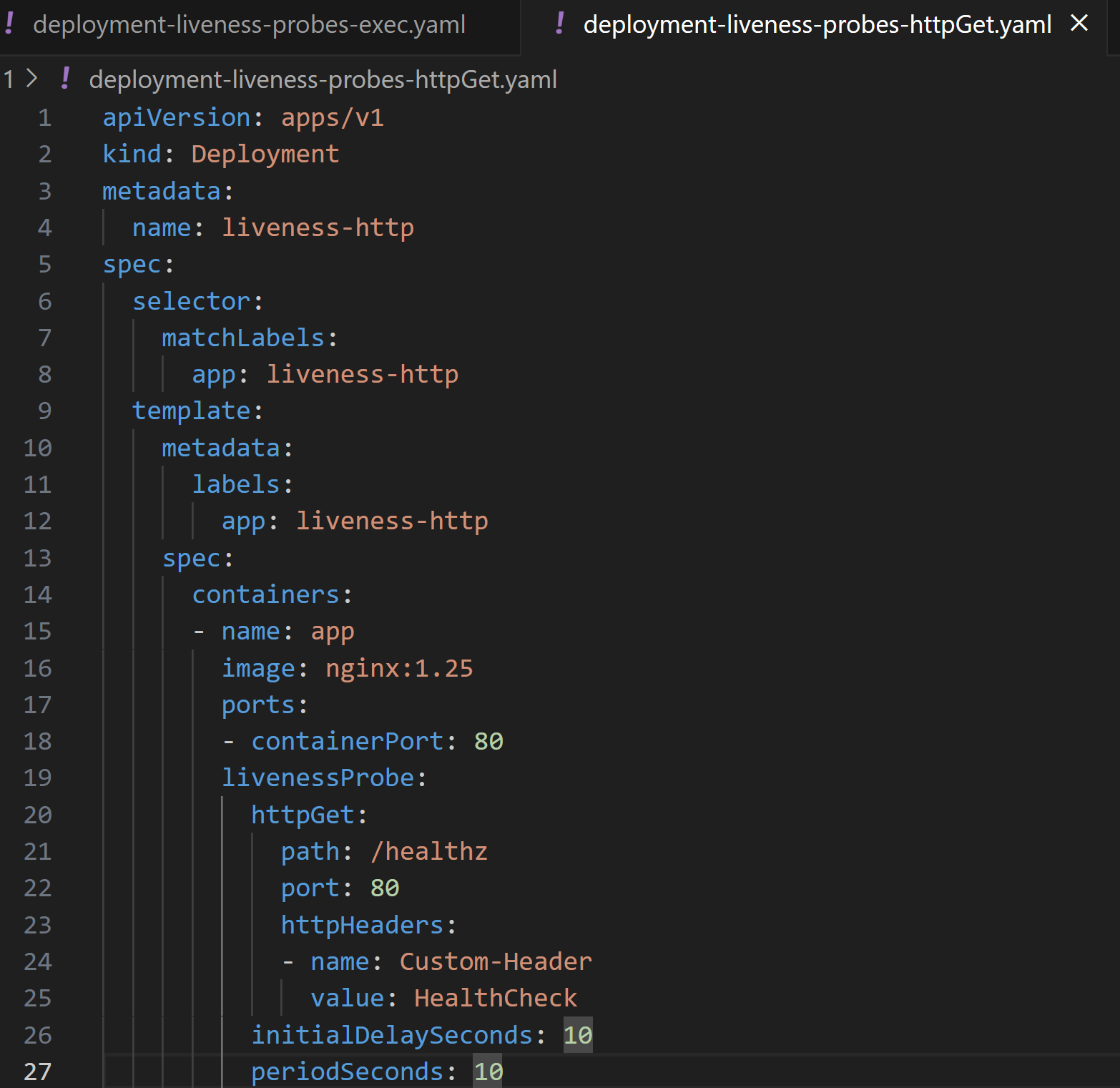
**Задание 6.1**

Созданы необходимые Deployment:







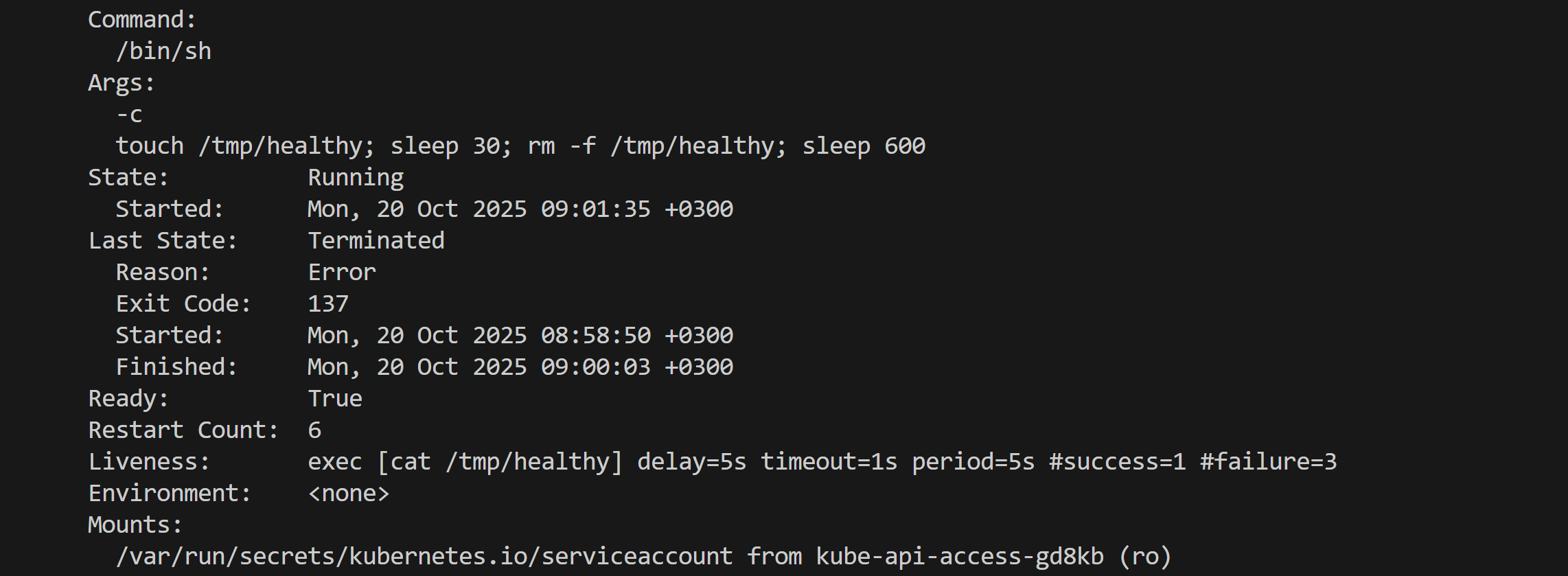
**Проверочные команды**

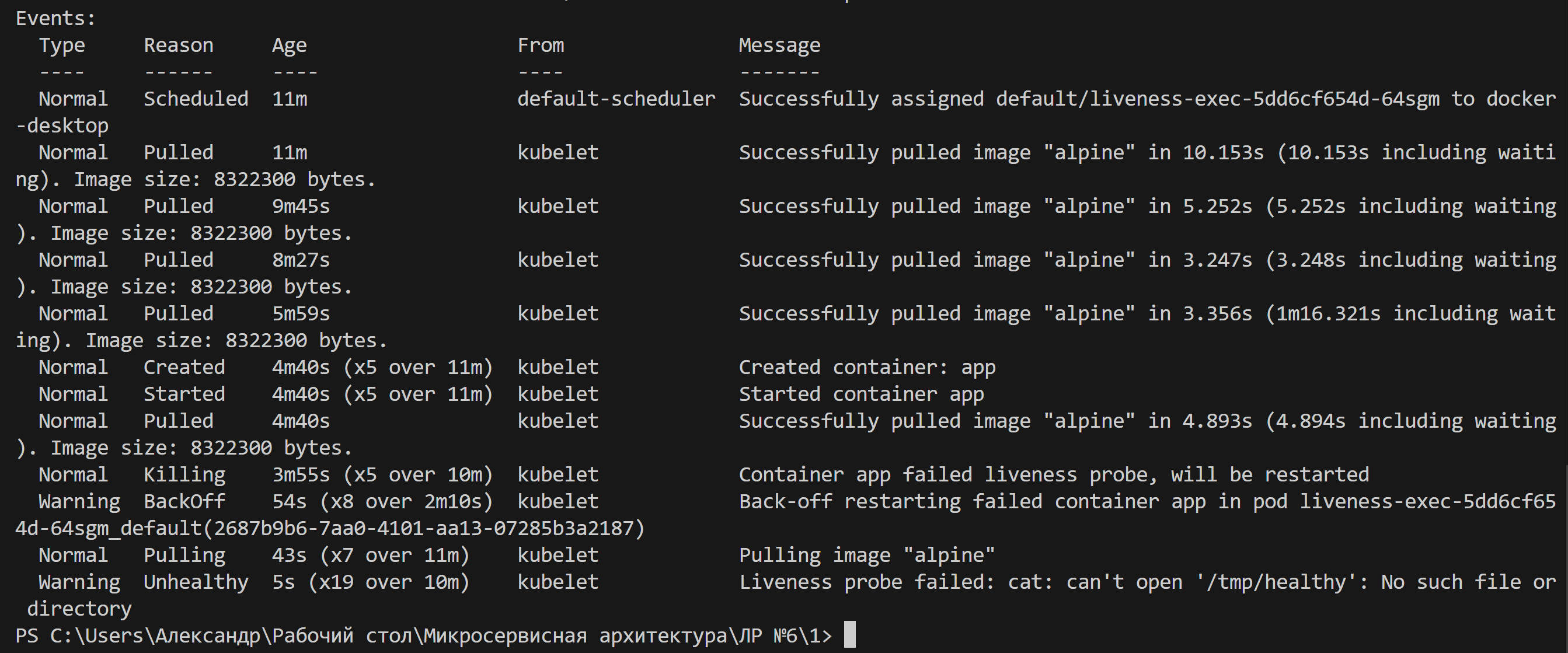
1. Общий мониторинг состояния подов: **kubectl get pods -o wide**



