DOI 10.37882/2223-2966.2026.02.35

ШАБЛОН РАЗВЕРТЫВАНИЯ И УПРАВЛЕНИЯ СОСТОЯНИЕМ ДЛЯ РАСПРЕДЕЛЕННЫХ КЭШЕЙ ВТОРОГО УРОВНЯ (L2) В KUBERNETES-КЛАСТЕРАХ С ЦЕЛЬЮ ОБЕСПЕЧЕНИЯ ZERO-DOWNTIME ОПЕРАЦИЙ

DEPLOYMENT AND STATE MANAGEMENT TEMPLATE FOR SECOND-LEVEL DISTRIBUTED CACHES (L2) IN KUBERNETES CLUSTERS, IN ORDER TO ENSURE ZERO-DOWNTIME OPERATIONS

P. Romanovsky
V. Smetanin
A. Kapliyenko
E. Samborenko
A. Rusakov

Summary. Within the framework of this scientific work, the current problem of ensuring high availability and fault tolerance of distributed second-level (L2) caching systems in the Kubernetes container orchestration environment is investigated. The author conducts an in-depth analysis of the limitations of standard controllers (Deployment), revealing their unsuitability for servicing stateful applications with strict requirements for maintaining the state and stability of network identification. The article proposes and practically implements a specialized deployment template based on the Redis Cluster and the StatefulSet controller, which integrates Headless service mechanisms to provide deterministic addressing and Persistent Volume Claims to save data on local node storage. The practical significance of the work is confirmed by a series of experiments in the K3s cluster, including simulation of emergency failures and scheduled updates. The test results (verified by data write and read operations after node restart) prove the possibility of providing zero-downtime operations and significantly reducing cache recovery time, which prevents degradation of the performance of underlying databases in highly loaded systems.

Keywords: Kubernetes, orchestration, Redis Cluster, L2 cache, StatefulSet, fault tolerance, zero-downtime, cloud computing, Persistent Volume, Headless Service, K3s, High Availability, High Availability, Stateful storage, data sharding.

Романовский Петр Юрьевич
Российский технологический университет МИРЭА
Сметанин Виктор Максимович
Российский технологический университет МИРЭА
Каплиенко Александра Михайловна
Российский технологический университет МИРЭА
Самборенко Евгений Алексеевич
Российский технологический университет МИРЭА
mduejej@gmail.com
Русаков Алексей Михайлович
Кандидат технических наук, старший преподаватель, Российский технологический университет МИРЭА

Аннотация. В рамках данной научной работы исследуется актуальная проблема обеспечения высокой доступности и отказоустойчивости распределенных систем кэширования второго уровня (L2) в среде контейнерной оркестрации Kubernetes. Проводится глубокий анализ ограничений стандартных контроллеров (Deployment), выявляя их непригодность для обслуживания stateful-приложений с жесткими требованиями к сохранению состояния и стабильности сетевой идентификации. В статье предложен и практически реализован специализированный шаблон развертывания на базе Redis Cluster и контроллера StatefulSet, интегрирующий механизмы Headless-сервисов для обеспечения детерминированной адресации и Persistent Volume Claims для сохранения данных на локальных накопителях узлов. Практическая значимость работы подтверждена серией экспериментов в кластере K3s, включающих имитацию аварийных сбоев и плановых обновлений. Результаты тестирования (верифицированные операциями записи и чтения данных после рестарта узлов) доказывают возможность обеспечения zero-downtime операций и существенного сокращения времени восстановления кэша, что предотвращает деградацию производительности нижележащих баз данных в высоконагруженных системах.

Ключевые слова: Kubernetes, оркестрация, Redis Cluster, L2-кэш, StatefulSet, отказоустойчивость, zero-downtime, облачные вычисления, Persistent Volume, Headless Service, K3s, высокая доступность, High Availability, сохранение состояния, шардирование данных.

Введение

Развитие современной микросервисной архитектуры и рост популярности высоконагруженных платформ (онлайн-кинотеатры, маркетплейсы, игровые сервисы) предъявляют беспрецедентные требования к скорости обработки запросов [1]. В этих условиях кэширование данных второго уровня (L2) становится критически важным звеном инфраструктуры, напрямую влияющим на пользовательский опыт (latency) и стабильность систем [2]. Однако перенос таких stateful-сервисов в динамическую среду Kubernetes сопряжен с серьезными технологическими вызовами.

