**Шадрина 42.2**

**Задание 6.1. Базовые LivenessProbes**

Создайте Deployment с разными типами LivenessProbe:

- exec: проверка существования файла

- httpGet: проверка HTTP эндпоинта (убедитесь, что Pod с httpGet пробой переходит в состояние CrashLoopBackOff)

- TCP: проверка доступности порта

**Проверочные команды**

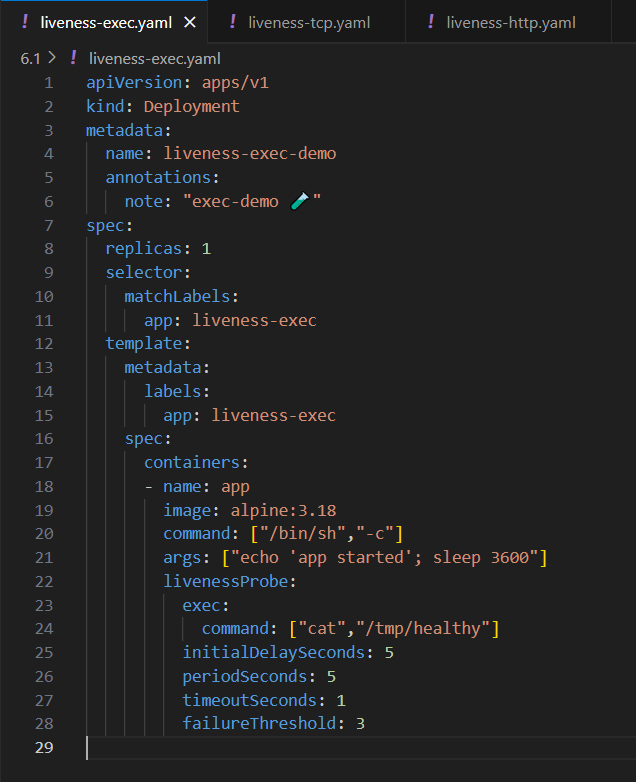
1. Общий мониторинг состояния подов

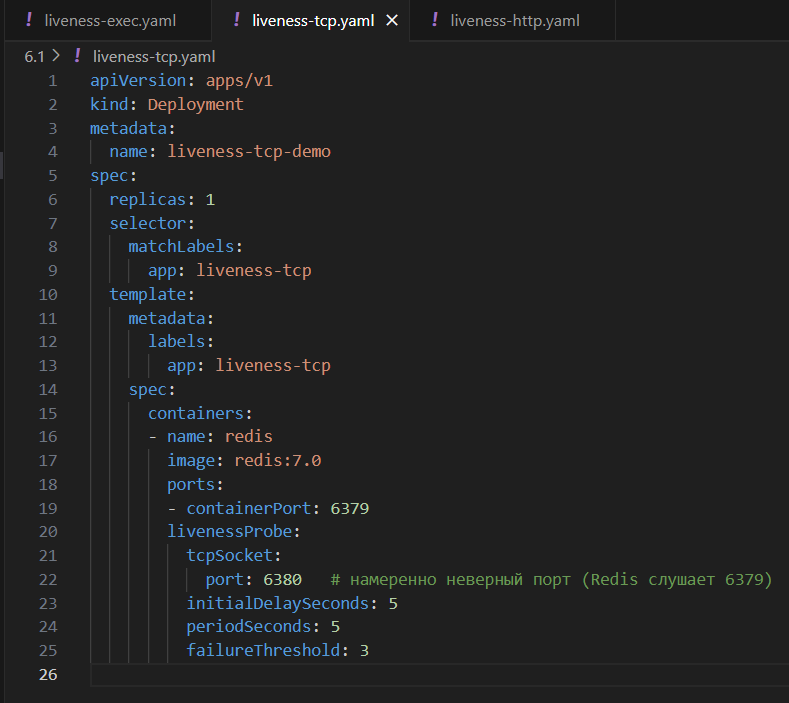
2. Детальная информация о пробах

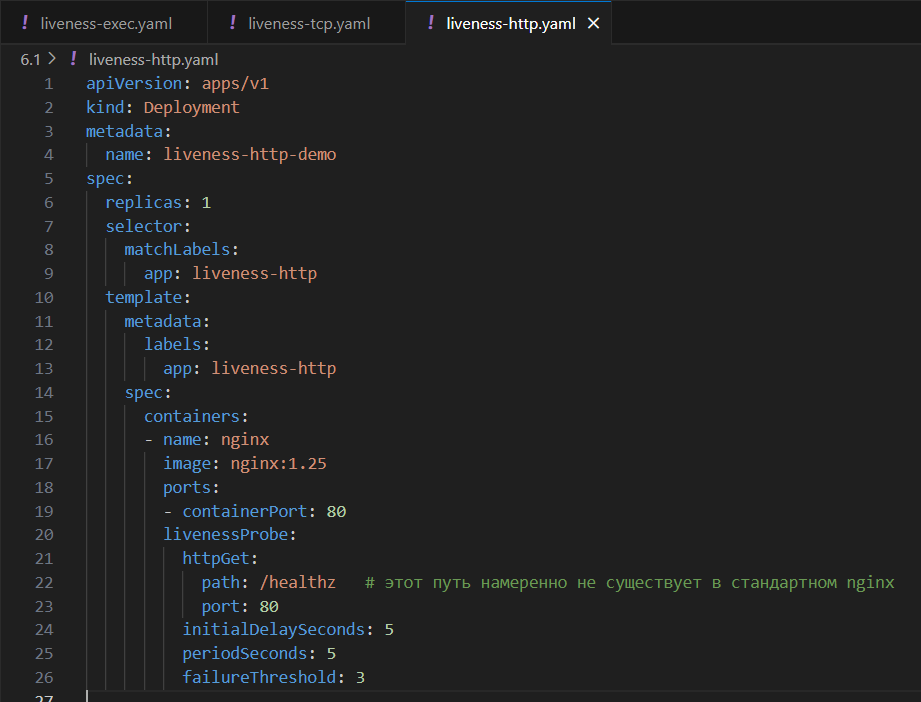
3. Проверка событий провала проб

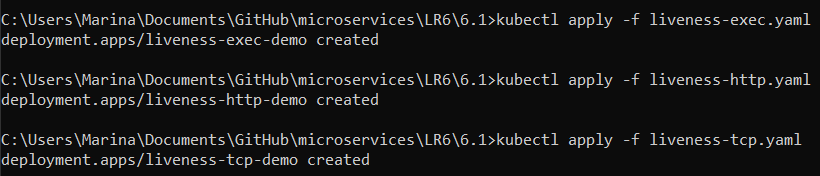
4. Логи контейнера для анализа причин падения