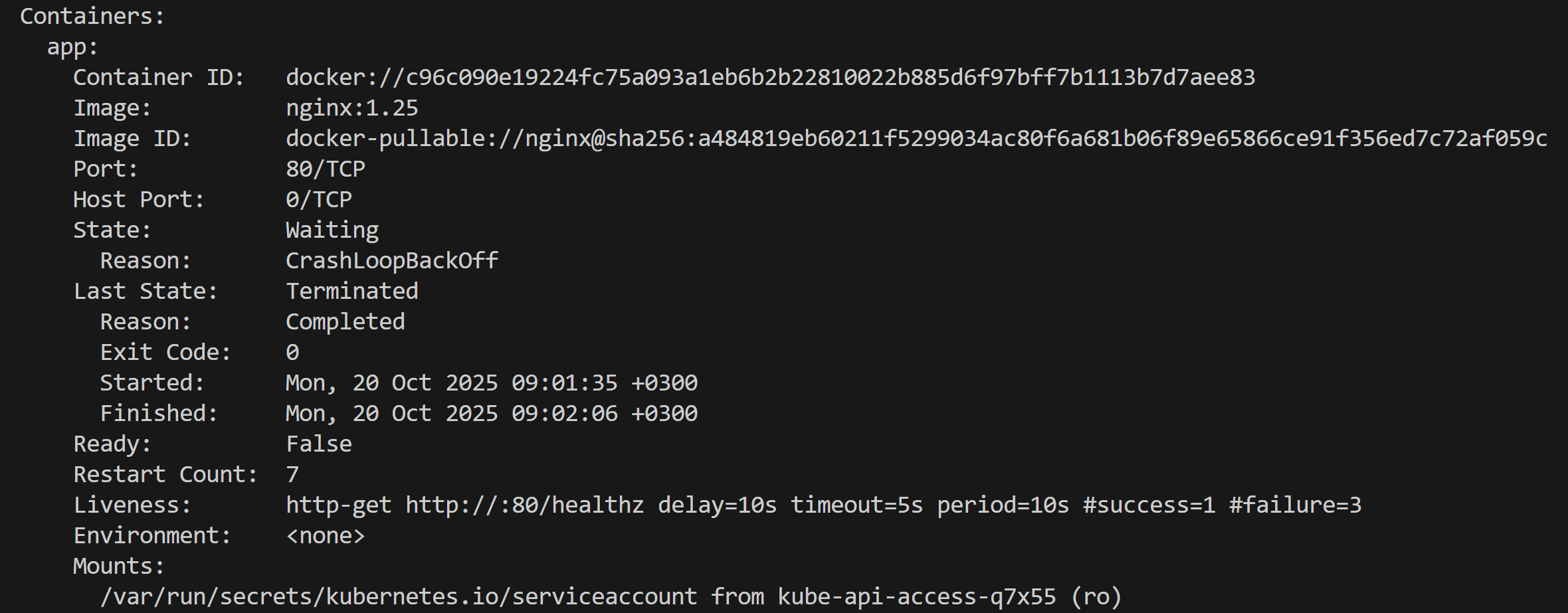
2. Детальная информация о пробах: **kubectl describe pod <имя пода>**