Стандартные методы оркестрации, ориентированные на эфемерность контейнеров, не учитывают специфику распределенных кэшей, где потеря состояния одного узла или изменение его сетевого адреса может привести к «кэш-шторму» (cache miss storm) — лавинообразному росту нагрузки на основную базу данных [3]. Отсутствие формализованных, отказоустойчивых шаблонов эксплуатации приводит к вынужденным простоям при плановых обновлениях и увеличивает время восстановления сервиса после аварий. Данная работа направлена на решение задачи непрерывного функционирования L2-кэша путем разработки архитектурного шаблона, гарантирующего сохранность данных и связность кластера в любых операционных сценариях [4, 5].

Цель исследования: Разработка и экспериментальная верификация шаблона управления распределенным кэшем второго уровня в Kubernetes (на примере Redis Cluster), обеспечивающего выполнение технологических операций с нулевым простоем (zero-downtime) и гарантированное сохранение состояния данных при отказах узлов.

В ходе выполнения исследования были определены и решены следующие задачи:

1. Проанализировать недостатки стандартных средств оркестрации (Deployment, ReplicaSet) при работе с распределенными системами хранения данных. Спроектировать архитектуру шаблона, интегрирующую механизмы стабильной сетевой идентификации и персистентного хранения данных для обеспечения жизненного цикла узлов кэша.
2. Спроектировать архитектуру шаблона, базирующуюся на использовании StatefulSet, Headless Service и динамических Persistent Volumes для обеспечения жизненного цикла узлов.
3. Развернуть экспериментальный стенд в инфраструктуре K3s, сконфигурировав отказоустойчивый Redis Cluster из шести узлов (мастер-узлы и реплики).
4. Апробировать механизм безопасного обновления кластера (rollout), исключающий потерю кворума и доступности сегментов данных.
5. Экспериментально подтвердить эффективность сохранения состояния данных («Alice/Bob» test) при имитации аварийного удаления подов.
6. Верифицировать стабильность сетевой связности кластера при использовании FQDN-идентификаторов вместо динамических IP-адресов.

Анализ недостатков стандартных средств оркестрации Kubernetes

Стандартная парадигма оркестрации в Kubernetes строится на принципах эфемерности и взаимозаменяемости компонентов (подход «cattle, not pets»). Это идеально подходит для приложений без сохранения состояния (stateless), однако создает фундаментальный технологический барьер при развертывании распределенных L2-кэшей. Основная проблема заключается в «конфликте жизненных циклов»: контроллер Kubernetes стремится к динамическому перераспределению ресурсов, в то время как распределенный кэш (например, Redis Cluster) требует жесткой детерминированности сетевой идентификации и топологической связности данных [6].

При использовании стандартных контроллеров типа Deployment или ReplicaSet возникают критические риски, связанные с нарушением консистентности кластера [7]. Поскольку данные в L2-кэше шардированы (распределены по сегментам), потеря одного узла без сохранения его состояния приводит к выпадению значительной части хеш-слотов. Стандартный механизм RollingUpdate не учитывает роли узлов (Master/Slave), что в условиях высоконагруженных систем неизбежно ведет к деградации производительности.

Для систематизации выявленных проблем проведем сравнительный анализ эксплуатационных характеристик стандартных средств оркестрации и специфических требований распределенных систем кэширования (Таблица 1).

Дополнительным фактором риска является стандартный механизм проверок готовности (Health Checks). В классических сценариях проверка порта 6379 является достаточной, однако для распределенного кэша этого мало. Узел может иметь открытый порт, но находиться в состоянии изоляции от кластера или в процессе долгой загрузки данных с диска. Стандартные средства оркестрации не позволяют «из коробки» учитывать внутренний статус готовности распределенной системы, что приводит к преждевременному направлению клиентского трафика на несинхронизированный узел.