5. Проверка истории перезапусков



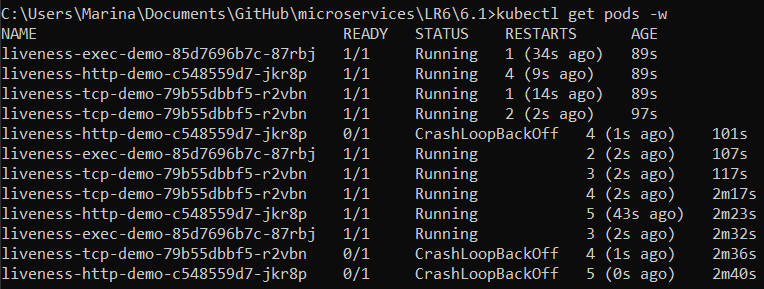






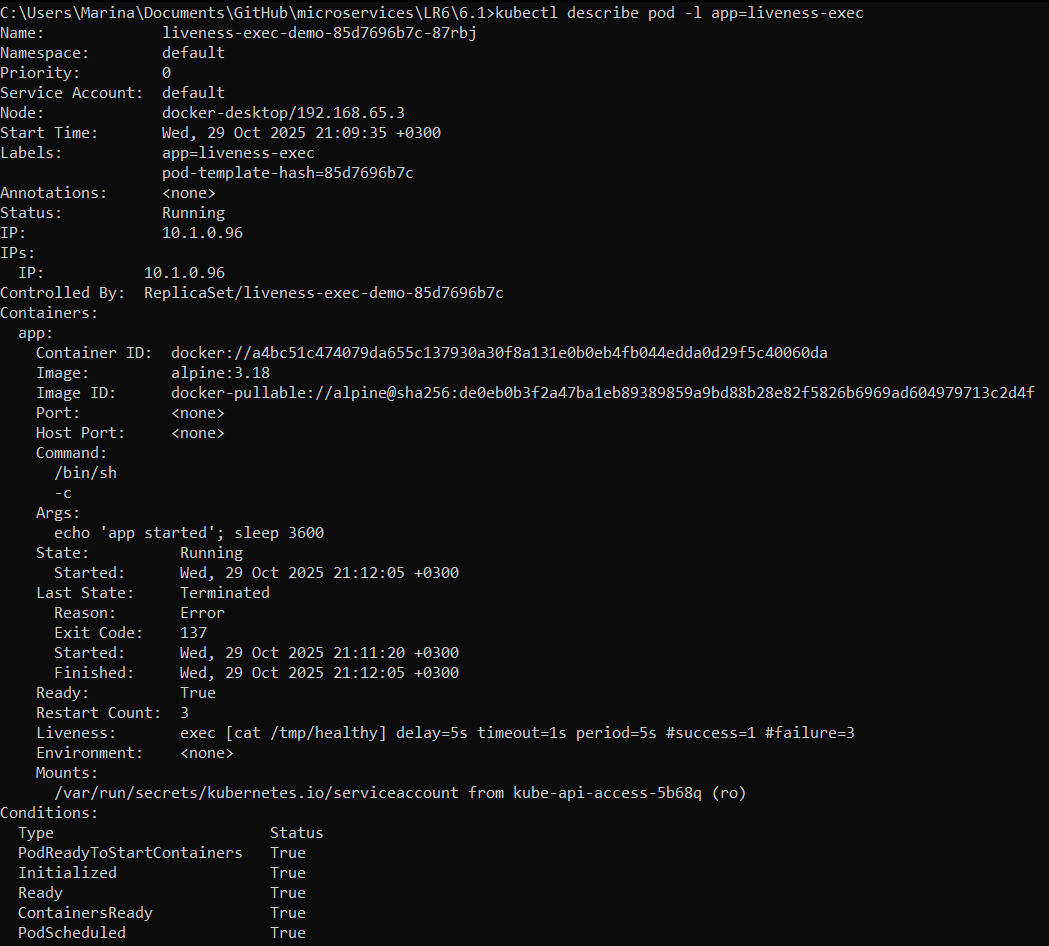
**ПРОВЕРОЧНЫЕ КОМАНДЫ**

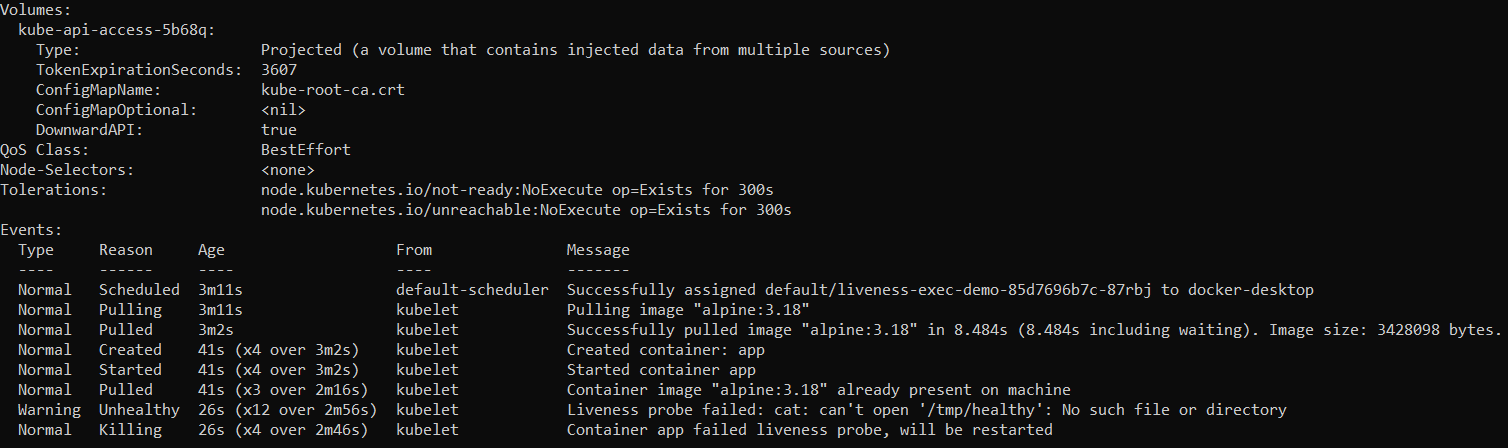
**1. Общий мониторинг состояния подов**

****

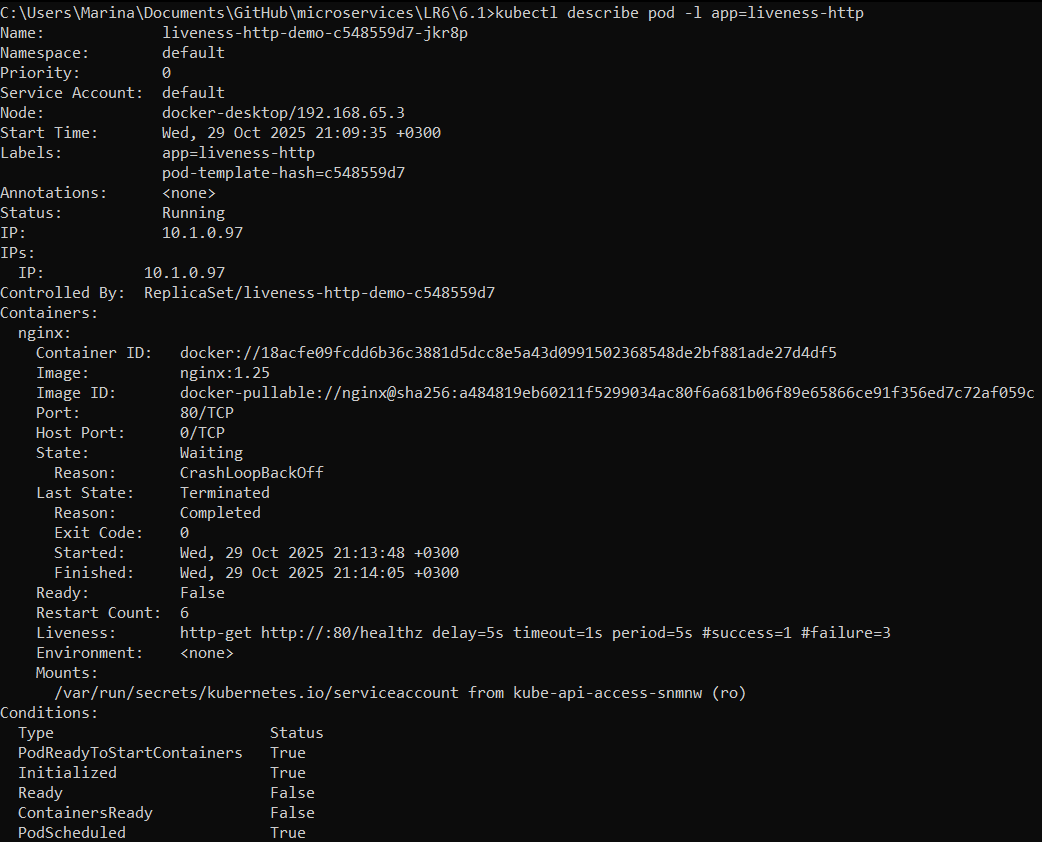