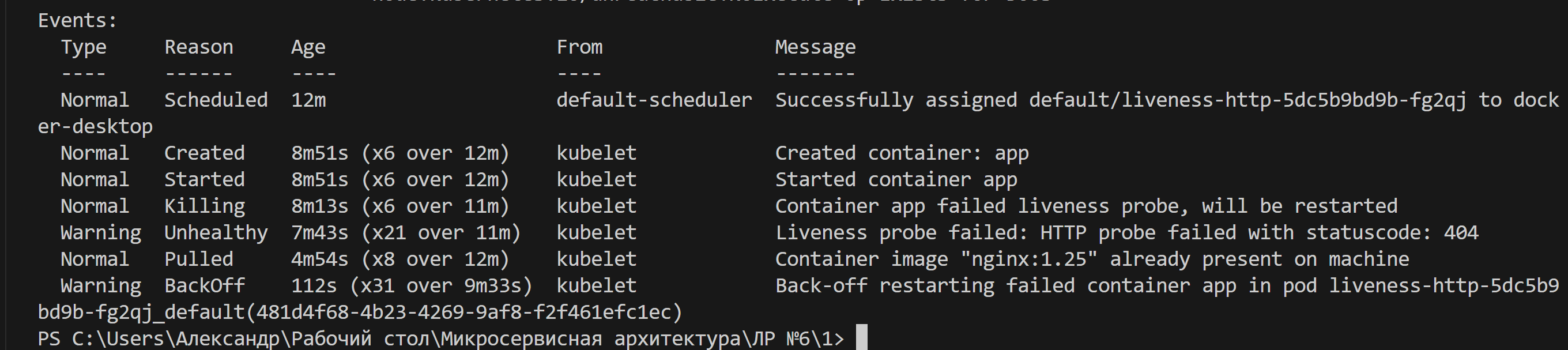
**Depoyment exec**



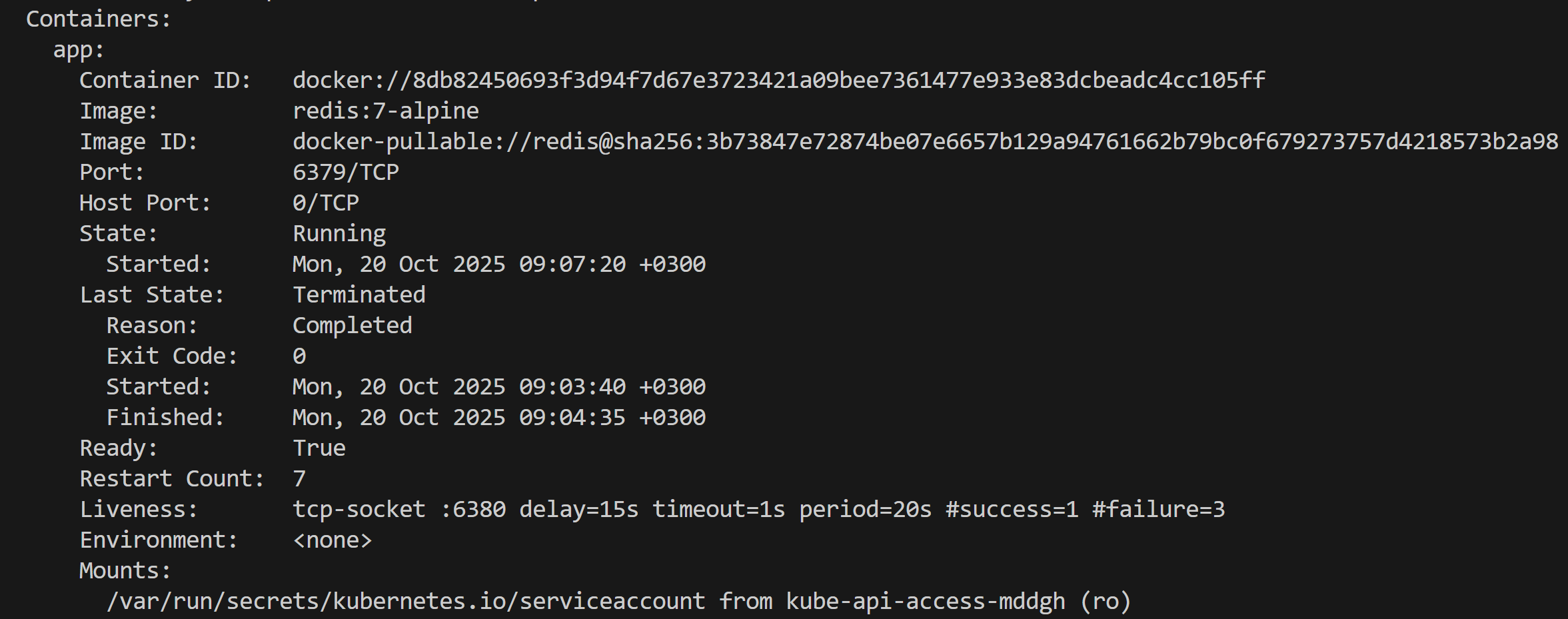


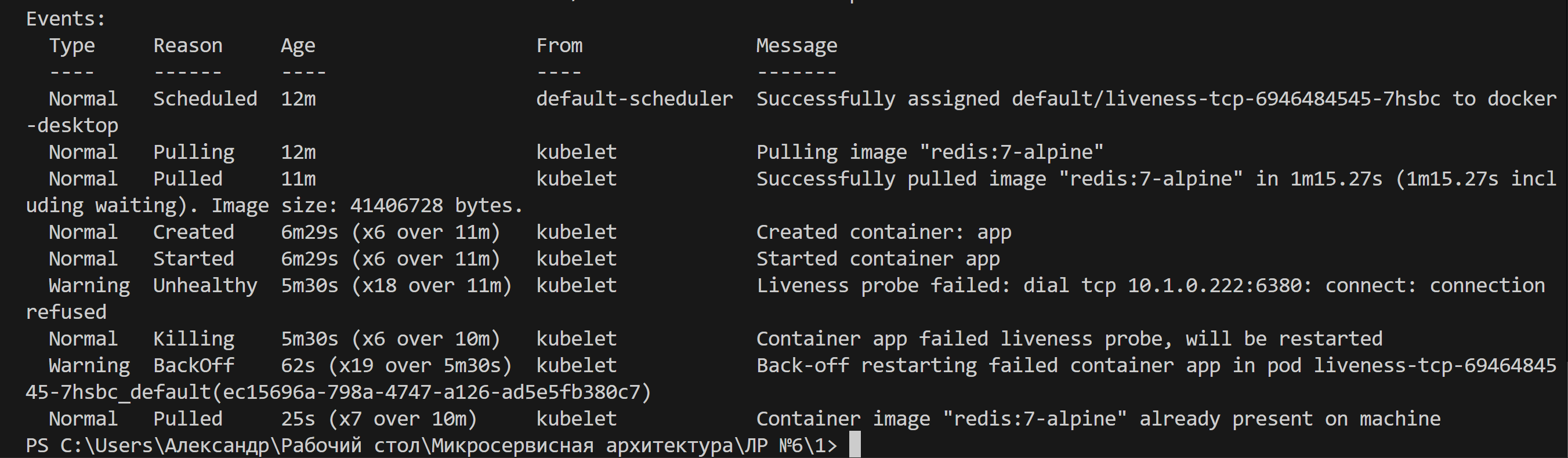
**Depoyment http**





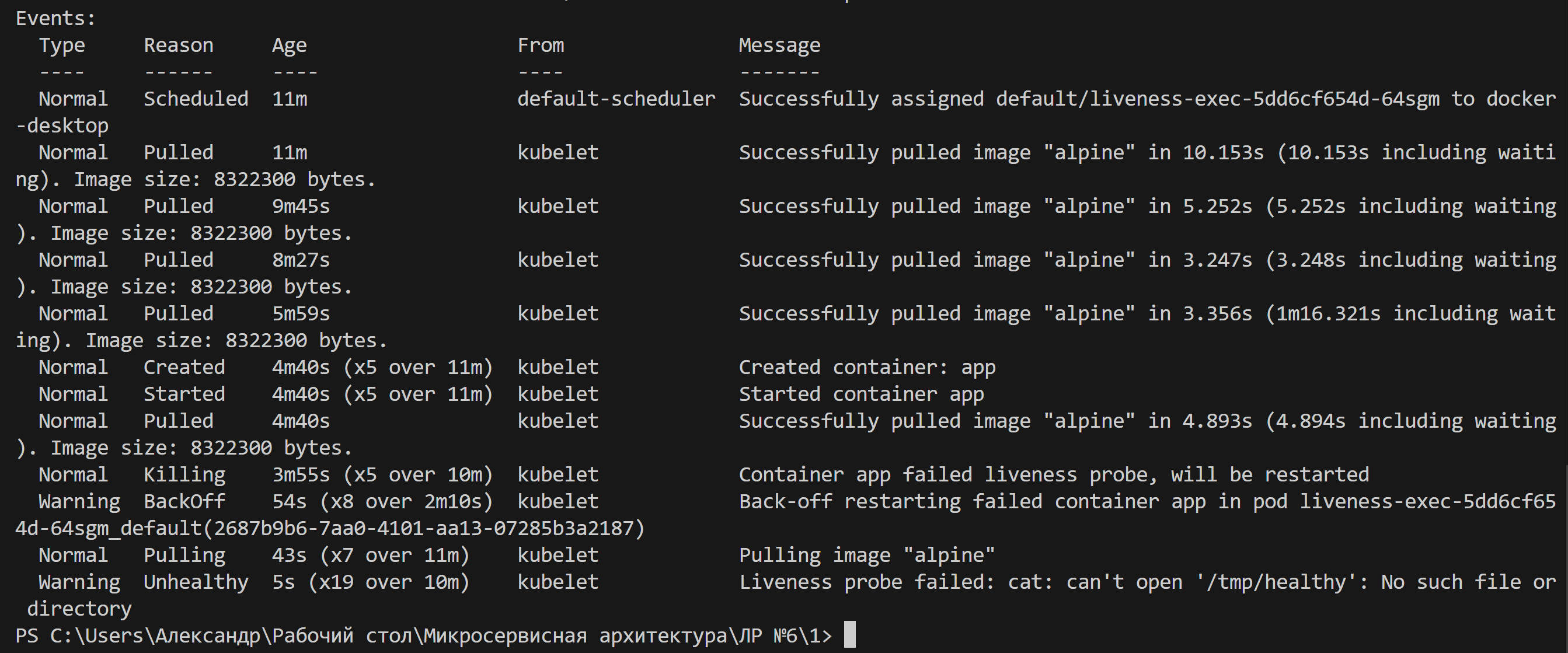
**Depoyment TCP**



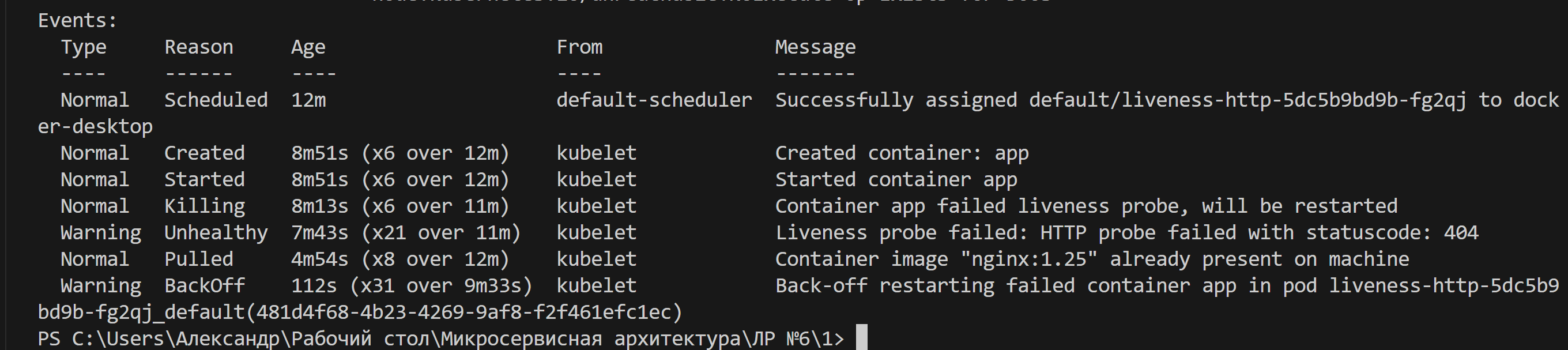


3. Проверка событий провала проб: **kubectl describe pod <имя пода> (секция events)**

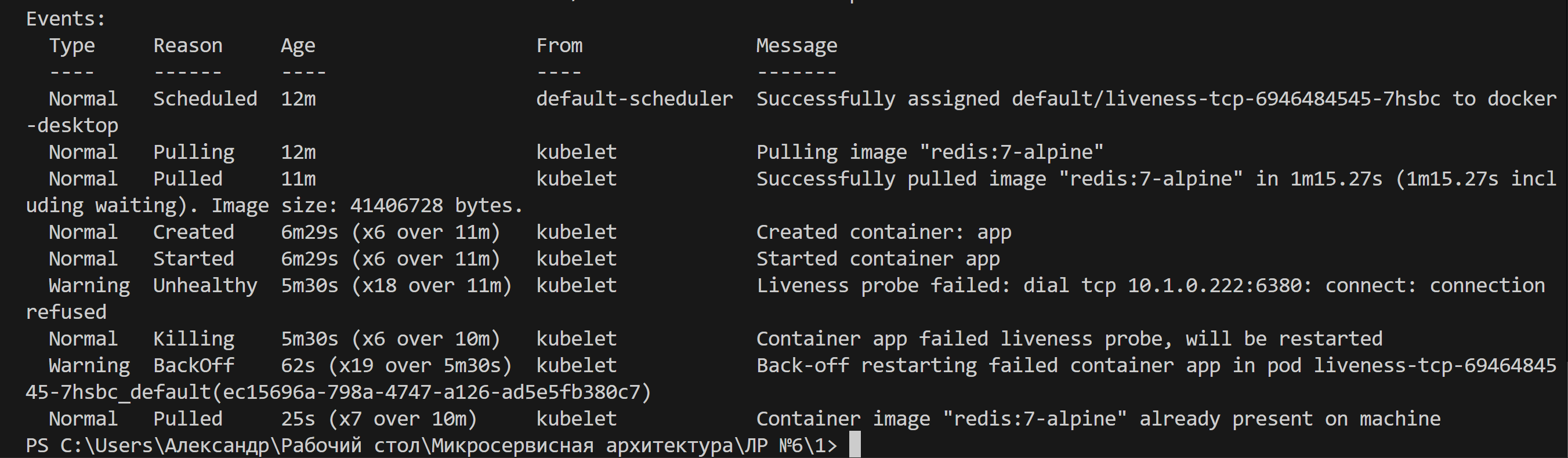
**Depoyment exec**



**Depoyment http**

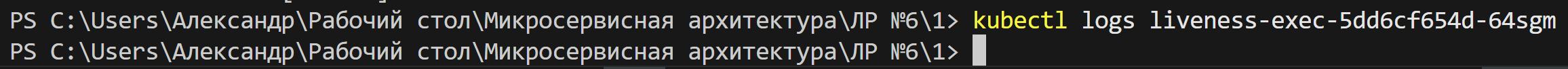


**Depoyment TCP**



4. Логи контейнера для анализа причин падения: **kubectl logs <имя пода>**

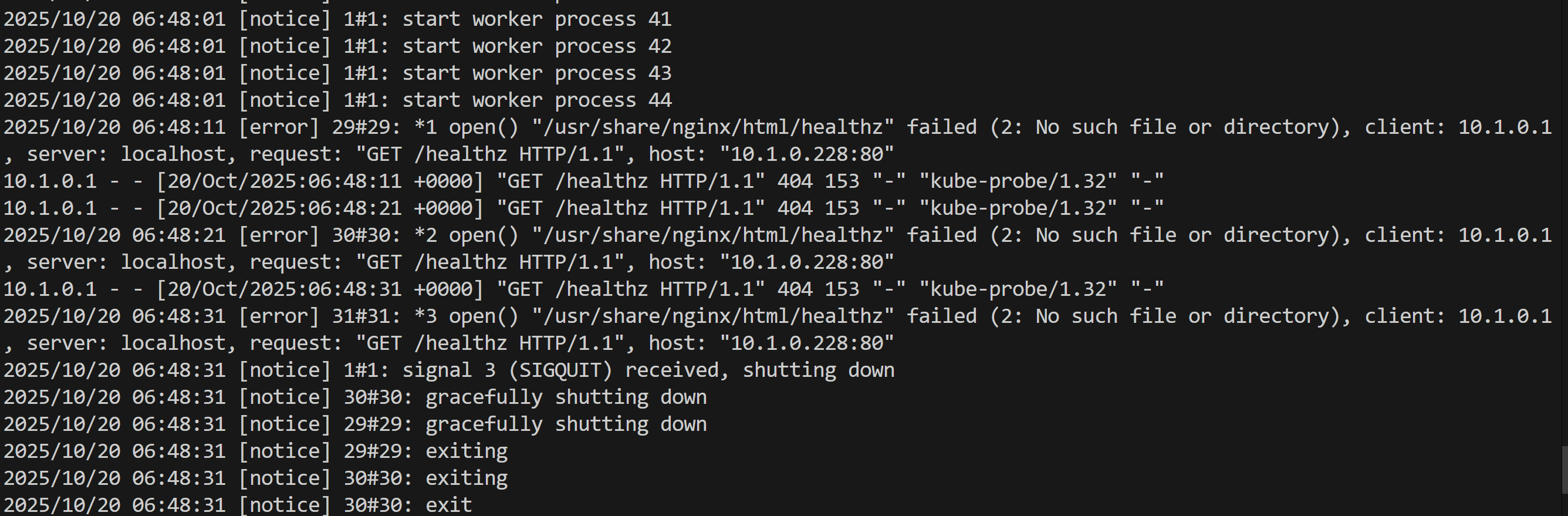
**Depoyment exec**

****

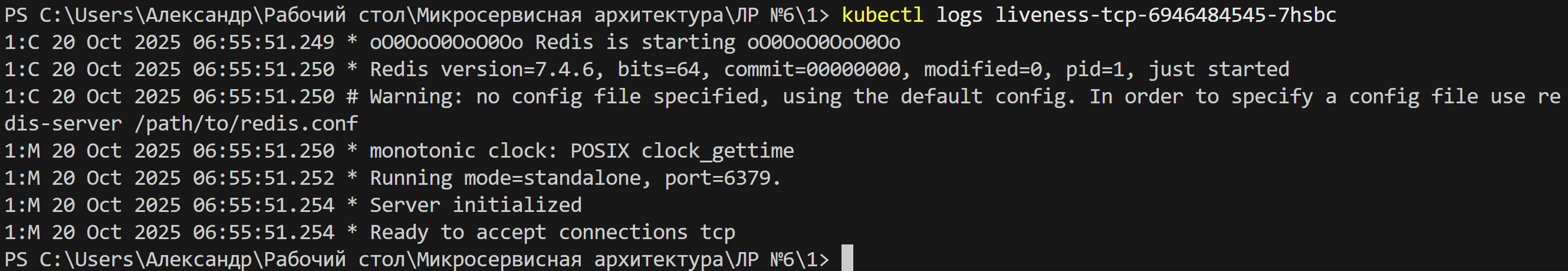
(контейнер не успел записать логи)

**Depoyment http**

Логи, которые показывают причину ошибки:

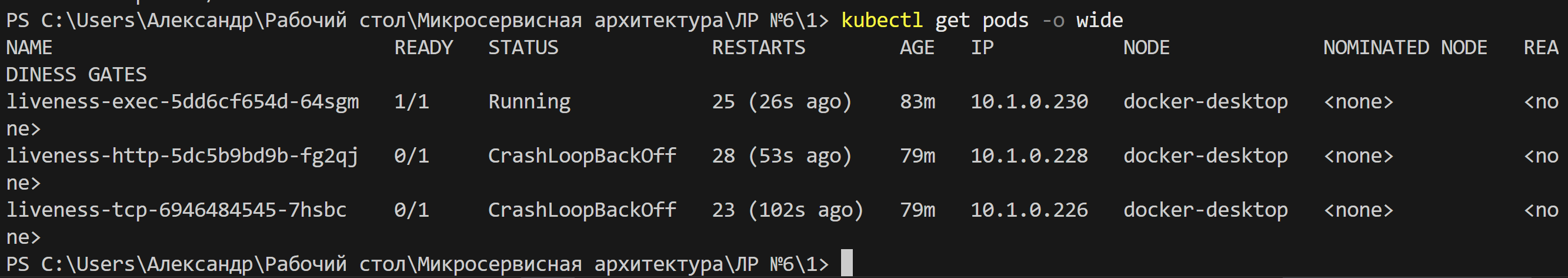


**Depoyment TCP**



(Проверка здоровья через TCP подключение к неправильному порту!)