Таким образом, для обеспечения операций уровня zero-downtime необходимо отойти от использования базовых контроллеров в пользу шаблона на базе StatefulSet [8]. Этот контроллер обеспечивает стабильный сетевой идентификатор (например, redis-cluster-0), упорядоченное развертывание и, что наиболее важно, сохранение связи между конкретным экземпляром приложения и его томом данных Persistent Volume (PV). Это создает необходимый фундамент для реализации механизмов автоматического восстановления и бесшовного обновления конфигурации [9].

Проектирование архитектуры и реализация стенда

Для реализации и верификации отказоустойчивого шаблона был спроектирован экспериментальный стенд на базе облегченного дистрибутива Kubernetes — K3s.

Таблица 1.

Сравнительный анализ ограничений стандартных абстракций Kubernetes для stateful-сервисов

Критерий эксплуатации	Поведение контроллера Deployment	Требование распределенного L2-кэша	Риски для стабильности системы
Сетевая идентификация	Динамическое назначение IP-адресов и случайных имен подов	Необходимость фиксированных FQDN-имен для поддержания Gossip-протокола	Нарушение топологии кластера и невозможность автоматического повторного присоединения (rejoin) узла
Управление хранилищем	Использование временных хранилищ (EmptyDir), полностью очищаемых при рестарте контейнера	Персистентность дискового кэша (AOF/RDB) для минимизации времени «прогрева» системы	Возникновение кэш-штормов (cache miss) и лавинообразная нагрузка на основную базу данных
Порядок жизненного цикла	Параллельный и произвольный порядок запуска и остановки реплик в группе	Строгая последовательность операций для обеспечения кворума и корректной синхронизации данных	Конфликты при записи и высокий риск расслоения данных (состояние split-brain)
Стратегия ротации	Ротация подов на основе общих лимитов доступности (maxUnavailable) без учета логики приложения	Приоритетный учет ролей узлов для предотвращения одновременного отключения мастера и его реплики	Частичная потеря доступности сегментов (слотов) данных и переход всего кластера в статус Fail
Связывание ресурсов	Отсутствие гарантии возврата пересозданного пода к конкретному ранее выделенному тому данных	Детерминированная привязка пары «узел-том» через механизм шаблонов VolumeClaimTemplates	Необходимость проведения ресурсоемкой полной ресинхронизации данных по сети (Full Snapshot)

Выбор данной платформы обусловлен её полной совместимостью с API Kubernetes при минимальных накладных расходах на ресурсы управляющего узла, что позволяет сфокусироваться на исследовании механизмов оркестрации stateful-приложений. Инфраструктурный слой стенда (Рисунок 1) включает в себя один управляющий узел (control-plane) и два рабочих узла (worker-nodes), что является минимально достаточным условием для проверки политик распределения подов (Pod Anti-Affinity) и отказоустойчивости кластера.

Центральным элементом архитектуры является шаблон развертывания, основанный на контроллере StatefulSet. В отличие от стандартных решений, данный шаблон реализует «жесткую» связку между логическим узлом кэша, его сетевым именем и дисковым пространством. Для обеспечения распределенного хранения данных в режиме L2-кэша используется связка из шести инстансов Redis, сконфигурированных в режиме Cluster (3 мастера и 3 реплики) [10, 11]. Сетевая связность обеспечивается через Headless Service, который создает в DNS кластера записи для каждого пода, позволяя узлам Redis взаимодействовать по стабильным именам (например, redis-cluster-0. redis-service), независимо от смены их внутренних IP-адресов.

Особое внимание в архитектуре уделено подсистеме хранения данных. Для каждого пода через механизм volumeClaimTemplates динамически резервируется персональный том Persistent Volume на базе локального хранилища (local-path storage). Это гарантирует, что при перезапуске любого узла кэша (вследствие аварии или планового обновления) данные оперативной памяти будут оперативно восстановлены из снимков на диске (RDB/AOF), предотвращая потерю состояния L2-слоя и исключая избыточную нагрузку на сеть кластера для полной ресинхронизации.

Для реализации отказоустойчивого шаблона был спроектирован стенд на базе облегченного дистрибутива Kubernetes — K3s (Рисунок 1).