**2. Детальная информация о пробах**

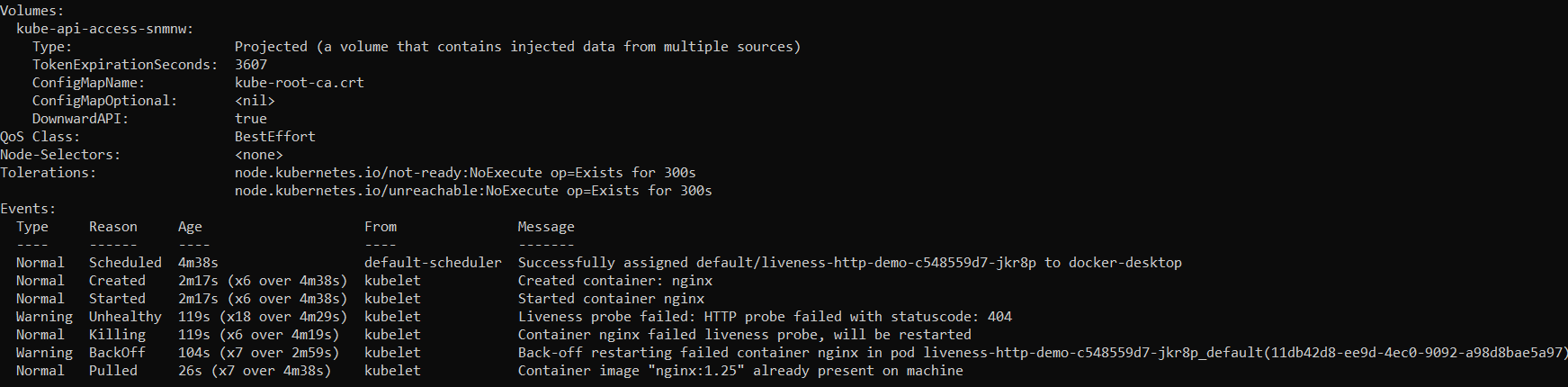
**liveness-exec**

****

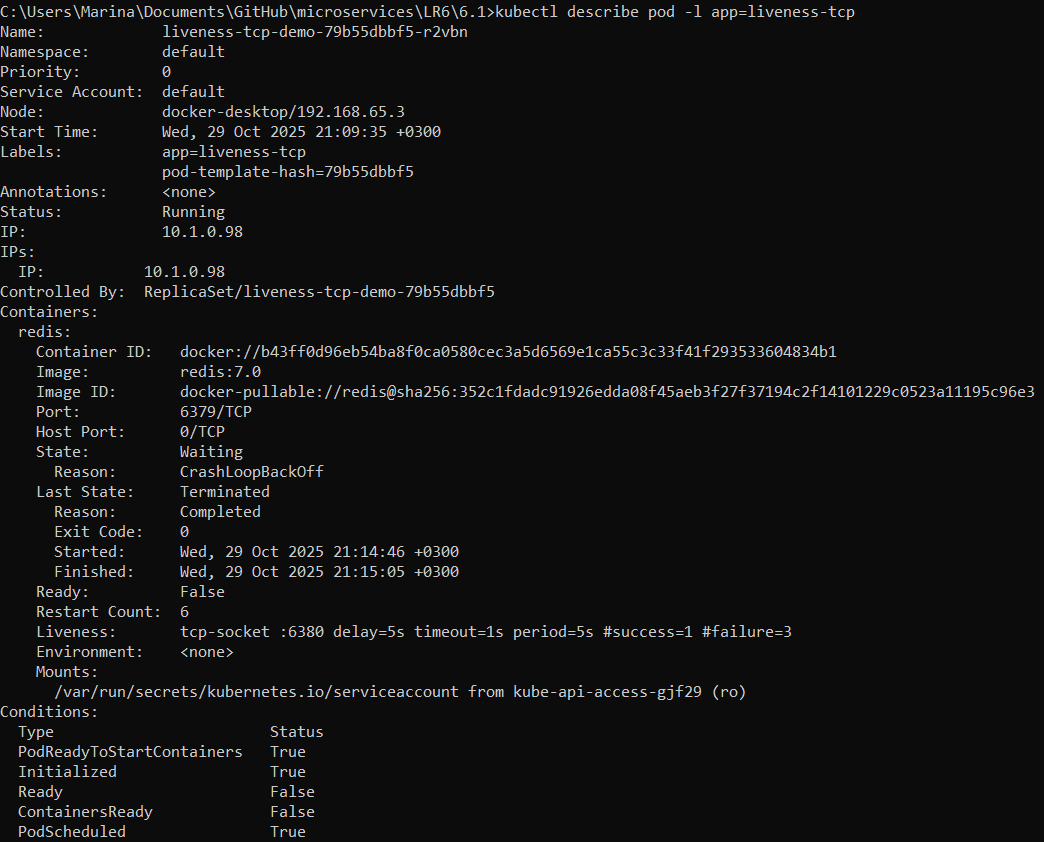
****

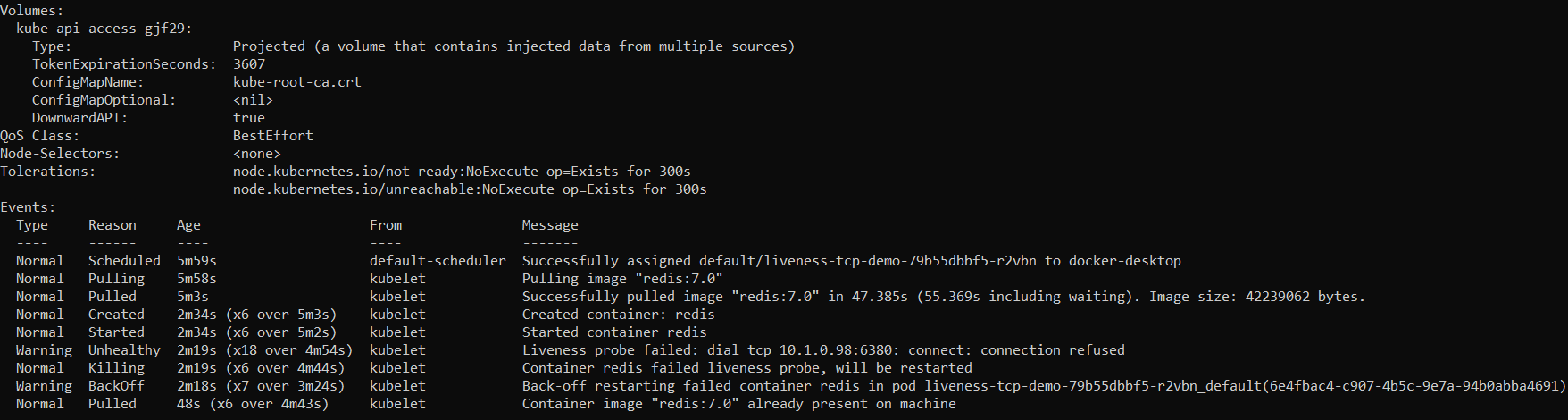
**liveness-http**

****

****

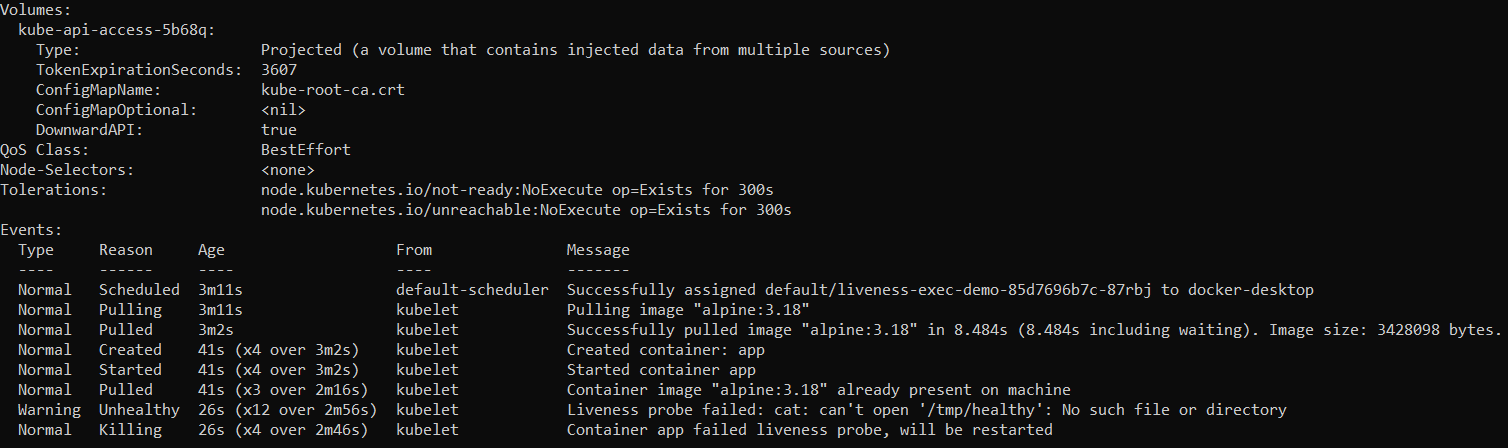
**liveness-tcp**

****

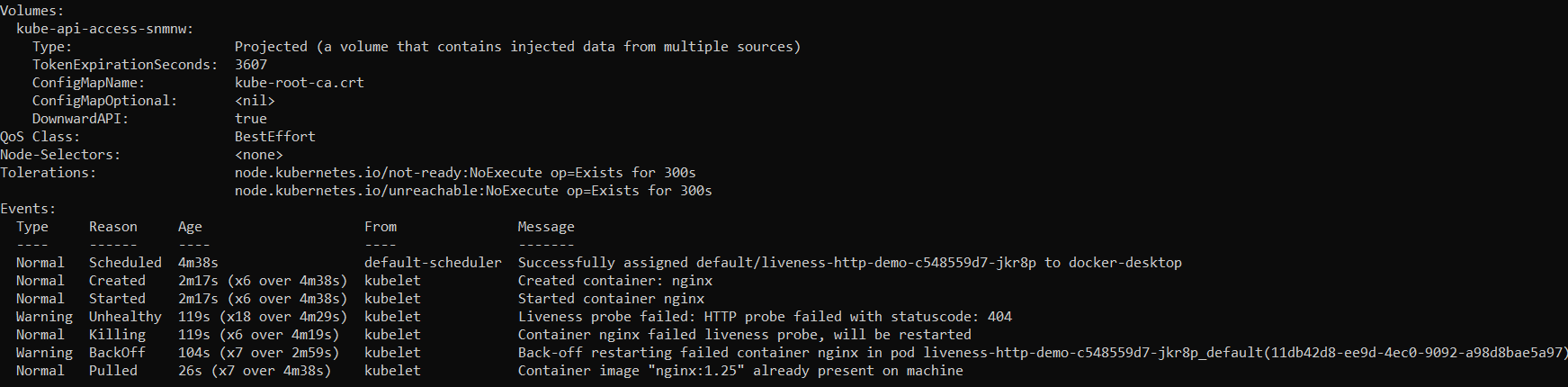
