Архитектура

1. Архитектура и подход

Для реализации системы хранения на базе Longhorn разработаны два ключевых компонента:

- `nova-apps-operator`: оператор, управляющий полным жизненным циклом кластера Longhorn. Он выполняет следующие функции:
  - Установка: автоматическое развертывание всех необходимых компонентов Longhorn в кластере.
  - Удаление: безопасное и полное удаление Longhorn и связанных ресурсов.
  - Масштабирование: добавление или удаление узлов для адаптации к изменяющимся требованиям хранения.
  - Изменение конфигурации: возможность обновления параметров кластера Longhorn, таких как Storage-пулы и клиентские узлы, с учетом текущих потребностей.
- `nova-storage-agent`: агент, запускаемый на каждом узле кластера. Он взаимодействует с API Kubernetes для управления блочными устройствами и осуществляет конфигурацию LVM в соответствии с заданными параметрами. Поддерживаемый функционал включает:
  - Создание LVM (Logical Volume Manager): автоматическое создание логических томов.
  - Удаление LVM: безопасное удаление ненужных томов.
  - Изменение размера LVM: адаптивная настройка размеров томов в зависимости от потребностей хранения.

1.1. Автоматизация

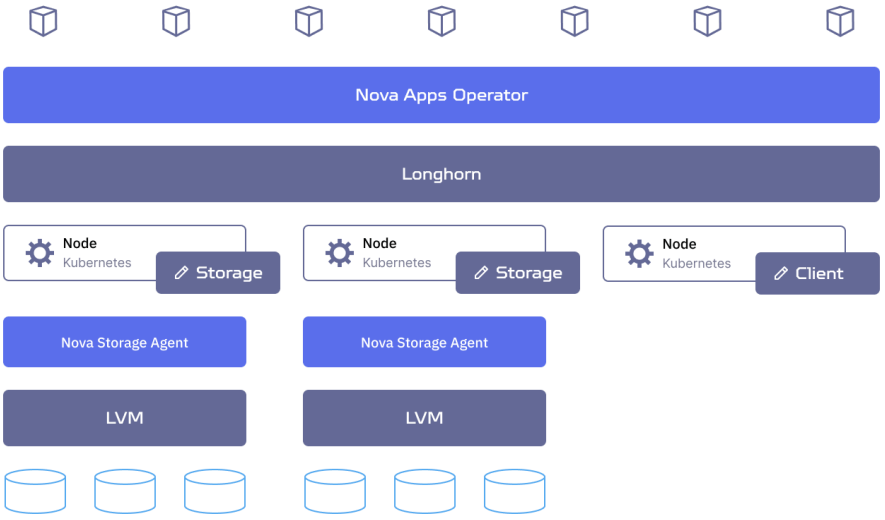
Компоненты `nova-apps-operator` и `nova-storage-agent` устанавливаются автоматически при развертывании кластера Nova. Настройка конфигурации локальных дисков, используемых Longhorn, не требует ручного вмешательства. Все задачи по управлению дисками и LVM выполняются встроенными сервисами

1.2. Архитектурное разделение узлов

Решение включает четкое разделение узлов на две категории:

- Client-узлы: предназначены для запуска сервисов, требующих постоянного хранилища (`Persistent Volume`). На этих узлах развёрнут весь набор компонентов Longhorn, но отключена возможность выделения дисков для хранения. Они работают как клиенты для получения данных из Longhorn.
- Storage-узлы: используются для конфигурации дисков, предоставляющих ресурсы `Persistent Volume` в Longhorn. С этих узлов происходит выделение пространства для хранилища. При необходимости storage-узлы также могут выполнять функции client-узлов, запуская пользовательские сервисы для более эффективного использования ресурсов.

На данный момент поддерживаются только диски v1 в режиме `filesystem`, однако создание дисков с другой конфигурацией не ограничивается.



2. Принцип работы

После установки платформы в систему конфигурируется и запускается `nova-storage-agent`, который играет ключевую роль в управлении хранилищем. Агент имеет строго ограниченные права доступа к API Kubernetes, что обеспечивает безопасность и изоляцию выполняемых операций. Для взаимодействия с кластером Kubernetes используется специально сгенерированный KUBECONFIG, предоставляющий агенту только необходимые разрешения.

Вся конфигурация `nova-storage-agent` хранится в директории `/etc/nova-storage-agent/`:

- `config.env` - основной конфигурационный файл;
- `nova-storage-agent.conf` - KUBECONFIG;
- `pki/` - директория для хрнения сертификатов.

`nova-apps-operator` устанавливается автоматически, начиная с версии 5.1.0. После его установки в namespace `nova-apps-operator` создается deployment `nova-apps-operator-controller-manager`, который выполняет все необходимые настройки для интеграции `nova-apps-operator` с `nova-storage-agent`.