На Рисунке 1 представлена спроектированная логическая архитектура стенда. В рамках кластера K3s развернуто три рабочих узла, на которых равномерно распределены шесть инстансов Redis. Мастер-узлы (Master) оперируют конкретными диапазонами хеш-слотов (от 0 до 16383), обеспечивая горизонтальное шардирование данных. Каждому мастеру соответствует реплика (Replica), расположенная на том же узле (согласно базовой конфигурации стенда), что позволяет верифицировать механизмы отказоустойчивости при сбое конкретных процессов. Важной особенностью является интеграция каждого пода с персональным томом Persistent Volume Claim (PVC), что гарантирует сохранность данных конкретного сегмента кэша при перезагрузке контейнеров.

Выбор технологического стека для реализации стенда обусловлен необходимостью моделирования реальных условий высоконагруженной среды при сохранении прозрачности управления. В качестве базовой инфраструктуры использован дистрибутив K3s, который, обладая полной совместимостью с API Kubernetes [12], оптимизирован для работы в условиях ограниченных ресурсов. Архитектура стенда предполагает разделение

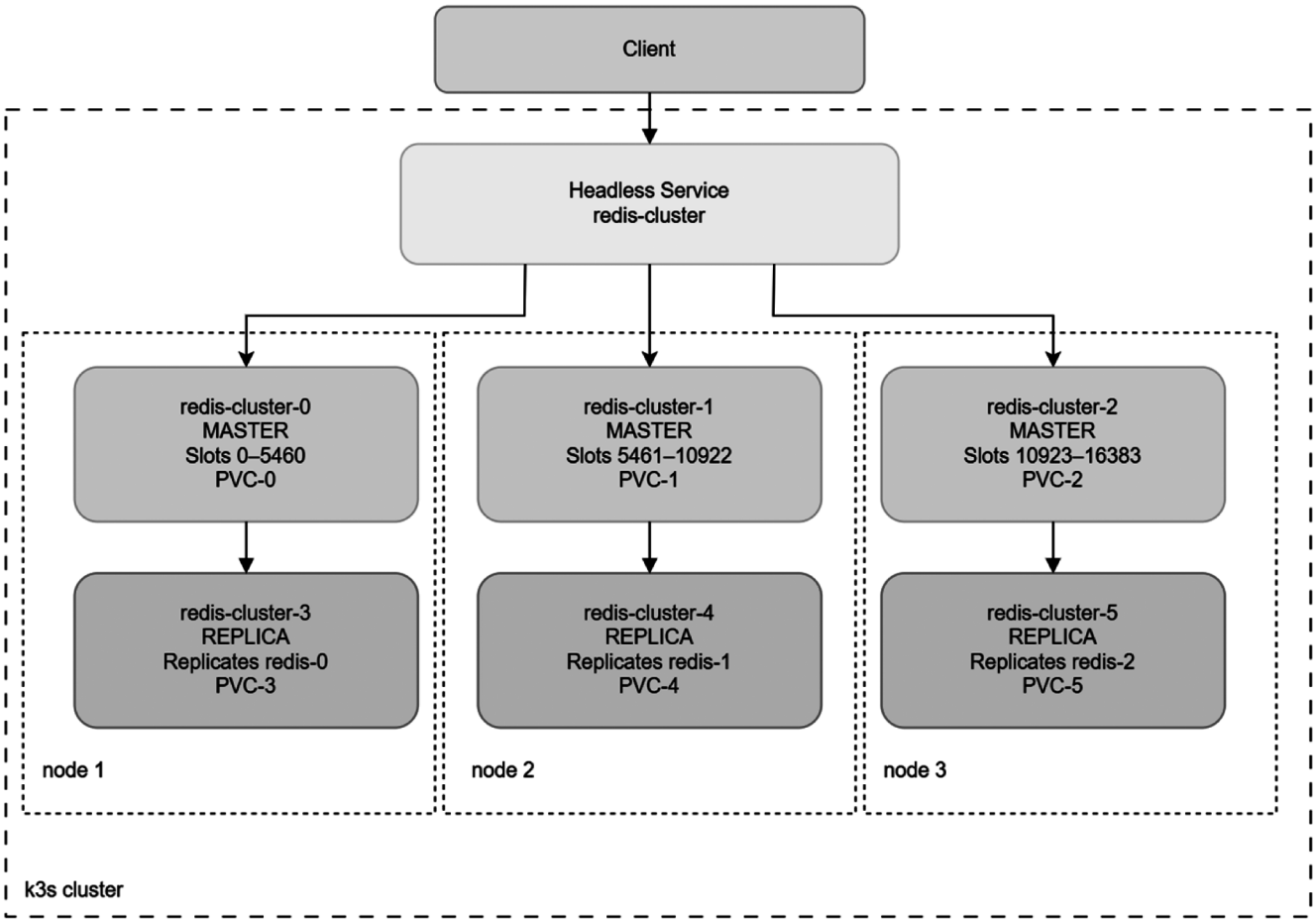

Рис. 1. Проектирование архитектуры и реализация стенда

ролей на управляющий узел (control-plane) и группу рабочих узлов (workers). Такое физическое или логическое разнесение критически важно для проверки политик Pod Anti-Affinity, которые гарантируют, что мастер-узел кэша и его соответствующая реплика не будут запущены на одном и том же физическом хосте, исключая единую точку отказа на уровне инфраструктуры [13]. Это позволяет минимизировать накладные расходы на управление кластером при сохранении полной совместимости с API Kubernetes (Рисунок 2).

Фундаментальной особенностью разработанного шаблона является использование контроллера StatefulSet в связке с Headless Service. В отличие от стандартных методов развертывания, данная связка обеспечивает каждому узлу Redis стабильный сетевой идентификатор (FQDN), который сохраняется на протяжении всего жизненного цикла кластера. Это решает проблему динамической адресации: в режиме Redis Cluster узлы должны обмениваться информацией о топологии и распределении хеш-слотов, используя фиксированные адреса [14]. Применение Headless-сервиса позволяет подам напрямую взаимодействовать друг с другом по именам вида redis-cluster-0. redis-service, что является необходимым условием для корректной работы механизмов автоматического переизбрания мастера (failover) без участия