****

**3. Проверка событий провала проб**

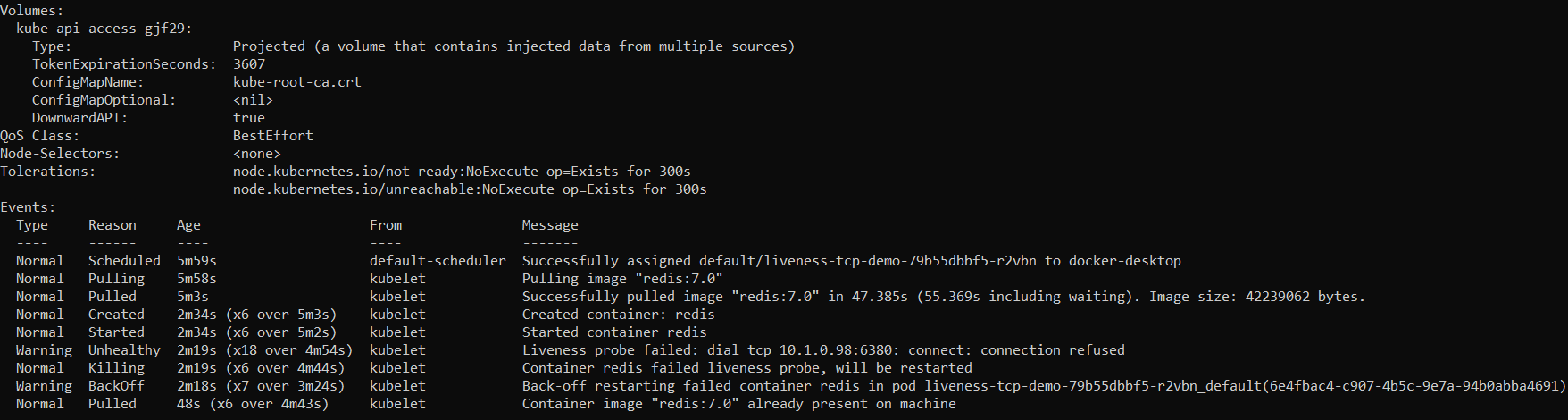
**liveness-exec**

****

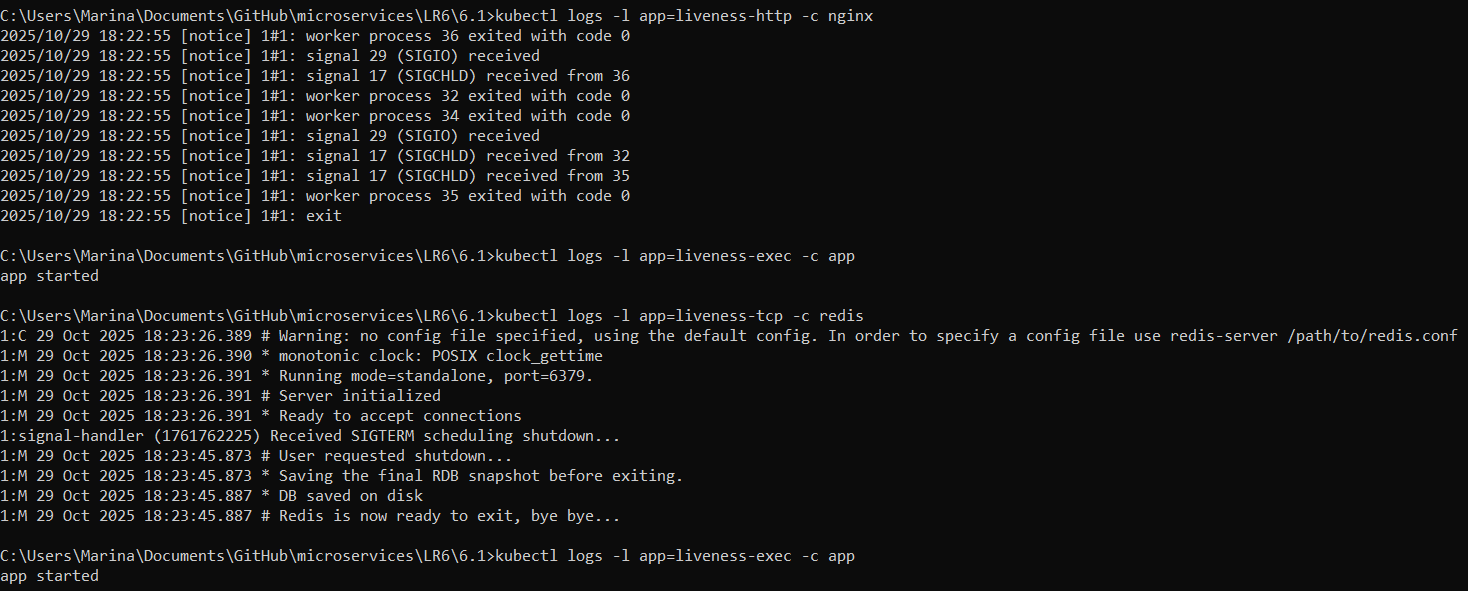
**liveness-http**

****

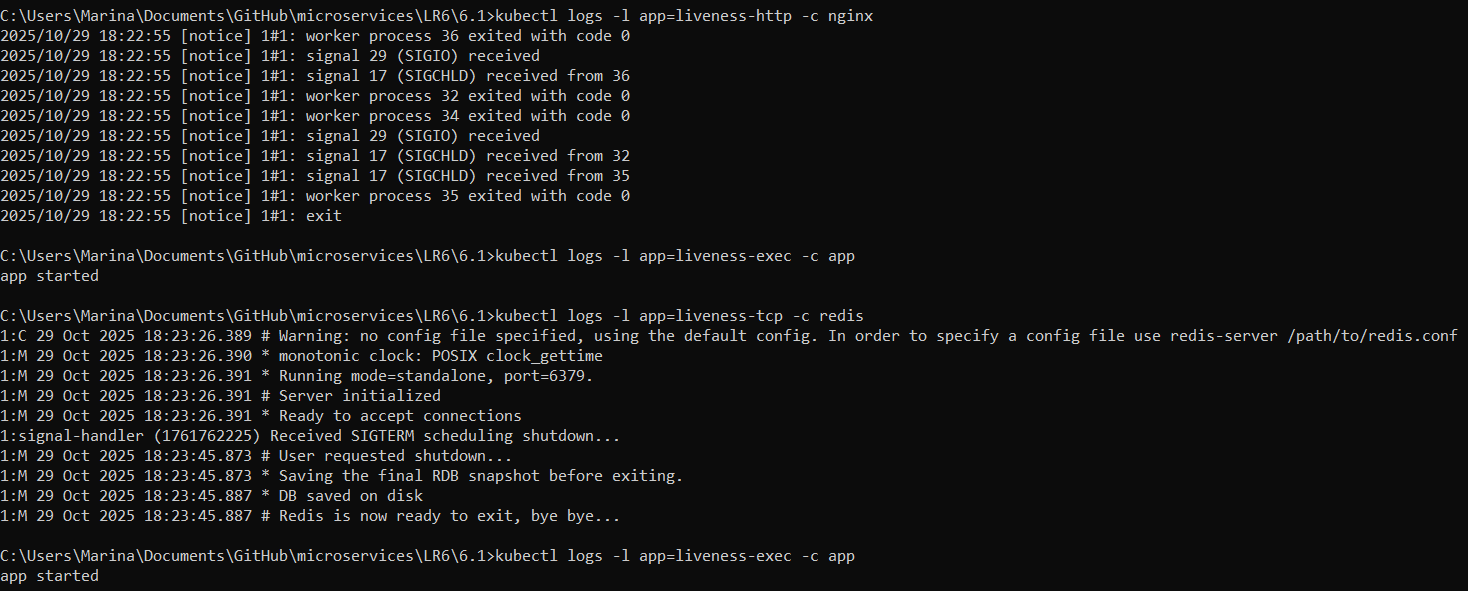
**liveness-tcp**

****

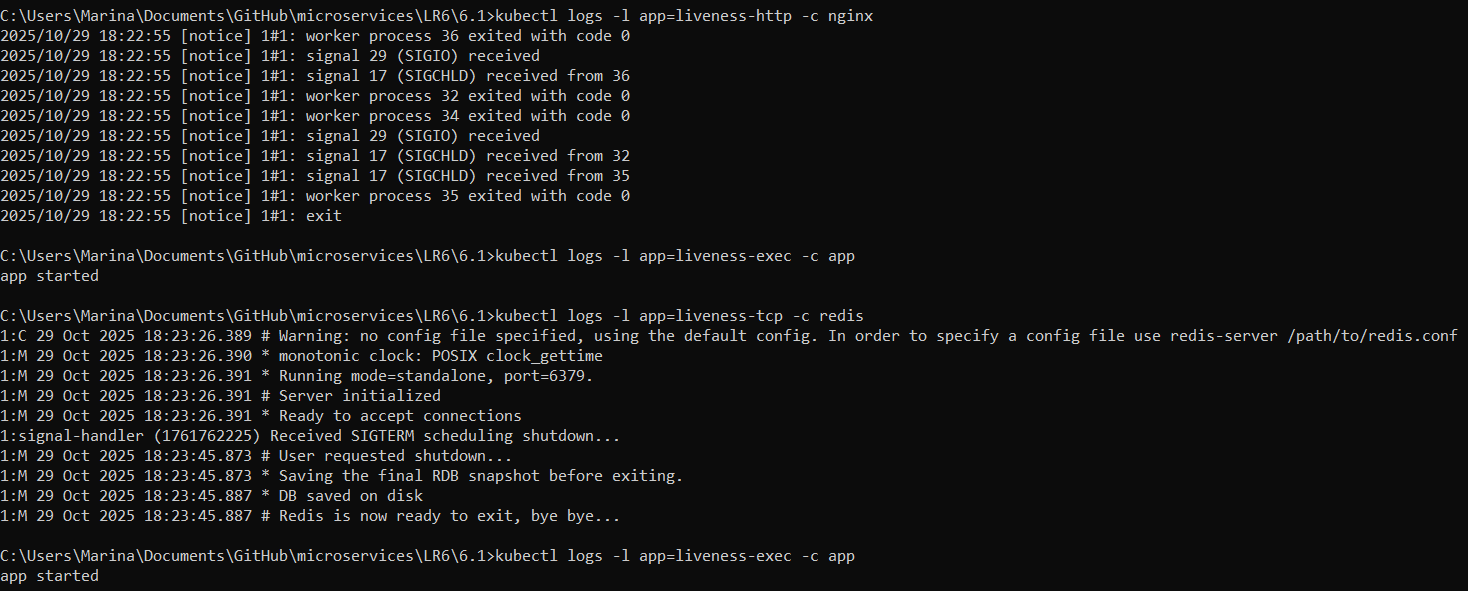
**4. Логи контейнера для анализа причин падения**