5. Проверка истории перезапусков: **kubectl get pods -o wide** (столбец restarts)



**Задание 6.2**

Создадим необходимый deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

  name: readiness-probe

spec:

  selector:

    matchLabels:

      app: readiness-probe

  template:

    metadata:

      labels:

        app: readiness-probe

    spec:

      containers:

      - name: app

        image: nginx:1.25

        ports:

        - containerPort: 80

        command: ["/bin/sh", "-c"]

        args:

          - |

            sleep 40;

            nginx -g 'daemon off;'

        readinessProbe:

          httpGet:

            path: /

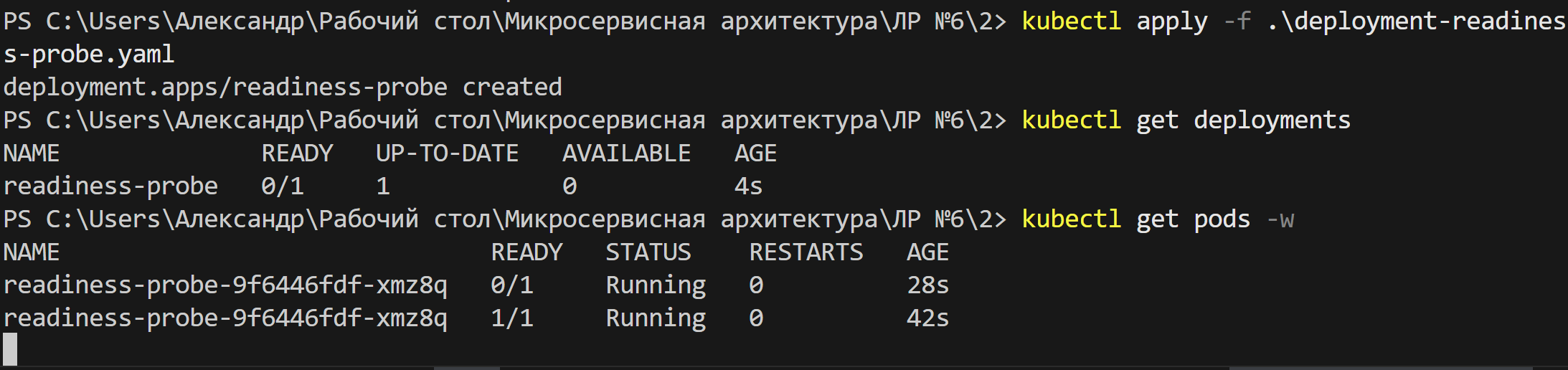
            port: 80

          initialDelaySeconds: 5

          periodSeconds: 5

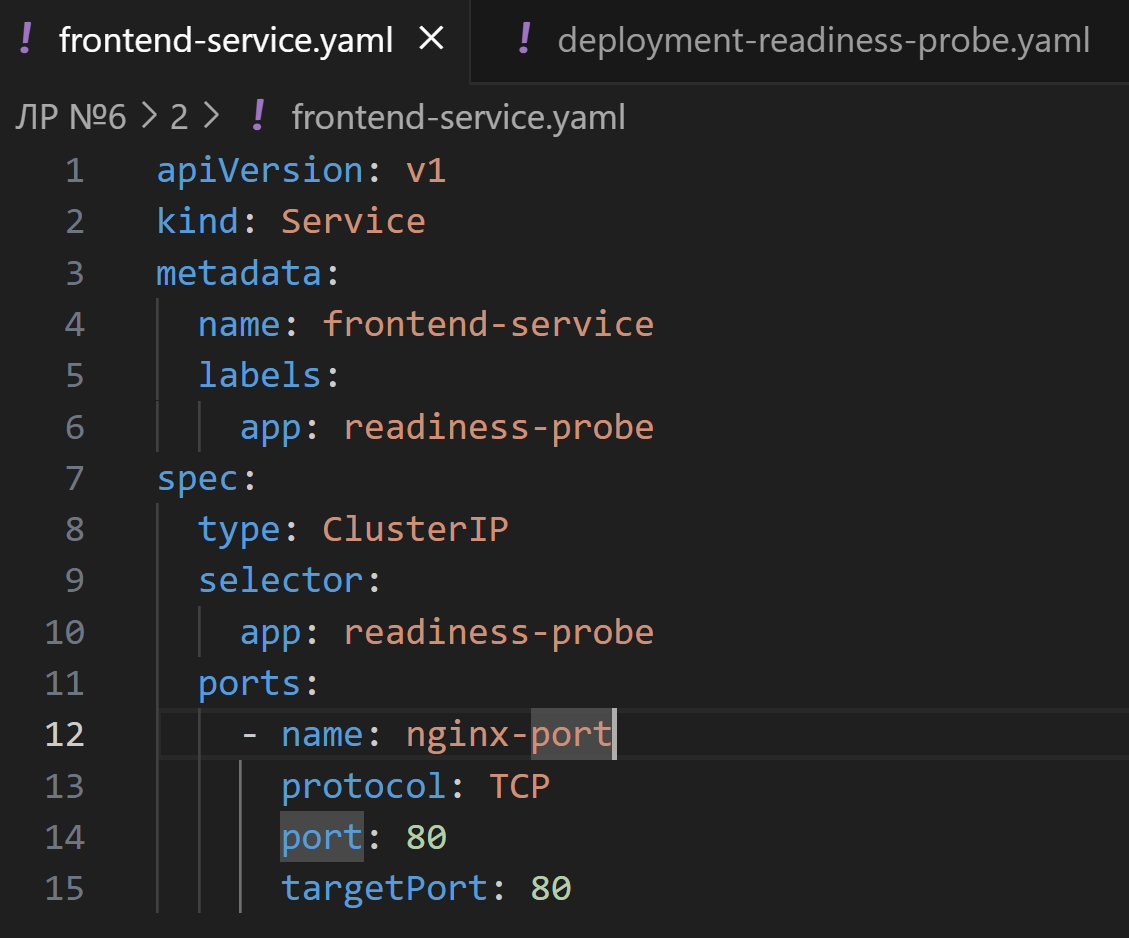
**Проверочные команды**

1. Наблюдение за изменением статуса Ready: **kubectl get pods -w**

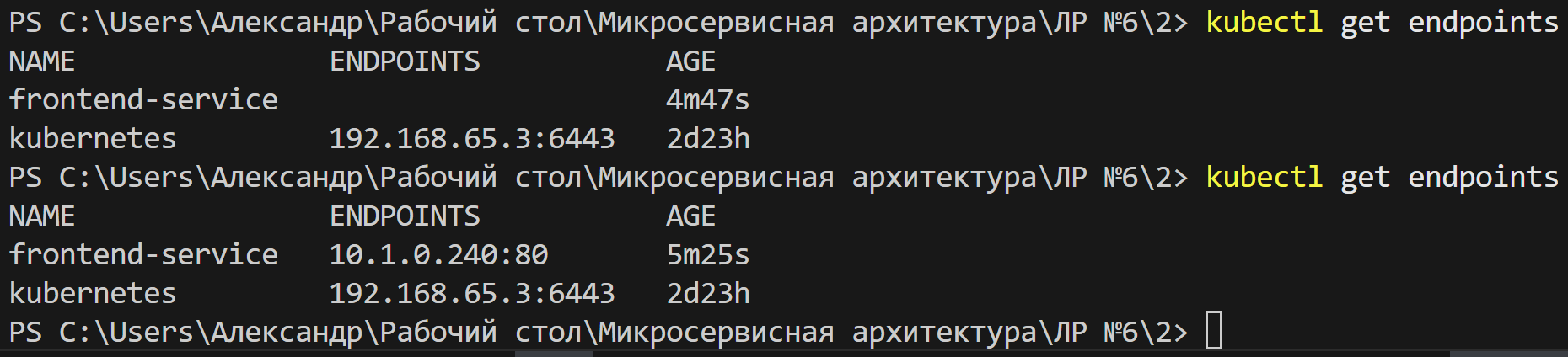


2. Проверка endpoints сервиса до и после готовности: **kubectl get endpoints**

Создадим вспомогательный service для просмотра endpoint:

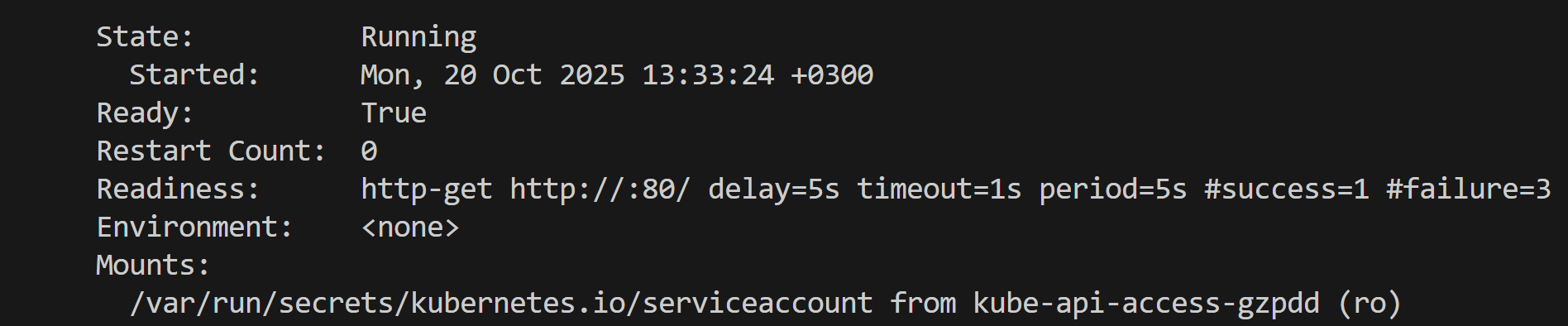


Проверка endpoints сервиса до и после готовности:

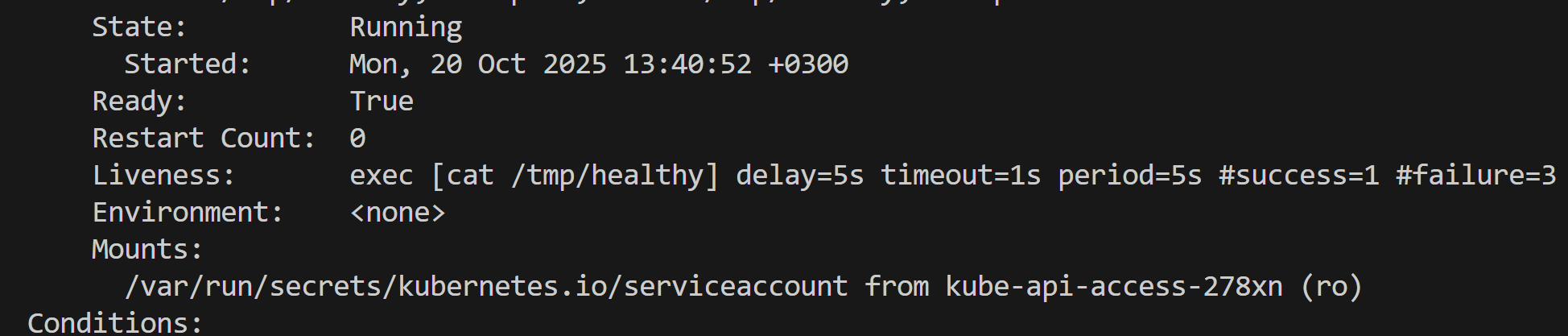


3. Сравнение Liveness и Readiness конфигурации

Readiness конфигурация:

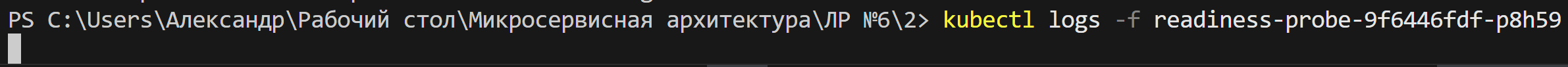


Liveness конфигурация:

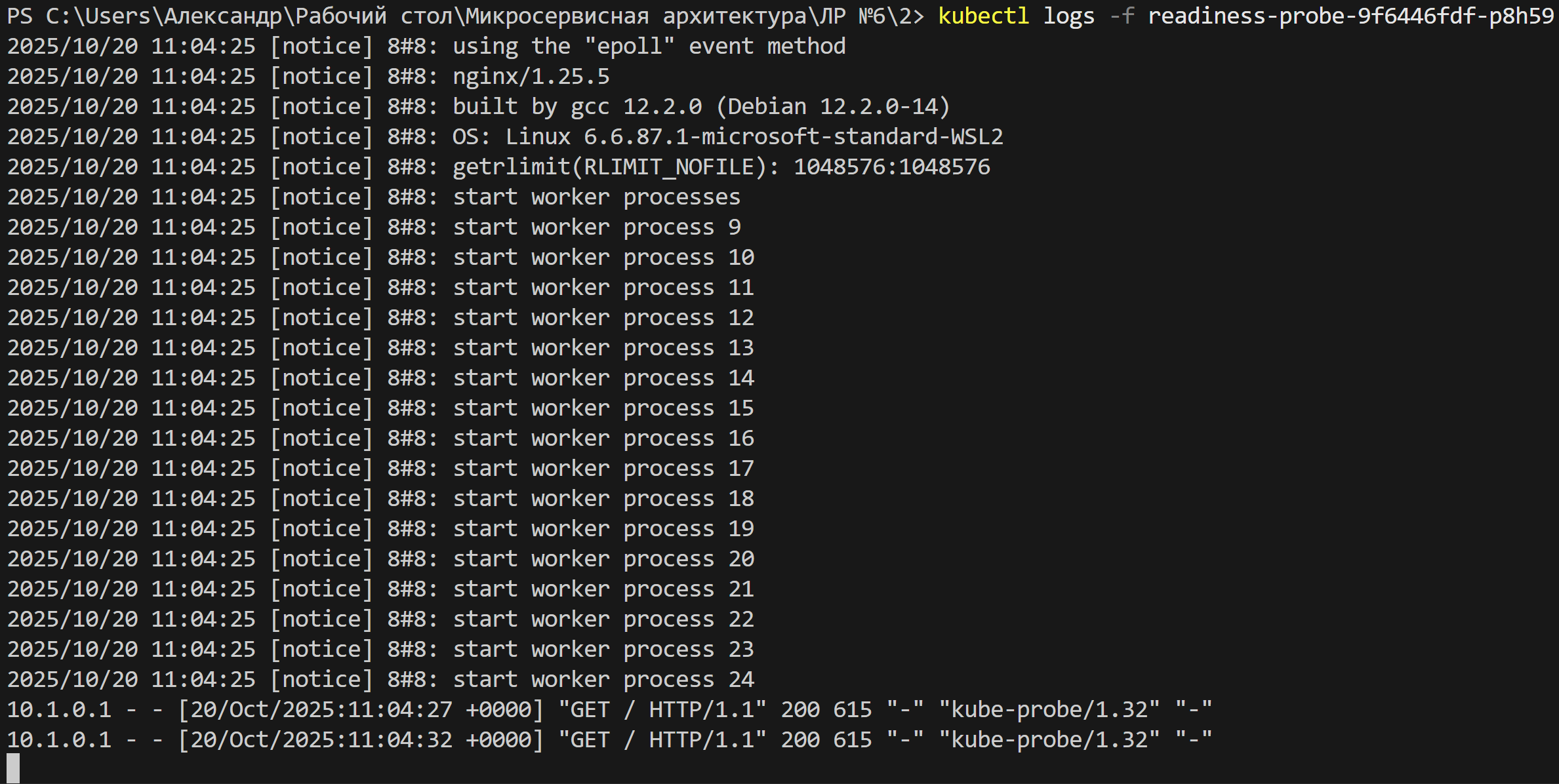


4. Логи инициализации приложения в реальном времени: **kubectl logs -f <имя пода>**

Вначале, из-за задержи никаких логов нет:

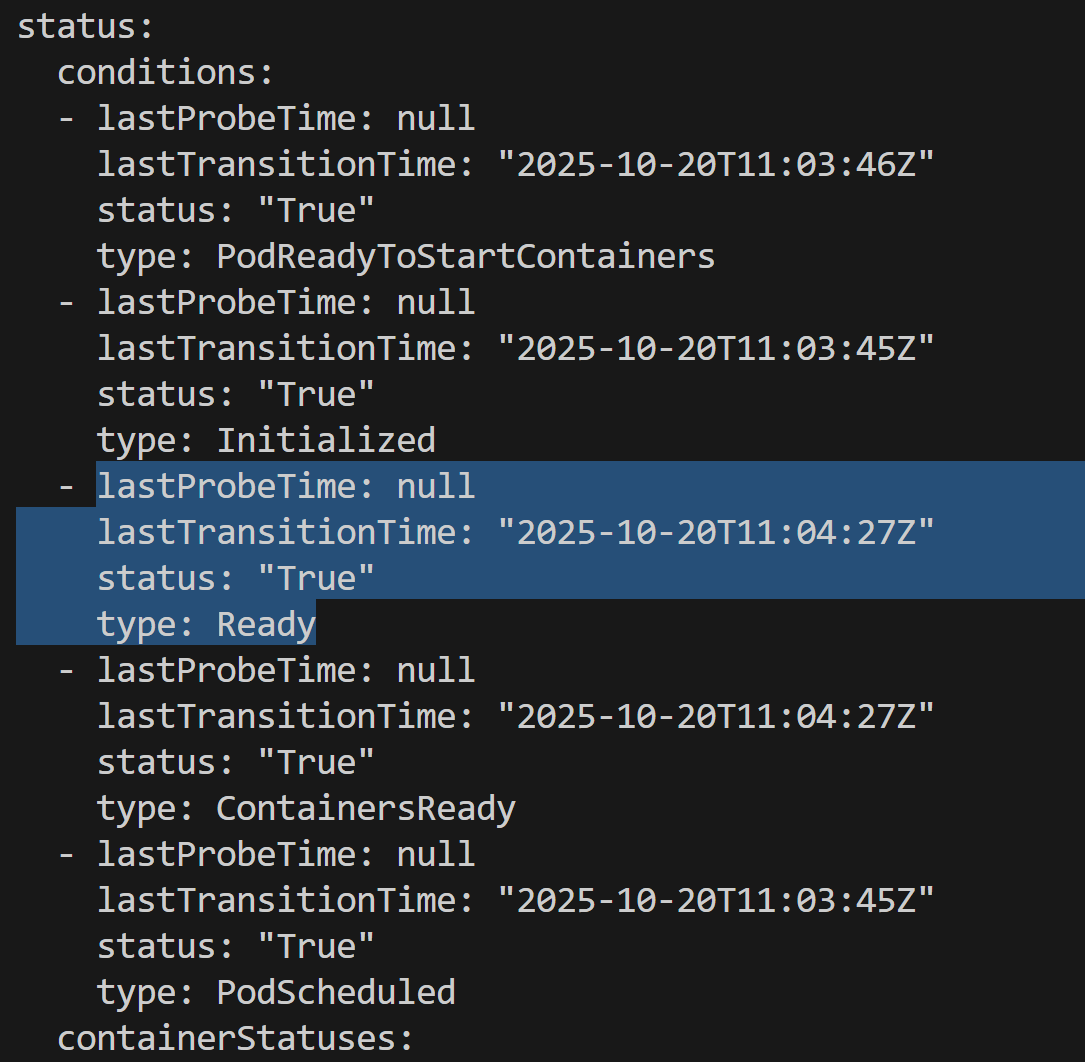


После установленной задержки появляются признаки инициализации:

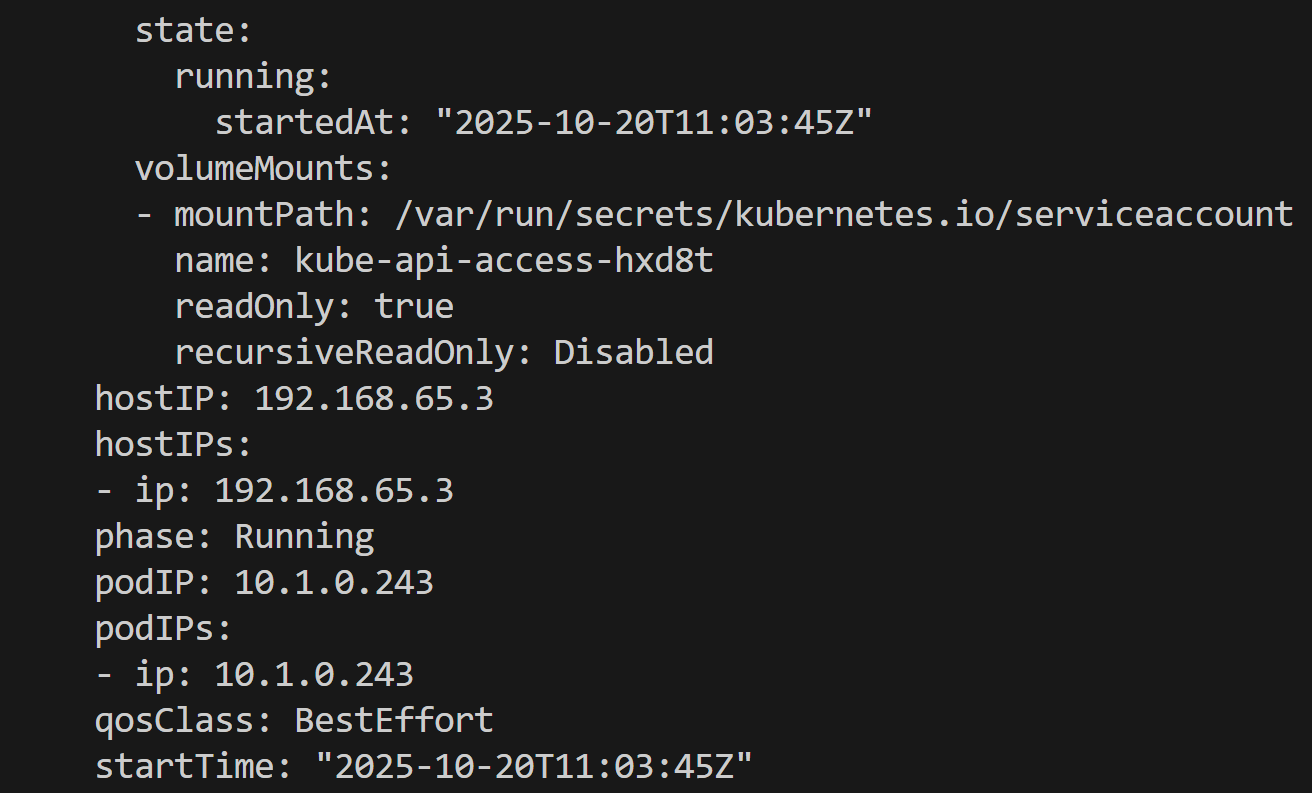


5. Время перехода в Ready состояние: **kubectl get pod <имя пода> -o yaml**

Время перехода в ready состояние:

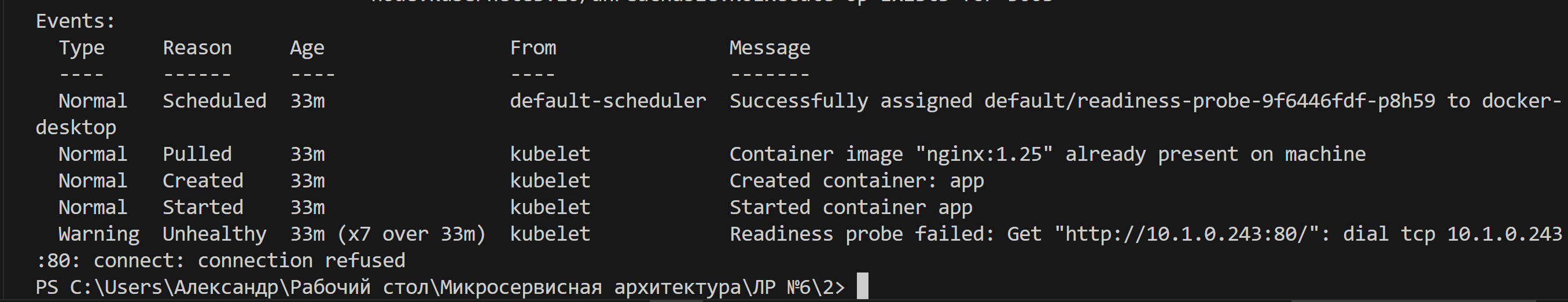


Время начала:



Итого время перехода составило 42 секунды.

6. Анализ временной шкалы событий запуска: **kubectl describe pod <имя пода>**



Планировщик назначил Pod, kubelet взял локальный образ, создал контейнер и запустил его. Затем Readiness probe не проходит как видно из Warning, а значит под не добавляется в endpoints и не получает трафик из-за искусственной задержи. Но после он начинает работать штатно и ошибок больше нет.

**Задание 6.3**

Создадим необходимый Deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

  name: volume-demo

spec:

  selector:

    matchLabels:

      app: volume-demo

  template:

    metadata:

      labels:

        app: volume-demo

    spec:

      containers:

      - name: writer

        image: alpine

        command: ["/bin/sh"]

        args: ["-c", "while true; do echo $(date) | tee -a /data/log.txt; sleep 5; done"]

        volumeMounts:

        - name: shared-data

          mountPath: /data

      - name: reader

        image: alpine

        command: ["/bin/sh"]

        args: ["-c", "tail -f /data/log.txt"]

        volumeMounts:

        - name: shared-data

          mountPath: /data

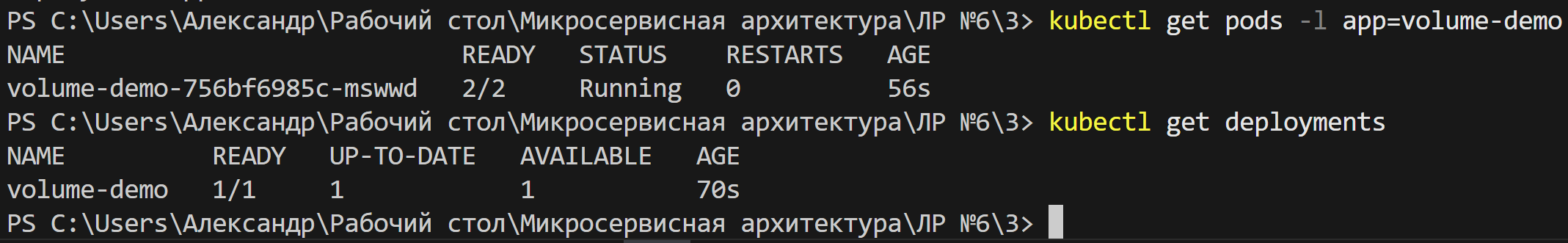
      volumes:

      - name: shared-data

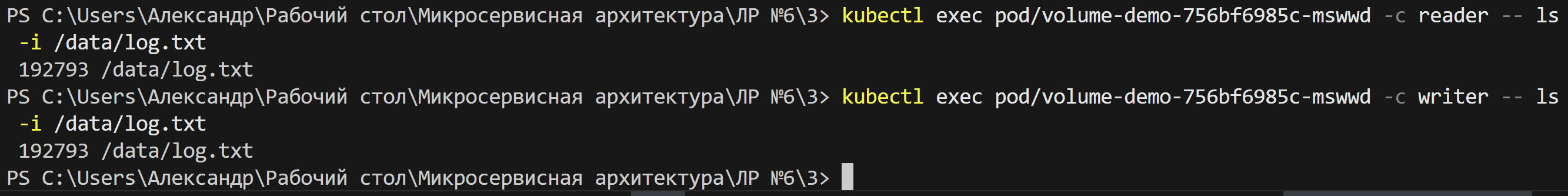
        emptyDir: {}

**Проверочные команды**

1. Проверка состояния многоконтейнерного пода

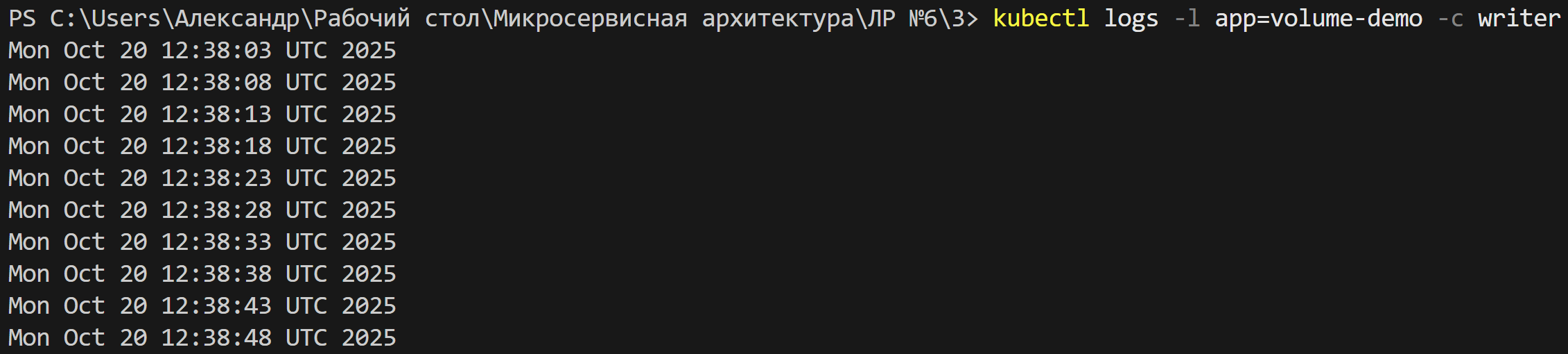


2, 3 Проверка монтирования volume в контейнерах и сравнение inode файлов в разных контейнерах: **kubectl exec pod/<имя пода> -c reader/writer -- ls -i /data/log.txt**

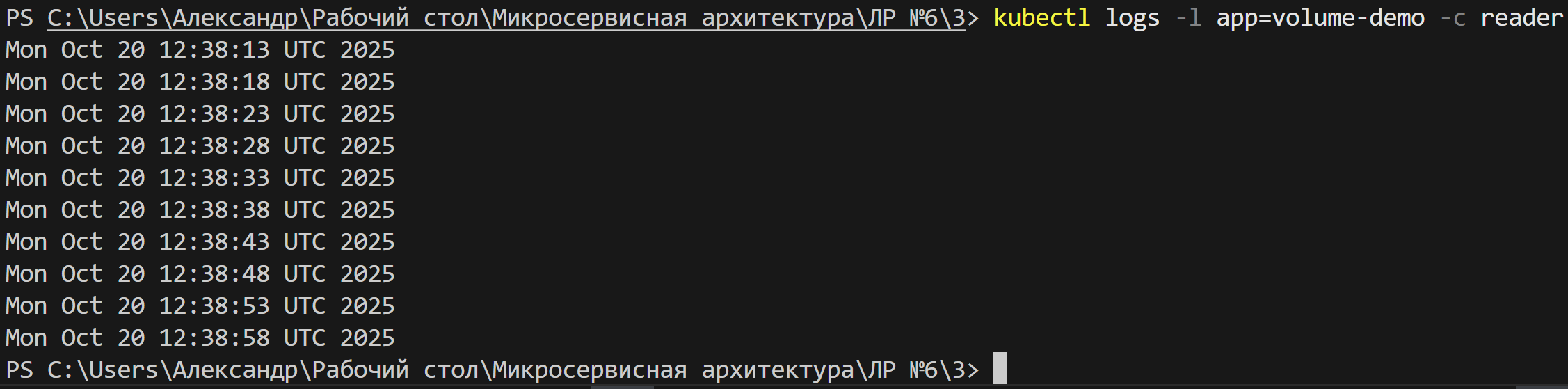


Volume корректно смонтирован и inode файлов совпадает: 192793.

4. Логи генератора данных: **kubectl logs -l app=volume-demo -c writer**



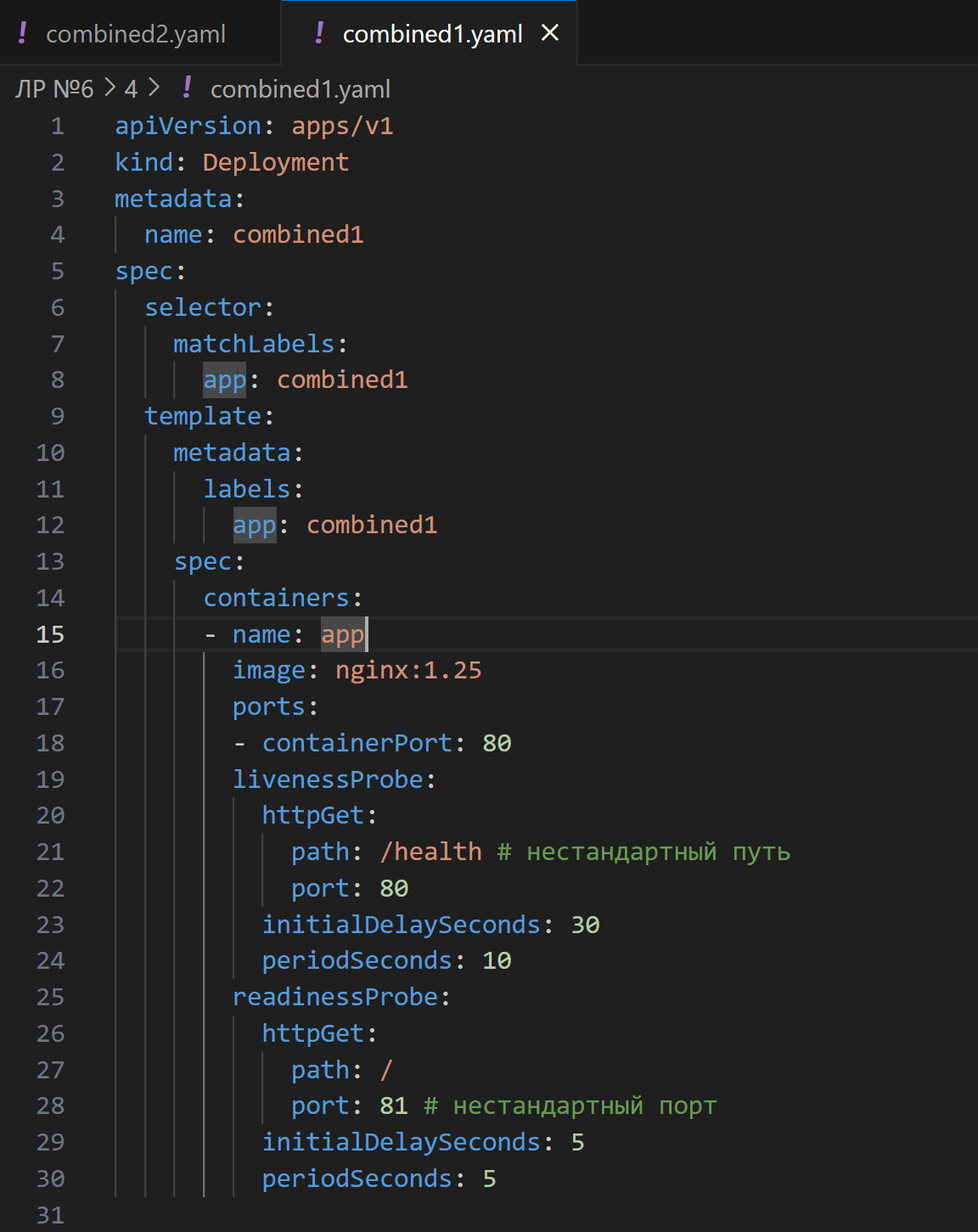
5. Логи читателя данных: **kubectl logs -l app=volume-demo -c reader**