```
user@k3s-master:~/redis-project$ kubectl get nodes
NAME          STATUS   ROLES                  AGE   VERSION
k3s-master    Ready    control-plane,master   26d   v1.33.6+k3s1
k3s-worker1   Ready    <none>                 26d   v1.33.6+k3s1
k3s-worker2   Ready    <none>                 26d   v1.33.6+k3s1
```

Рис. 2. Инфраструктурная база: кластер K3s в составе master-узла и двух рабочих узлов

```
user@k3s-master:~/redis-project$ kubectl get statefulset -n redis
NAME            READY   AGE
redis-cluster   6/6     10m
```

Рис. 3. Контроллер StatefulSet, управляющий шестью репликами Redis.

внешних балансировщиков. StatefulSet обеспечивает стабильные сетевые идентификаторы (redis-cluster-0, redis-cluster-1 и т.д.), упорядочивая развертывание и масштабирование. Связь каждого пода с персональным томом данных (Persistent Volume) закрепляется (Рисунок 3).

Управление состоянием данных в шаблоне реализовано через абстракцию VolumeClaimTemplates. Это позволяет динамически создавать индивидуальные тома Persistent Volume (PV) для каждого пода в кластере, что обеспечивает сохранность данных после сбоев или плановых обновлений (задача автоматического восстановления) (Рисунок 4) [15]. Теоретическая значимость такого подхода заключается в обеспечении «живучести» данных L2-кэша: при плановом обновлении образа (Rolling Update) или аварийном перезапуске пода, новый экземпляр автоматически монтирует тот же самый объем памяти (дисковый кэш), где сохранен файл конфигурации nodes.conf и снимки данных (AOF/RDB) [16]. Таким образом, процедура восстановления сокращается до времени старта процесса Redis, исключая длительный этап полной ресинхронизации данных по сети, что критически важно для минимизации задержек в системах с большими объемами кэшируемой информации.

Для zero-downtime операций кэш развернут в режиме Cluster. Данные шардируются между мастер-узлами, а каждый мастер имеет свою реплику (slave) (Рисунок 5).