контейнер был перезапущен из-за неверного пути /healthz

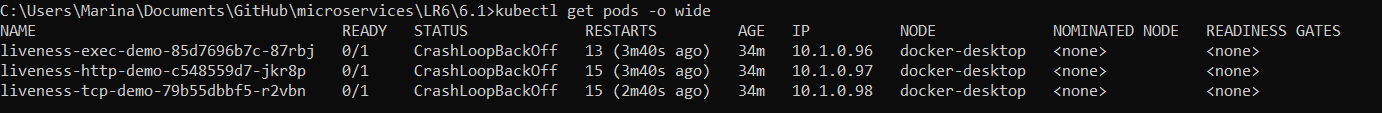


**liveness-probe проверяет 6380**, где ничего не слушает, kubelet не может подключиться, probe фейлится, контейнер получает SIGTERM, контейнер перезапускается



В логах видно только старт (app started), потому что контейнер успевает только запуститься и снова перезапускается.

**5. Проверка истории перезапусков**



**Задание 6.2. ReadinessProbe**

Создайте приложение, которое:

- имеет длительную инициализацию (30+ секунд)

- использует ReadinessProbe для исключения из балансировки во время старта

- демонстрирует разницу между Liveness и Readiness

- используйте sleep 40 в команде инициализации приложения. С помощью kubectl get pods -w наблюдайте, как статус меняется с 0/1 на 1/1 только после окончания инициализации

**Проверочные команды**

1. Наблюдение за изменением статуса Ready

2. Проверка endpoints сервиса до и после готовности

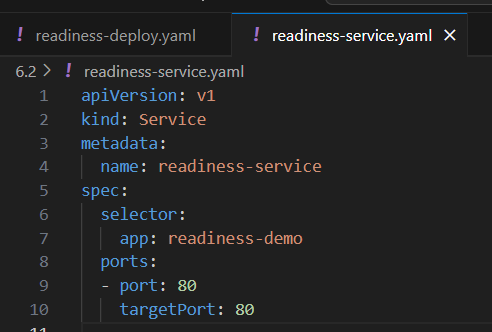
3. Сравнение Liveness и Readiness конфигурации

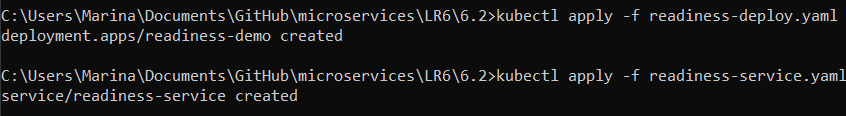
4. Логи инициализации приложения в реальном времени

5. Время перехода в Ready состояние

6. Анализ временной шкалы событий запуска

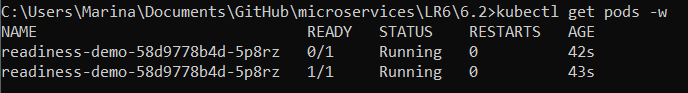




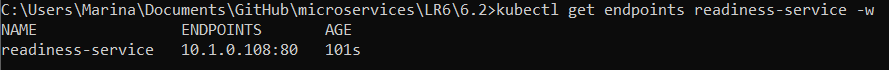


**ПРОВЕРОЧНЫЕ КОМАНДЫ**

**1. Наблюдение за изменением статуса Ready**

****

**2. Проверка endpoints сервиса до и после готовности**

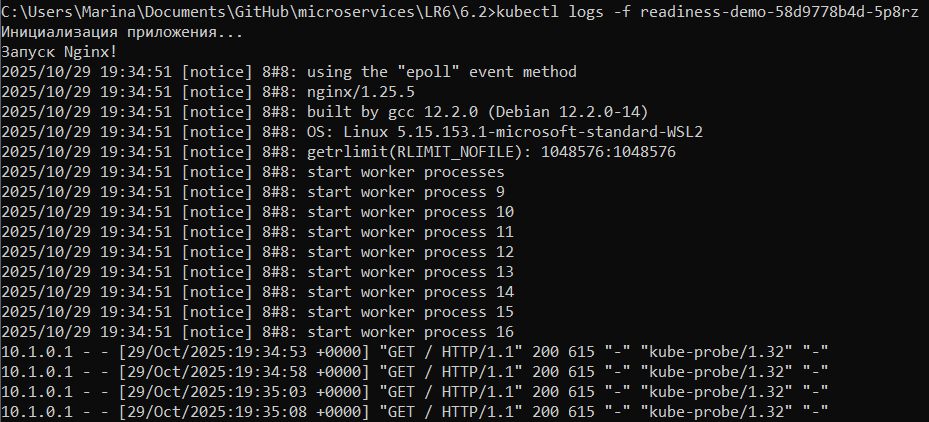
****

**3. Сравнение Liveness и Readiness конфигурации**

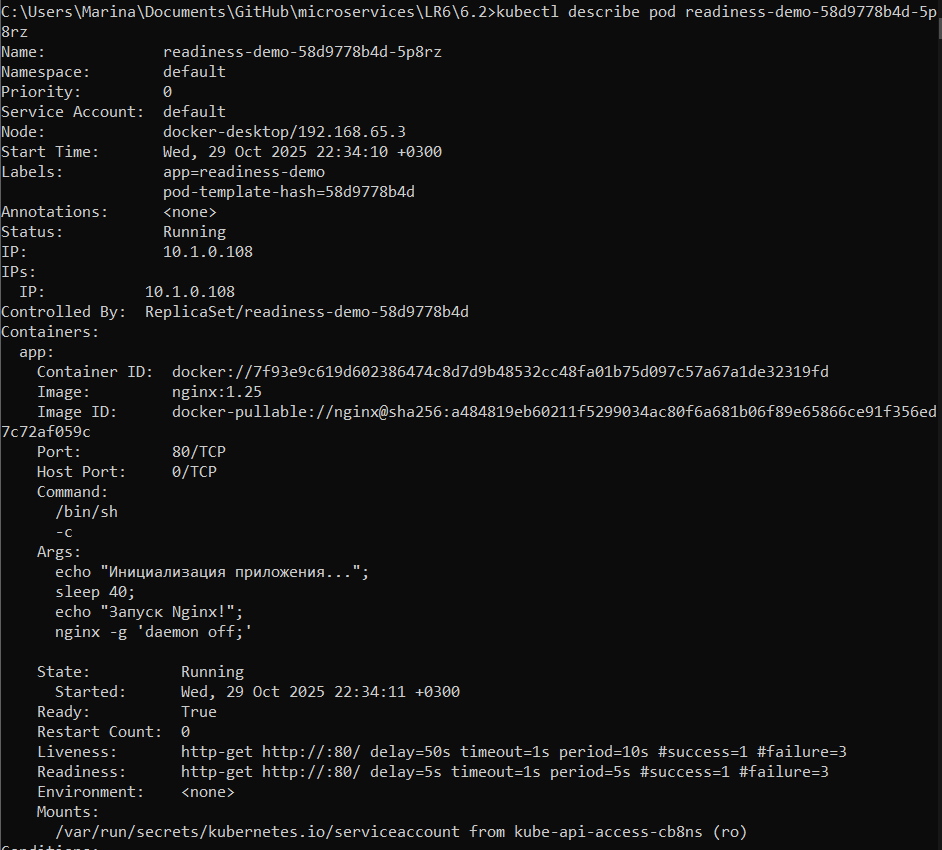
**ReadinessProbe** не перезапускает контейнер, а просто исключает из балансировки.

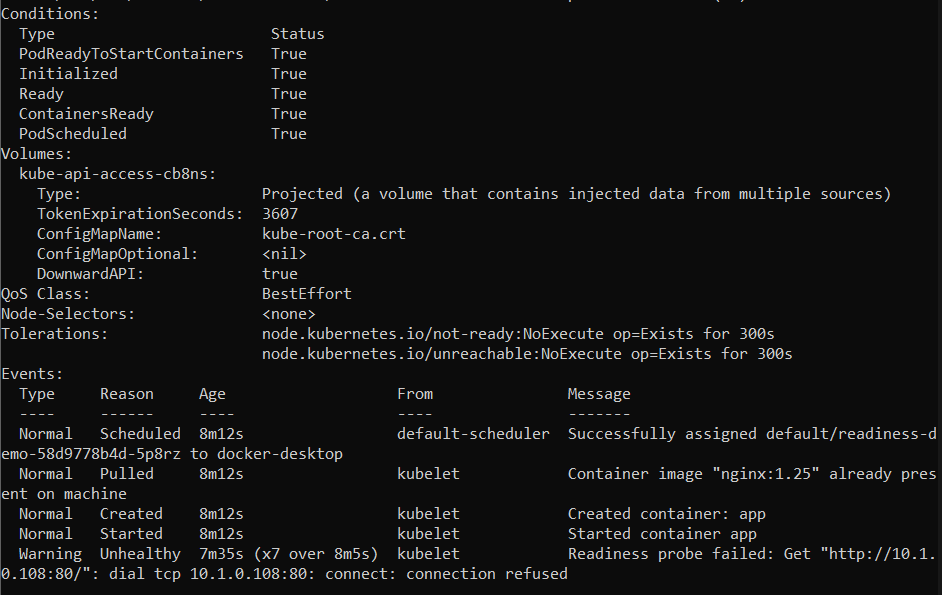
**LivenessProbe** (в нашем случае с initialDelaySeconds: 50) проверит контейнер позже — если он зависнет, произойдёт перезапуск.