Генератор и читатель пишут и читают данные соответственно каждые 5 секунд.

**Задание 6.4**

Создадим 2 проблемных deployment:

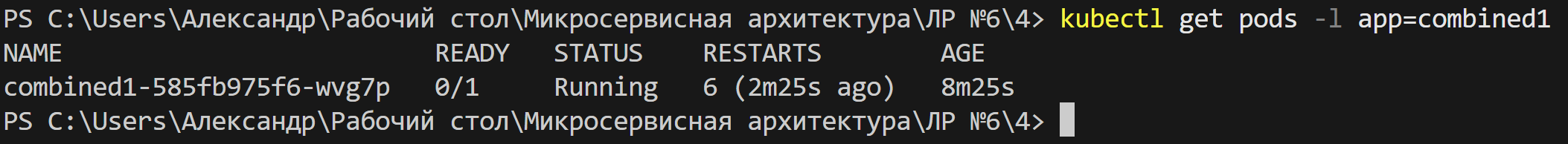




**Проверочные команды**

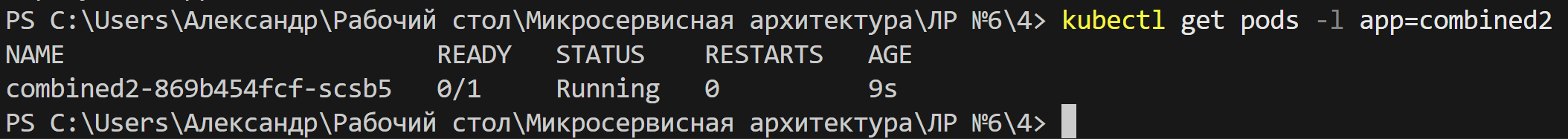
1. Комплексная диагностика проблемного пода: **kubectl get pods -l app=<метка app>**

Для сombined1.yaml:



С подом что-то не так.

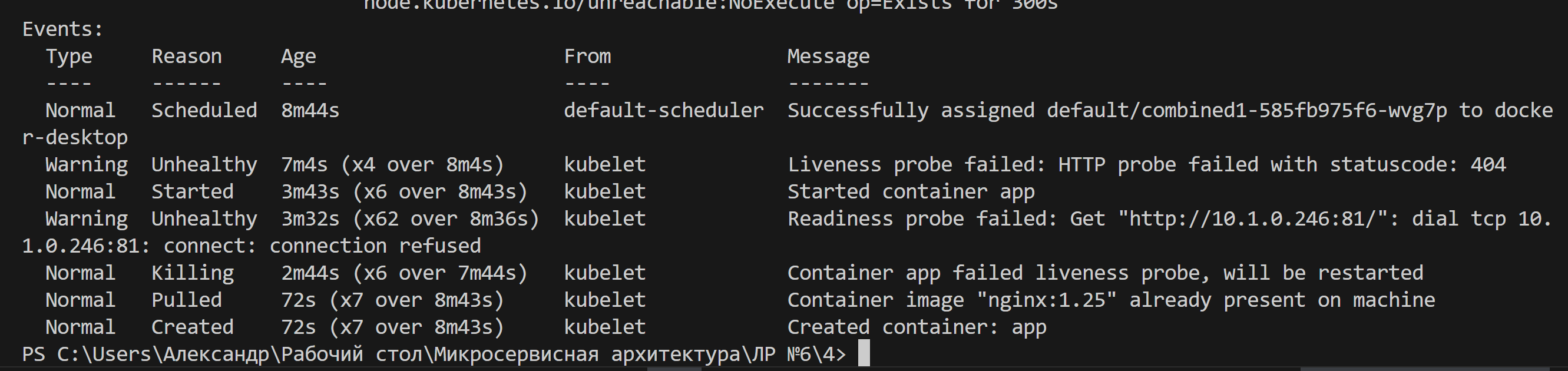
Для сombined2.yaml:



С этим подом тоже что-то не так.

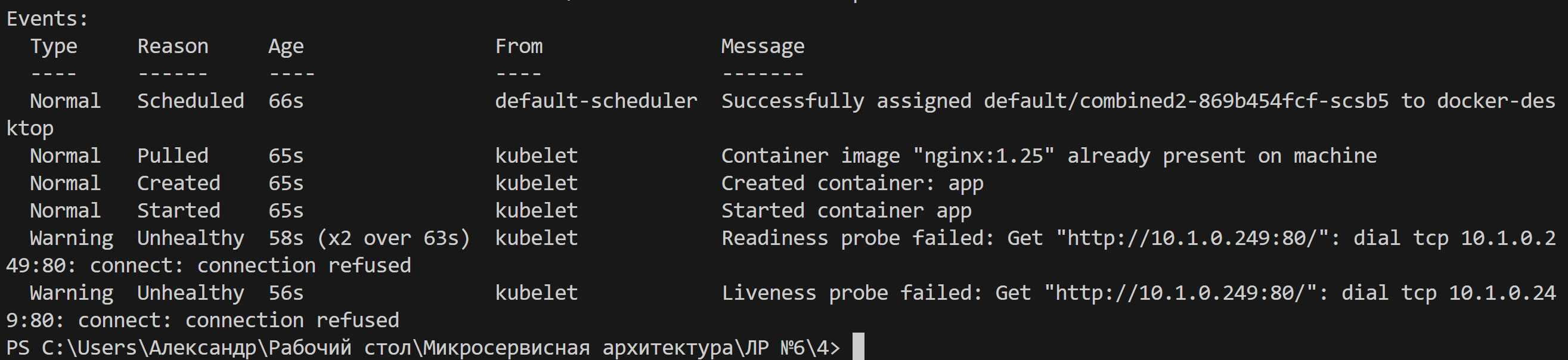
2. Детальный анализ событий: **kubectl describe pod <имя пода>**

Для сombined1.yaml:



Проба закончилась с ошибкой на порт 81 (выше было видно, что контейнер работает на 80 порту)

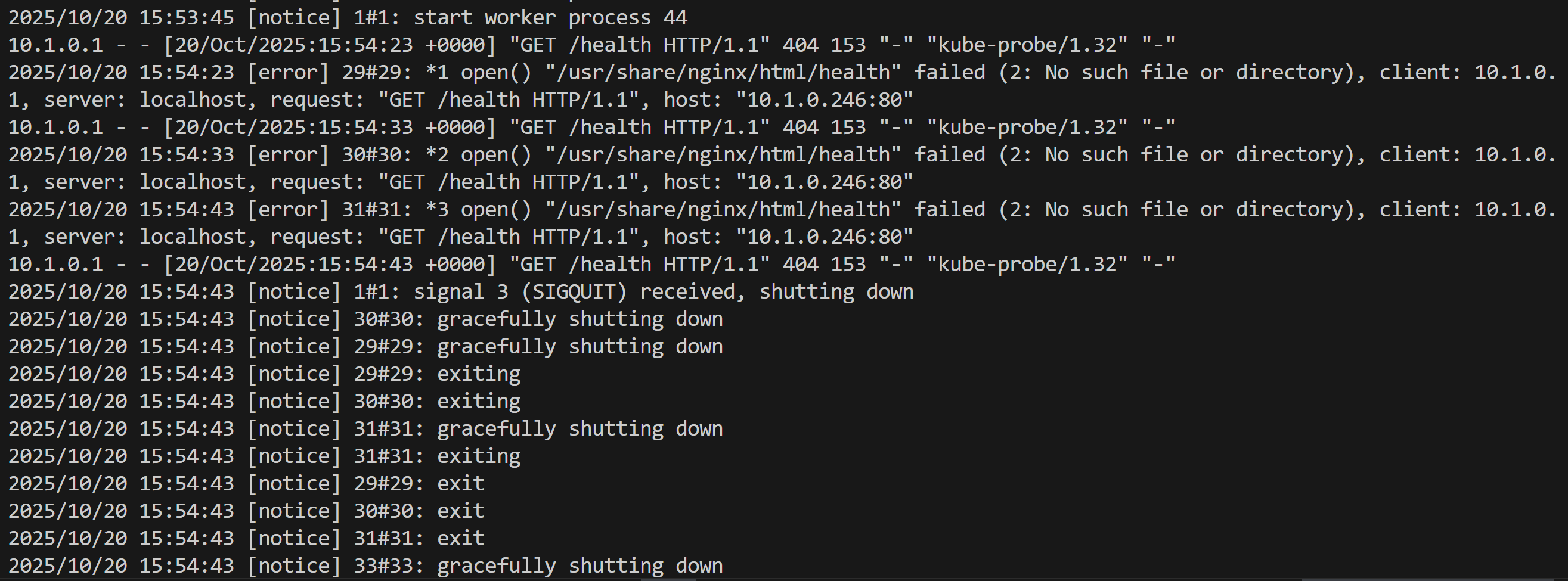
Для сombined2.yaml:



Здесь уже порт корректен, но проблемы всё равно имеются. Проверка пробами закончилась ошибками (так как контейнер просто не успел запуститься из-за sleep)

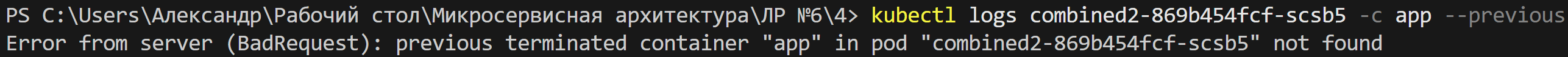
3. Логи предыдущего контейнера после перезапуска: **kubectl logs <имя пода> -c app --previous**

Для сombined1.yaml:



Попытка обратиться по пути /health заканчивается ошибкой 404.