Тестирование и практические результаты

Проверка работоспособности шаблона осуществляется путем комплексного тестирования, включающего имитацию плановых обновлений и аварийных сценариев. Основная цель — подтвердить сохранение доступности данных и работоспособности кластера в условиях ротации узлов.

На первом этапе проверялась стабильность кластера при выполнении команды kubectl rollout restart. Благодаря использованию StatefulSet, обновление узлов происходит последовательно. После перезапуска каждый под автоматически подхватывает файл конфигурации nodes.conf, что позволяет ему мгновенно восстановить свою роль в кластере без ручного вмешательства (Рисунок 6).

На Рисунке 6 видно, что каждый мастер-узел (master) имеет связанную реплику (slave), находящуюся на другом физическом или виртуальном узле (согласно affinity rules, настроенным в шаблоне).

Для обеспечения zero-downtime операций критически важна работа через Headless-сервис. В ходе тестирования было подтверждено, что узлы идентифицируют друг друга по полным доменным именам (FQDN), что гарантирует неизменность топологии при смене вну-

```
user@k3s-master:~/redis-project$ kubectl get pvc -n redis
NAME                        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
  VOLUMEATTRIBUTESCLASS   AGE
redis-data-redis-cluster-0   Bound    pvc-44c98e6e-2d2b-4466-85f4-d46b7c77e03a   2Gi        RWO            local-path
  <unset>                 9m41s
redis-data-redis-cluster-1   Bound    pvc-38530380-b425-4b1e-a2ee-d049949be899   2Gi        RWO            local-path
  <unset>                 9m41s
redis-data-redis-cluster-2   Bound    pvc-2b75088c-b789-4060-96a5-4504e6cfb8d6   2Gi        RWO            local-path
  <unset>                 9m41s
redis-data-redis-cluster-3   Bound    pvc-a48622ad-aa8c-408d-ab7a-33d8497d52cc   2Gi        RWO            local-path
  <unset>                 9m41s
redis-data-redis-cluster-4   Bound    pvc-48ae014b-9fb0-4e5a-8e5a-99e4e806653b   2Gi        RWO            local-path
  <unset>                 9m41s
redis-data-redis-cluster-5   Bound    pvc-8f61595e-675b-4449-9e7d-d085aec23428   2Gi        RWO            local-path
  <unset>                 9m41s
```

Рис. 4. Использование локальных томов (local-path) для хранения состояния каждого узла кэша

```
user@k3s-master:~/redis-project$ kubectl get pods -n redis
NAME                READY   STATUS    RESTARTS   AGE
redis-cluster-0     1/1     Running   0          9m4s
redis-cluster-1     1/1     Running   0          9m4s
redis-cluster-2     1/1     Running   0          9m4s
redis-cluster-3     1/1     Running   0          9m4s
redis-cluster-4     1/1     Running   0          9m4s
redis-cluster-5     1/1     Running   0          9m4s
```

Рис. 5. Состояние подов распределенного кэша: запущены 3 мастера и 3 реплики

```
user@k3s-master:~/redis-project$ kubectl -n redis exec redis-cluster-0 -- redis-cli cluster nodes
150089f33cf2bcce3bd74fd6f527cf5069d29c62 10.42.2.33:6379@16379 master - 0 1768336387483 2 connected 5461-10922
8e17deac27b22111fbc5f0b4bb84cb44ff8ec704 10.42.2.34:6379@16379 slave b64ffb44f3068a38df269fff6db6e32ac3b7844e 0 17683363
86581 3 connected
596f04b01abcda81efcf0691d8f4fbf97444b93d 10.42.0.36:6379@16379 slave fee2cc5968a66f0093d8cfc45a6e7d8fe12f79a2 0 17683363
87986 1 connected
fee2cc5968a66f0093d8cfc45a6e7d8fe12f79a2 10.42.1.35:6379@16379 myself,master - 0 1768336387000 1 connected 0-5460
b64ffb44f3068a38df269fff6db6e32ac3b7844e 10.42.0.35:6379@16379 master - 0 1768336386480 3 connected 10923-16383
e66f02e39c88d69a04924c5dd97e0d9f7e3d12fd 10.42.1.37:6379@16379 slave 150089f33cf2bcce3bd74fd6f527cf5069d29c62 0 17683363
88086 2 connected
```

Рис. 6. Топология кластера после восстановления

тренних IP-адресов подов. Имена вида redis-cluster-0. redis-cluster.redis.svc.cluster. local обеспечивают надежную адресацию внутри K8s. (Рисунок 7).