**4. Логи инициализации приложения в реальном времени**

****

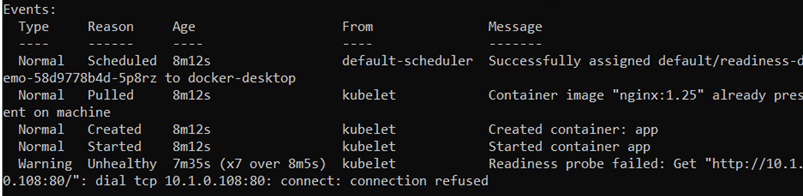
**5. Время перехода в Ready состояние**

****

****

Общее время перехода в состояние Ready — **около 42 секунд** (с учётом периода проб).

**6. Анализ временной шкалы событий запуска**

****

В ходе анализа событий запуска видно, что Pod был успешно назначен и запущен, однако в течение первых 42 секунд ReadinessProbe фиксировала сбои подключения (connection refused). Это связано с искусственной задержкой инициализации, Pod находился в состоянии Running, но не был готов принимать трафик. После завершения инициализации и старта nginx, Readiness-проверка начала проходить успешно, Pod стал Ready=True, и его IP-адрес появился в списке endpoints сервиса. Таким образом, ReadinessProbe корректно исключала Pod из балансировки во время старта приложения.

**Задание 6.3. Volume emptyDir**

Создайте многоконтейнерный Pod с:

- контейнером-генератором данных

- контейнером-читателем данных

- общим emptydir volume для обмена данными

**Проверочные команды**

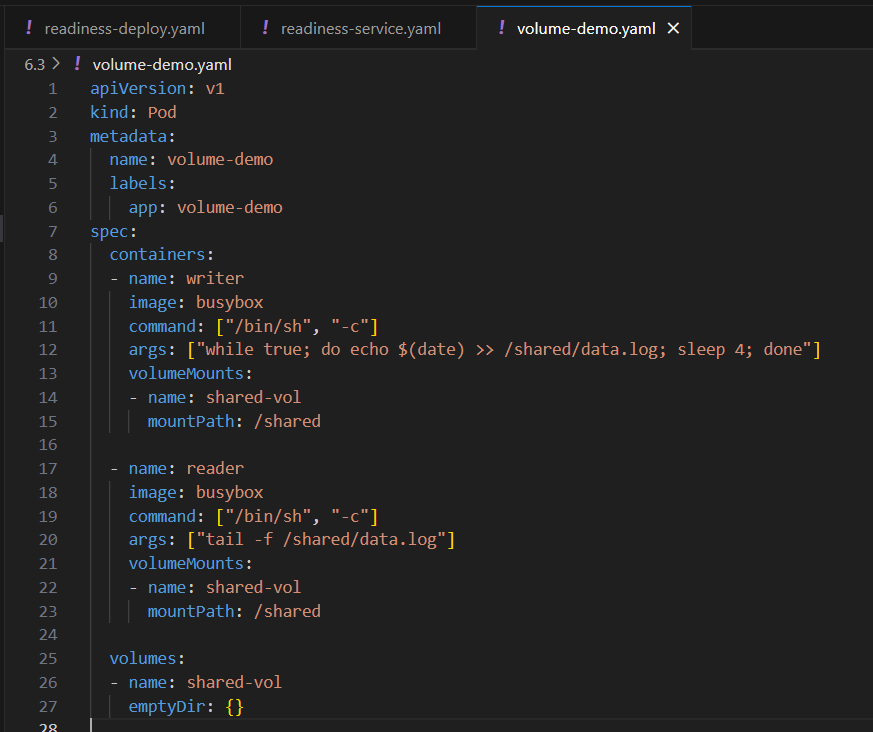
1. Проверка состояния многоконтейнерного пода