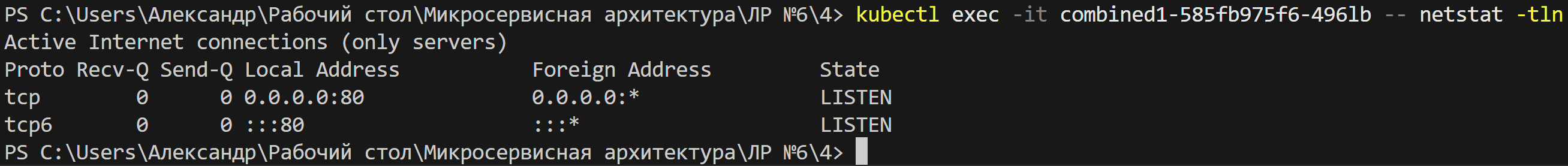
Для сombined2.yaml:



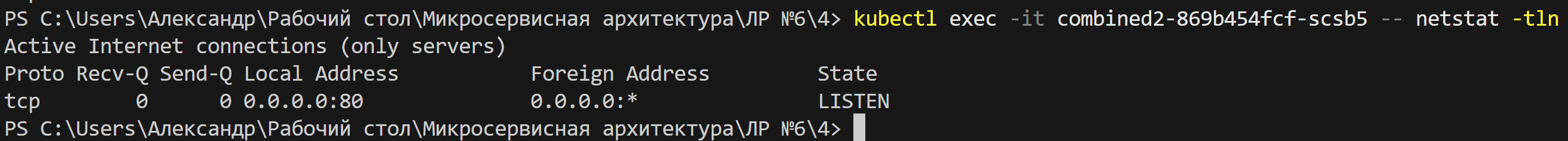
Предыдущий контейнер не найден, а значит он не уничтожался и не создавался вновь.

4. Проверка доступности портов внутри контейнера (может понадобиться предварительная установка в контейнер): **kubectl exec -it <имя пода> -- netstat -tln**

Для сombined1.yaml:



Для сombined2.yaml:

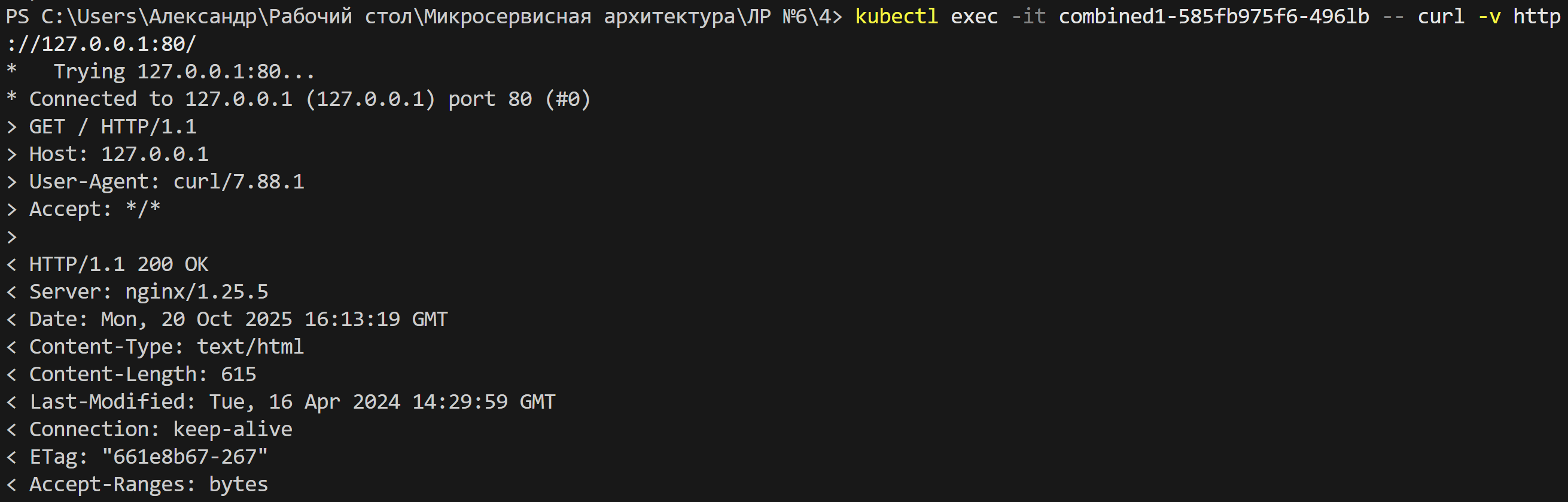


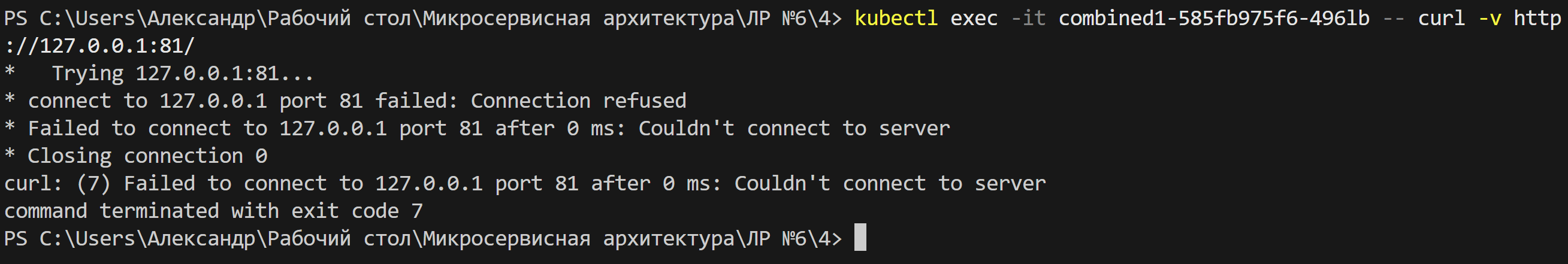
Всё корректно.

5. Тестирование endpoints изнутри контейнера: **kubectl exec -it <имя пода> -- curl -v http://127.0.0.1:<порт>/**

Для сombined1.yaml:

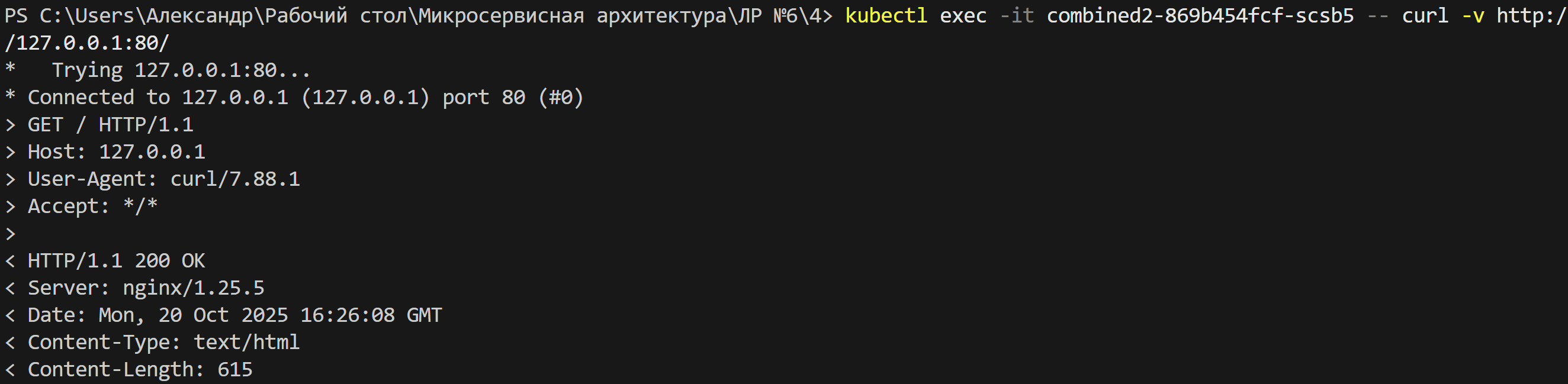
По 80 порту всё работает:

  
По 81 нет:



А значит, что ошибка просто в указании порта для probs.

Для сombined2.yaml:



Всё работает, а значит контейнер просто не сразу запустился, поэтому probs в events показали ошибки.

**Контрольные вопросы для всех заданий**

В чем основное отличие LivenessProbe от ReadinessProbe?

**Ответ:** Liveness проверяет, нужно ли перезапустить контейнер. Readiness проверяет, готов ли контейнер принимать трафик.

Какие типы проверок поддерживают probes и когда каждый из них следует использовать?

**Ответ:** exec выполняет команду в контейнере, можно использовать для внутренних проверок файлов или процессов. httpGet делает HTTP GET запрос, можно использовать для веб-сервисов. tcpSocket делает проверку TCP-подключения, можно использовать для сервисов, слушающих какой-то порт.

Что произойдет, если не настроить ReadinessProbe для приложения с долгим стартом?

**Ответ:** УPod-а появится endpoint до его полной готовности, на него пойдёт трафик и запросы будут ошибаться.

Какой смысл параметров initialDelaySeconds, periodSeconds и timeoutSeconds?

**Ответ:**

initialDelaySeconds – кол-во сек от старта контейнера до запуска LivenessProbe

periodSeconds – кол-во секунд между между пробами

timeoutSeconds – кол-во секунд ожидания пробы

Что означает failureThreshold и successThreshold?

**Ответ:**

successThreshold – мин кол-во проверок, чтобы проба считалась неудачной

failureThreshold – сколько раз сделает команду, чтобы считать контейнер умершим и перезапустить

Что такое emptyDir и для каких сценариев он предназначен?

**Ответ:** Общее хранилище данных между контейнерами. Два и более контейнеров работают с одним файлом. В результате: обмен данными + сохранение при перезапусках контейнеров.

Какие три команды вы бы использовали для диагностики проблем с LivenessProbe?

**Ответ:**

1. **kubectl describe pod <имя pod>** - события и параметры pod-а
2. **kubectl logs < имя pod > -c <имя container> --previous** - логи предыдущей инстанции перед рестартом.
3. **kubectl get pods** – показывает информацию об имеющихся подах и их статус

Что делать, если контейнер постоянно перезапускается (CrashLoopBackOff)?

**Ответ:** Посмотреть список events и логи.

Как узнать, сколько раз контейнер перезапускался из-за провала LivenessProbe?

**Ответ:** С помощью команды **kubectl get pods** можно увидеть количество перезапусков, которые можно сопоставить с событиями events и логами для установки того, что перезапуски происходят из-за LivenessProbe.

Какие события в kubectl describe pod указывают на проблемы с probes?

**Ответ: Warning Unhealthy Readiness probe failed** и **Warning Unhealthy Liveness probe failed.**

Как проверить, что ReadinessProbe корректно исключает под из балансировки?

**Ответ:** Посмотреть статус Ready в **kubectl get pods** и **kubectl get endpoints <service>** наличие эндпоинта. Пока эндпоинтов сервиса для данного пода нет, значит он исключен из балансировки. После успешной пробы в endpoints появится ip пода.

Как доказать, что два контейнера используют один физический файл?

**Ответ:** Проверить специальной командой в каждом контейнере видимость файла и сравнить inode (он должен быть одинаковым).

Какой тип Probe (Liveness/Readiness) должен сработать, если ваше приложение полностью функционирует, но временно не может обрабатывать новые запросы из-за перегрузки базы данных?

**Ответ:** ReadinessProbe, так как приложение не нуждается в перезагрузке, на него просто не нужно больше слать трафик.

Что произойдет с данными в emptyDir, если узел (node), на котором работает Pod, перезагрузится?

**Ответ:** Данные в emptyDir будут утеряны