Функционирование системы как кэша второго уровня (L2) предполагает корректное шардирование и доступность данных с любого узла кластера. Для проверки в узел redis-cluster-0 были внесены тестовые записи. Система автоматически перенаправляет запрос на соответствующий мастер-узел, отвечающий за конкретный хеш-слот (Рисунок 8).

Далее была проведена проверка доступности данных через другой узел кластера (например, redis-cluster-3), что подтвердило успешную репликацию и целостность распределенной базы данных [17]. Данные, записанные на одном узле, успешно извлекаются через другой, что подтверждает корректность настройки кэширующего слоя. (Рисунок 9).

Ключевым тестом стала имитация аварии пода с целью проверки работы механизмов управления состоянием. Был принудительно удален под redis-cluster-0. В момент его отсутствия кластер сохранил работоспособность, зафиксировав временную недоступность одной из нод. Узлы фиксируют статус 'fail' для отсутствующего сегмента, переключая обслуживание на реплики (Рисунок 10).

Заключительный этап подтвердил эффективность использования Persistent Volume. После автоматического перезапуска под redis-cluster-0 подключился к ранее выделенному тому данных. Проверка показала, что данные, записанные до удаления пода, остались доступны. После «холодного» перезапуска данные кэша восстановлены из локального тома, что исключает необходимость прогрева кэша и снижает нагрузку на БД (Рисунок 11).

Таким образом, в ходе проведенного исследования был успешно спроектирован, реализован и экспериментально апробирован универсальный архитектурный шаблон развертывания распределенных кэшей второго уровня в среде Kubernetes, который эффективно устраняет критическую проблему сохранения состояния (state) в динамических кластерных системах [18]. На основе детального анализа практической реализации кластера Redis в среде K3s доказано, что использование контроллера StatefulSet в сочетании с механизмами динамического резервирования томов (PVC) на базе локальных хранилищ и Headless-сервисов позволяет полностью нивелировать фундаментальные ограничения стандартных средств оркестрации, такие как эфемерность данных и волатильность сетевых идентификаторов. В рамках верификации разработанного решения были проведены расширенные тесты на целостность данных и отказоустойчивость, которые продемонстрировали,