2. Проверка монтирования volume в контейнерах

3. Сравнение inode файлов в разных контейнерах

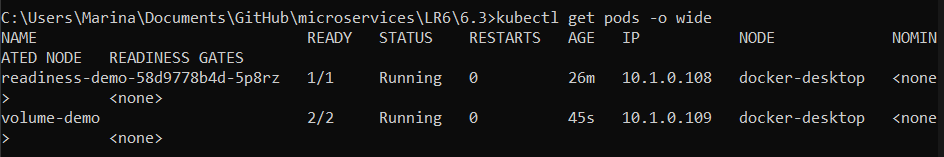
4. Логи генератора данных

5. Логи читателя данных

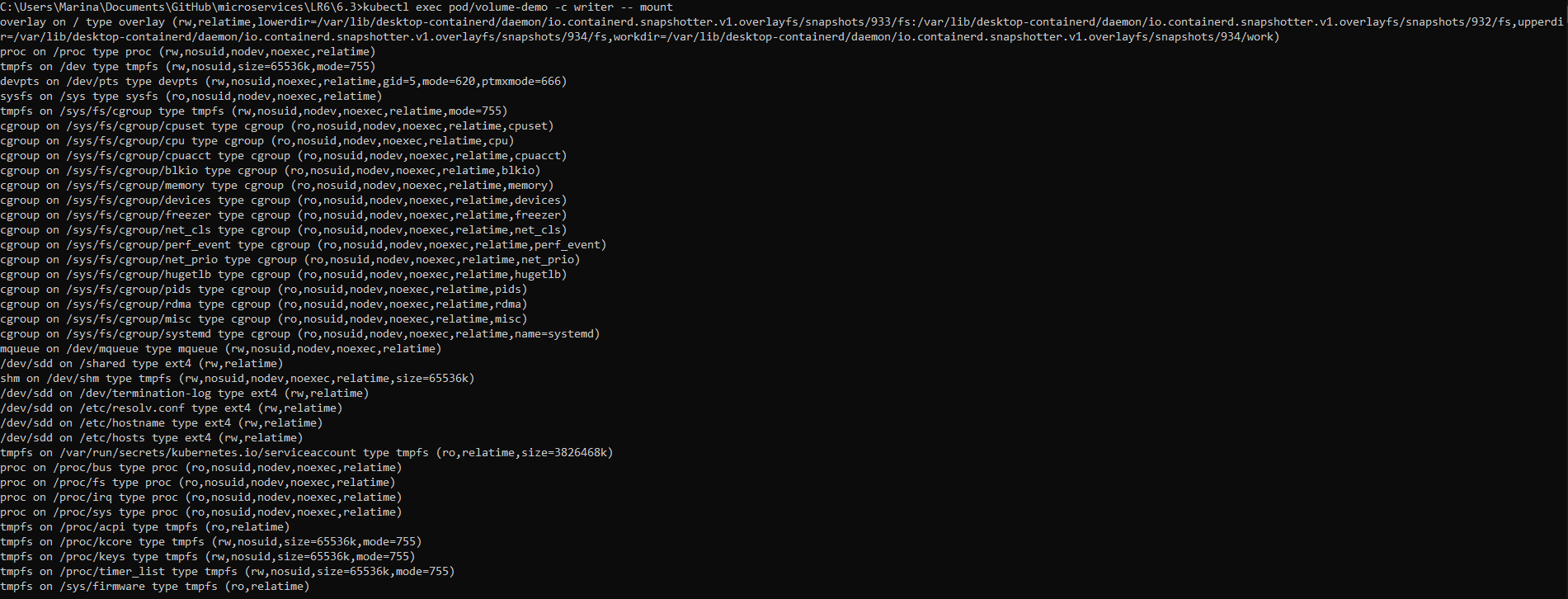


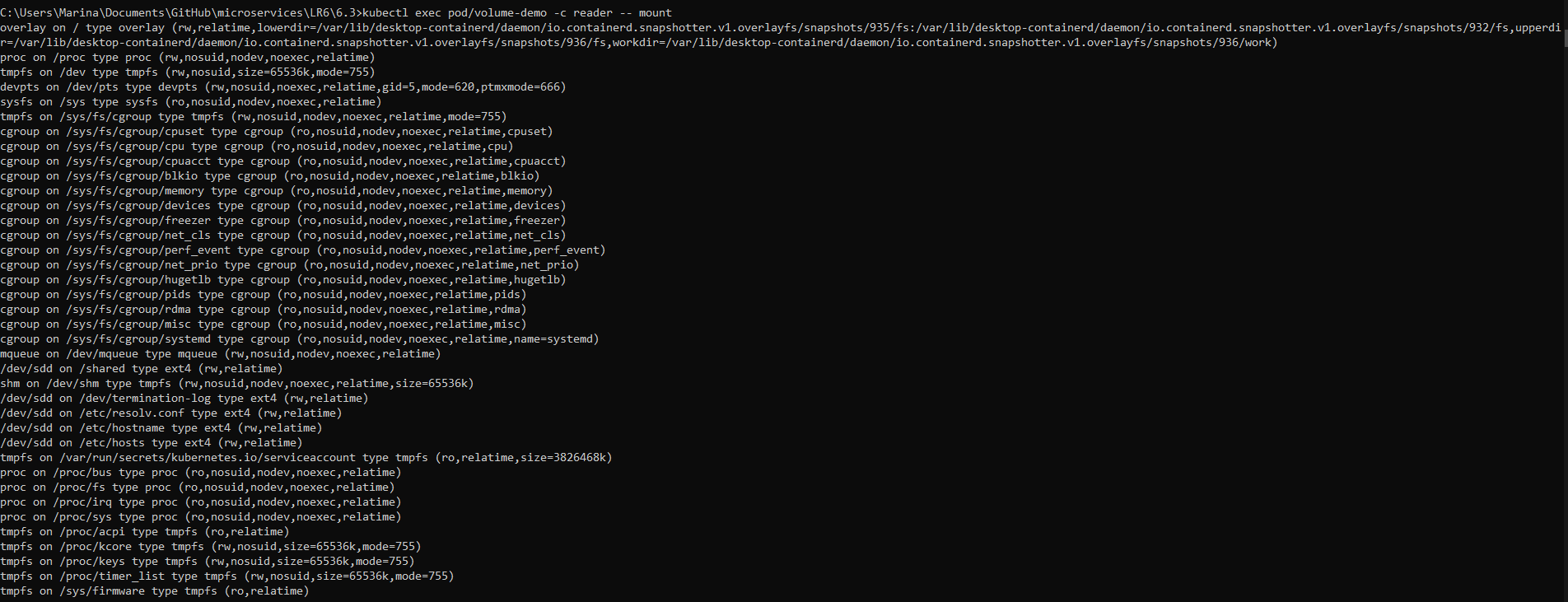
**ПРОВЕРОЧНЫЕ КОМАНДЫ**

**1. Проверка состояния многоконтейнерного пода**

****

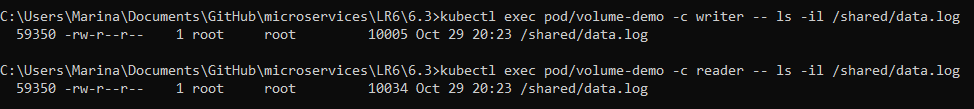
**2. Проверка монтирования volume в контейнерах**

****

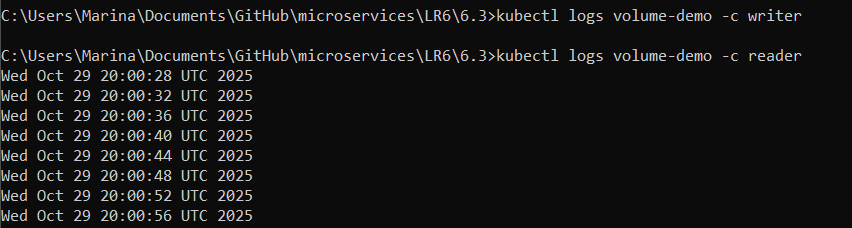
****

Volume одинаково смонтирован в обоих контейнерах /dev/sdd on /shared type ext4 (rw,relatime)

**3. Сравнение inode файлов в разных контейнерах**

****

**4-5. Логи генератора и читателя данных**

****

writer пишет данные во временный файл /shared/data.log, reader контейнер читаeт их в реальном времени через тот же emptyDir, значит всё работает корректно.

**Задание 6.4.**

**Диагностика probes**

Создайте проблемные сценарии и научитесь их диагностировать:

- неправильные параметры таймаутов

- некорректные пути/порты

- анализ событий и логов

- создайте сценарий, где Probe имеет initialDelaySeconds: 2, но приложение стартует 10 секунд. Объясните, что произойдет

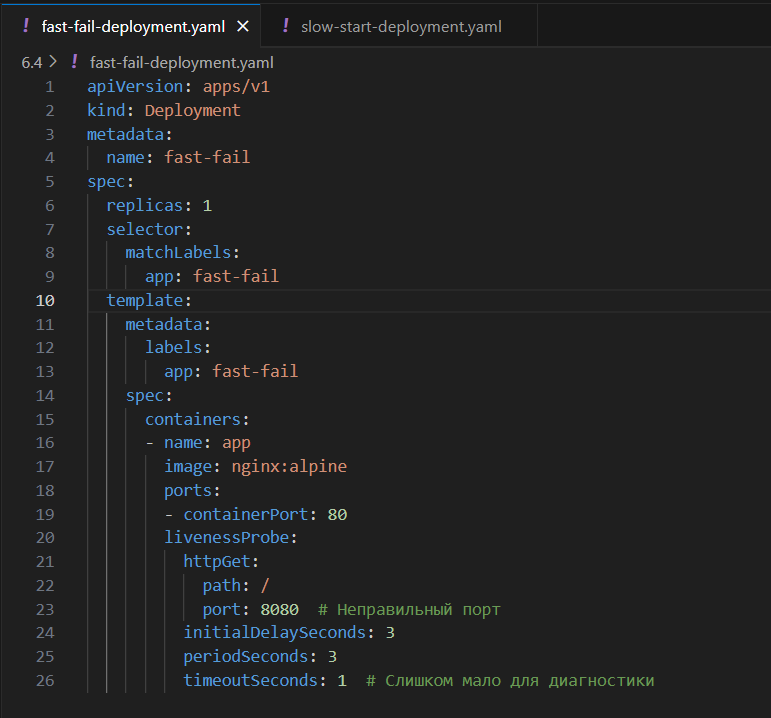
**Проверочные команды**

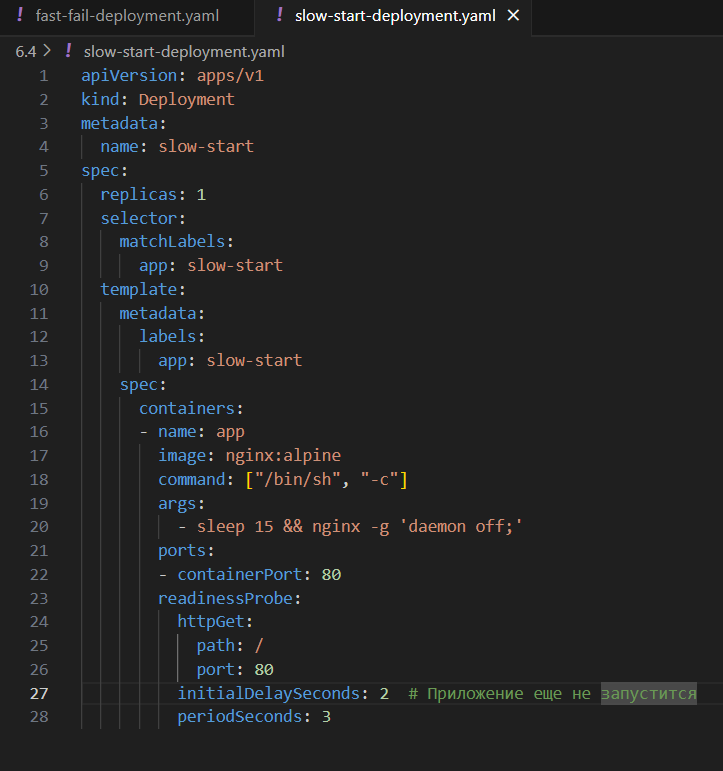
1. Комплексная диагностика проблемного пода

2. Детальный анализ событий

3. Логи предыдущего контейнера после перезапуска

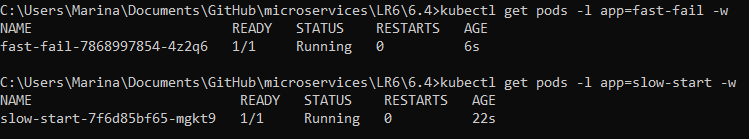
4. Проверка доступности портов внутри контейнера





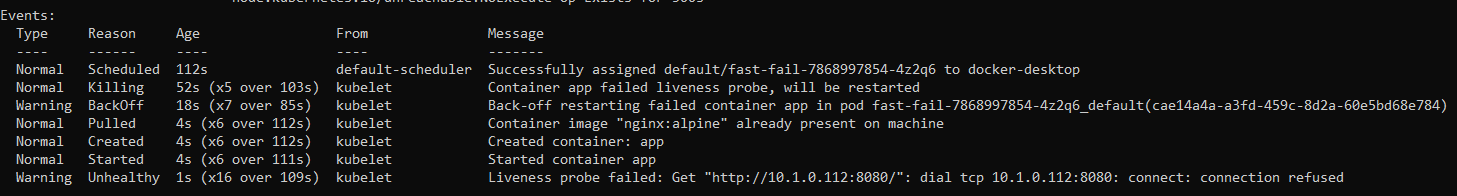
**ПРОВЕРОЧНЫЕ КОМАНДЫ**

**1. Комплексная диагностика проблемного пода**



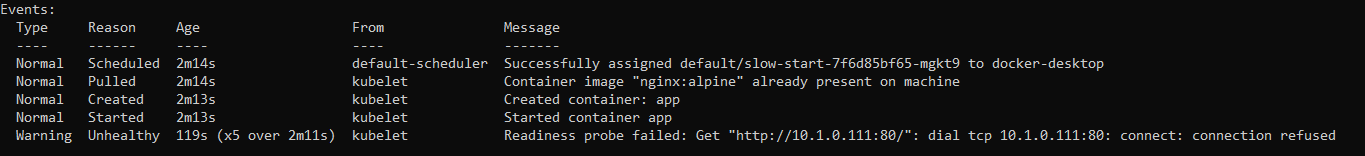
**2. Детальный анализ событий**

**C:\Users\Marina\Documents\GitHub\microservices\LR6\6.4>kubectl describe pod fast-fail-7868997854-4z2q6**

****

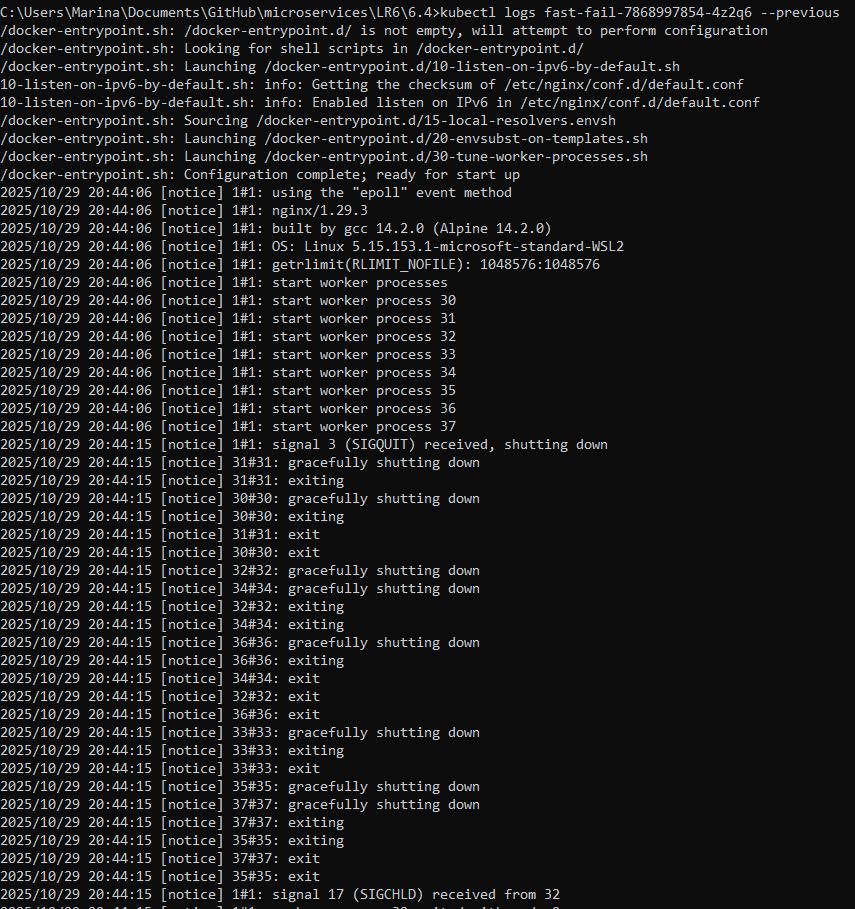
 Liveness probe настроен на неверный порт 8080 вместо 80, что вызывает бесконечный цикл перезапусков контейнера из-за постоянных ошибок подключения, при этом в моменты между пробами контейнер временно показывает статус Ready, но затем снова падает.

**C:\Users\Marina\Documents\GitHub\microservices\LR6\6.4>kubectl describe pod slow-start-7f6d85bf65-mgkt9**

****

Readiness probe начинается слишком рано (через 2 секунды) до фактического старта приложения (15 секунд), что временно исключает под из балансировки, но не вызывает перезапусков, и после полного запуска nginx под успешно переходит в готовое состояние.

**3. Логи предыдущего контейнера после перезапуска**

****

Контейнер nginx успешно запустился и корректно завершил работу, но был принудительно остановлен. Это подтверждает, что проблема не в самом приложении, а в неправильно настроенных пробах здоровья.

****

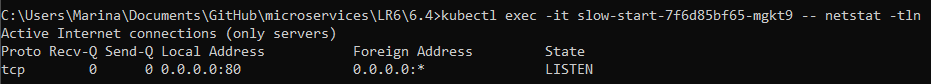
Второй контейнер не перезапускался, что указывает на проблему с readiness probe, а не liveness - приложение работает, но не готово принимать трафик из-за преждевременных проверок.

**4. Проверка доступности портов внутри контейнера**

kubectl exec -it fast-fail-7868997854-4z2q6 -- netstat –tln



kubectl exec -it slow-start-7f6d85bf65-mgkt9 -- netstat -tln



**Контрольные вопросы для всех заданий**

**В чем основное отличие LivenessProbe от ReadinessProbe?**

**Ответ:** LivenessProbe определяет, не "завис" ли контейнер и не требуется ли его перезапуск. ReadinessProbe оценивает, может ли контейнер начать обслуживать пользовательские запросы.

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

**Ответ:**

exec: Запускает произвольную команду внутри контейнера. Применяется для сложных проверок, которые нельзя выполнить простым HTTP-запросом (например, проверка состояния файла или процесса).

httpGet: Выполняет HTTP GET-запрос по указанному пути и порту. Стандартный выбор для веб-сервисов и REST API.

tcpSocket: Проверяет, открыт ли указанный TCP-порт. Используется для сервисов, которые не используют HTTP, но работают с сетевыми подключениями (например, базы данных).

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

**Ответ:** Pod будет помечен как готовый сразу после запуска контейнера. Сервис начнет отправлять трафик на приложение, которое еще не инициализировалось, что приведет к ошибкам для пользователей.

**Какой смысл параметров initialDelaySeconds, periodSeconds и timeoutSeconds?**

**Ответ:**

initialDelaySeconds: Задержка в секундах между стартом контейнера и началом выполнения пробы. Нужна, чтобы дать приложению время подготовиться.

periodSeconds: Периодичность выполнения проверки в секундах. Определяет, как часто kubelet будет выполнять probe.

timeoutSeconds: Время в секундах, по истечении которого проверка считается неудачной, если от нее не поступило ответа.

**Что означает failureThreshold и successThreshold?**

**Ответ:**

successThreshold: Минимальное количество последовательных успешных проверок, необходимое для того, чтобы Probe, ранее бывшая неудачной, была признана работающей.

failureThreshold: Количество последовательных неудачных проверок, после которого система сочтет контейнер неработоспособным (для LivenessProbe это приведет к перезапуску).

**Что такое emptyDir и для каких сценариев он предназначен?**

**Ответ:** emptyDir — это временное хранилище, которое создается при запуске Pod и существует до его удаления. Основные сценарии: обмен данными между контейнерами в одном Pod (например, когда один контейнер генерирует файлы, а другой их обрабатывает) и использование диска в качестве кеша.

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

**Ответ:**

kubectl describe pod <имя-pod> — просмотр событий (Events) и состояния контейнеров.

kubectl logs <имя-pod> -c <имя-контейнера> --previous — анализ логов контейнера из предыдущей попытки запуска перед его перезагрузкой.

kubectl get pods -w — мониторинг статуса Pod'ов в реальном времени, включая количество рестартов и состояние готовности.

**Что делать, если контейнер постоянно перезапускается (CrashLoopBackOff)?**

**Ответ:** Необходимо изучить логи контейнера и события Pod'а, чтобы найти причину сбоя. Команды kubectl logs и kubectl describe pod покажут ошибки приложения или проблемы с пробами.

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

**Ответ:** В выводе команды kubectl get pods отображается общее количество перезапусков (RESTARTS). Чтобы убедиться, что причиной являются провалы LivenessProbe, нужно проверить события Pod'а (kubectl describe pod) на наличие сообщений Liveness probe failed.

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

**Ответ**: Сообщения типа Warning Unhealthy: Liveness probe failed и Warning Unhealthy: Readiness probe failed прямо указывают на сбои проверок жизнеспособности и готовности.

**Как проверить, что ReadinessProbe корректно исключает под из балансировки?**

**Ответ**: Убедиться, что Pod не находится в статусе Ready/1 при неудачной пробе (команда kubectl get pods). Также можно проверить, что IP-адрес этого Pod'а отсутствует в списке эндпоинтов сервиса (kubectl get endpoints <service-name>).

**Как доказать, что два контейнера используют один физический файл?**

**Ответ:** Нужно зайти в каждый контейнер и проверить inode файла с помощью команды ls -i. Если inode совпадает, значит, оба контейнера видят один и тот же физический файл на диске.

**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. Приложение работает корректно и не требует перезапуска, но временно не должно получать новый трафик, пока не восстановит подключение к базе данных.

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

**Ответ:** Все данные в emptyDir будут безвозвратно утеряны, так как это временное хранилище, физически расположенное на узле, и его жизненный цикл неразрывно связан с жизнью Pod'а.