Рис. 7. Работа кластера с использованием стабильных сетевых идентификаторов

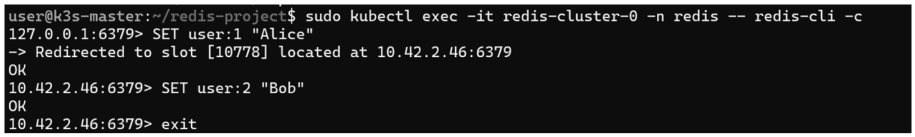

Рис. 8. Операция записи данных в кластер

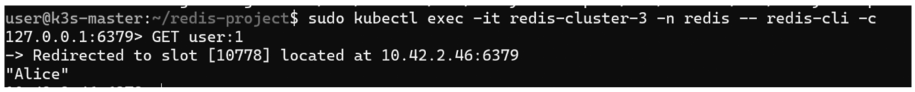

Рис. 9. Проверка межрегионального чтения

Рис. 10. Состояние кластера в момент сбоя

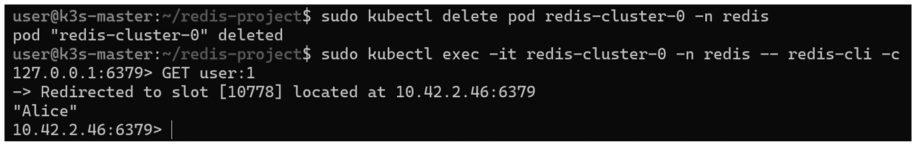

Рис. 11. Результат автоматического восстановления

что записи, внесенные до преднамеренного аварийного завершения работы узлов, успешно сохраняются и остаются доступными для чтения с произвольных сегментов кластера благодаря стабильным FQDN-адресам и детерминированной привязке томов данных к конкретным инстансам подов. Результаты экспериментов подтверждают, что предложенный подход обеспечивает выполнение плановых операций обновления (Rolling Update) по принципу zero-downtime, гарантируя не только непрерывность сетевого доступа, но и мгновенную доступность «горячих» данных сразу после перезапуска узла без необходимости их повторной репликации по сети. Ключевым выводом работы является подтверждение гипотезы о том, что интеграция механизмов управления состоянием на уровне оркестратора позволяет минимизировать показатель времени восстановления (MTTR) и эффективно предотвращать возникновение «кэш-штормов» (cache miss storm) на нижележащую инфраструктуру хранения данных. Разработанный шаблон представляет собой завершенный технологический артефакт, готовый к внедрению в эксплуатационные циклы высоконагруженных распределенных приложений с целью повышения их отказоустойчивости (SLA) и оптимизации задержек в облачных средах.

ЛИТЕРАТУРА

1. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка / М. Клеппман. — СПб.: Питер, 2018. — 688 с.
2. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга / К. Ричардсон. — СПб.: Питер, 2019. — 544 с.
3. Лукша А.С. Kubernetes в действии / А.С. Лукша; пер. с англ. А.Н. Киселева. — М.: ДМК Пресс, 2018. — 672 с.
4. Найгард М. Release It! Проектирование и дизайн ПО для тех, кому не всё равно / М. Найгард. — СПб.: Питер, 2019. — 464 с.
5. Бёрнс Б. Kubernetes: лучшие практики / Б. Бёрнс, Э. Вильяльба, Д. Стрейбел, Л. Эвенсон. — СПб.: Питер, 2020. — 272 с.
6. Осипов Г.С. Методы и средства обеспечения высокой доступности распределенных систем / Г.С. Осипов // Информационные технологии. — 2021. — № 4. — С. 12–18.
7. Сейфан Д. Мастерство Kubernetes / Д. Сейфан. — СПб.: Питер, 2019. — 416 с.
8. StatefulSets [Электронный ресурс] // Kubernetes Documentation. — URL: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/ (дата обращения: 04.01.2026).
9. Zero-downtime Deployment Patterns [Электронный ресурс] // Google Cloud Architecture Center. — URL: https://cloud.google.com/architecture/deployment-strategies (дата обращения: 04.01.2026).
10. Хайневер Р. Redis в действии / Р. Хайневер; пер. с англ. А.Н. Киселева. — М.: ДМК Пресс, 2014. — 312 с.
11. Redis Cluster Specification [Электронный ресурс] // Redis.io. — URL: https://redis.io/docs/reference/cluster-spec/ (дата обращения: 15.01.2026).
12. K3s Documentation: Lightweight Kubernetes [Электронный ресурс] // K3s.io. — URL: https://docs.k3s.io/ (дата обращения: 04.01.2026).
13. Garrick R. High Availability in Kubernetes: A Practical Guide / R. Garrick // Cloud Native Computing Foundation. — 2022.
14. Harrison G. Next Generation Databases: NoSQL, NewSQL, and Big Data / G. Harrison. — Apress, 2015. — 240 p.
15. Persistent Volumes [Электронный ресурс] // Kubernetes Documentation. — URL: https://kubernetes.io/docs/concepts/storage/persistent-volumes/ (дата обращения: 04.01.2026).
16. Container Storage Interface (CSI) for Kubernetes GA [Электронный ресурс] // Kubernetes Blog. — URL: https://kubernetes.io/blog/2019/01/15/container-storage-interface-csi-for-kubernetes-ga/ (дата обращения: 04.01.2026).
17. Карлсон Д. Redis в облаке: масштабирование и отказоустойчивость / Д. Карлсон. — М.: Рипол-Классик, 2017. — 256 с.
18. Черняк Л. Контейнеры и оркестровка: новые подходы к инфраструктуре / Л. Черняк // Открытые системы. СУБД. — 2020. — № 2. — С. 34–40.