Хотя установка Longhorn выполняется с использованием `nova-apps-operator`, общий архитектурный подход к управлению модулями остается неизменным. Конфигурационные данные для Longhorn хранятся в репозитории Gitea, что позволяет централизованно управлять состоянием хранилища. Оператор динамически обновляет **kustomization** в зависимости от пользовательских параметров, внося изменения перед их применением и обеспечивая автоматическую настройку всех компонентов.

Процесс выглядит следующим образом:

1. Пользователь создает конфигурацию **NodeVolumeConfig**, которая определяет параметры томов и блочных устройств, необходимые для пула хранения.
2. Затем пользователь определяет и передает конфигурацию **LonghornClusterConfig**, включая настройки Storage-пулов и клиентских узлов.
3. `nova-apps-operator` применяет изменения в **kustomization**, обновляя конфигурацию в соответствии с полученными параметрами.
4. Оператор устанавливает обновленную **kustomization** в кластер, интегрируя компоненты с Longhorn.
5. Система **Flux**, отслеживая изменения в репозитории **Gitea**, автоматически подтягивает и применяет все манифесты, поддерживая синхронизацию состояния платформы с репозиторием.

Таким образом, весь процесс установки и конфигурации Longhorn автоматизирован. Оператор берет на себя выполнение сложных операций, упрощая управление хранилищем и обеспечивая согласованность конфигурации с репозиторием.

Как упоминалось ранее, также были реализованы инструменты для упрощения работы с Longhorn, что позволяет управлять кластером стандартными методами:

- UI - после создания кластера Longhor, будет создан **ingress** `nova-longhorn-ui.<dnsBaseDomain>`
- Kubernetes **Custom resources**

3. Функциональность

3.1. LonghornClusterConfig

LonghornClusterConfig предоставляет гибкие возможности для управления кластером Longhorn с использованием манифестов Kubernetes. Основные функции включают:

- **Создание кластера Longhorn:** Автоматическая установка и конфигурация всех необходимых компонентов Longhorn, включая развертывание служб управления хранилищем и агентов узлов.
- **Удаление кластера Longhorn:** Полное удаление всех ресурсов Longhorn из кластера Kubernetes с корректной очисткой и деинициализацией связанных объектов.
- Изменение конфигурации кластера:
  - **Добавление и удаление клиентских узлов (clientNodes):** Гибкое управление клиентскими узлами для увеличения или уменьшения доступных вычислительных ресурсов, которые подключаются к Longhorn для использования хранилища.
  - **Добавление и удаление StoragePool:** Динамическое управление пулами хранения, включая добавление новых пулов для расширения пространства хранения или удаление ненужных пулов, с возможностью перераспределения данных для минимизации простоя.

3.2. NodeVolumeConfig

NodeVolumeConfig управляет блочными устройствами и их конфигурацией, обеспечивая эффективное использование LVM для Longhorn. Основные функции, предоставляемые данным компонентом, включают:

- **Создание LVM устройств:** Автоматическое создание логических томов с учётом спецификаций для работы с Longhorn, включая конфигурацию устройств и файловых систем.
- **Удаление LVM устройств:** Удаление ненужных LVM устройств с соблюдением процедур безопасности данных и очисткой соответствующих записей.
- **Увеличение размера LVM устройства:** Расширение существующих LVM томов для увеличения доступного пространства без прерывания работы сервисов, с обновлением файловой системы для использования нового объёма.
- **Актуализация fstab:** Обновление таблицы монтирования (fstab) для обеспечения надёжного монтирования LVM устройств при перезагрузке системы, что гарантирует доступность хранилища Longhorn.
- **Управление директориями для монтирования LVM устройств:** Создание, обновление и удаление точек монтирования для LVM устройств, оптимизированных для использования с Longhorn, с автоматическим управлением конфигурацией и синхронизацией с настройками Longhorn.

После установки кластера Longhorn автоматически создается стандартный **StorageClass** `longhorn-storage-single-replica`. Следует отметить, что этот StorageClass создает

тома без репликации. Если требуется создать **Persistent Volumes**, которые будут работать в режиме высокой доступности, а также использовать нестадарнтные параметры, необходимо создать собственный **StorageClass**.

Системные требования

1. Аппаратные ресурсы

Требования по вычислительным ресурсам можно найти в статье

2. Сетевые требования

Сеть между узлами, участвующими в кластере Longhorn, должна обеспечивать пропускную способность не менее 1 Гбит/с.

3. Функциональные требования

Узлы	**CPU**
Client-узлы	Количество данных узлов не ограничено.
Блочные устройства для Storage-узлов	1. Количество блочных устройств в одной **Volume group** устанавливается самим LVM2. В нашей конфигурации мы отключаем все лимиты. 2. Для создания **StoragePool** на узле требуется наличие блочного устройства, которое: 2.1. Не должно иметь файловой системы (неразмеченное устройство). 2.2. Будет использовано для создания **Physical Volume** (PV), **Volume Group** (VG) и LVM устройств.
mountPath	1. Указанный путь для монтирования LVM устройства должен быть либо пустой директорией, либо полностью отсутствовать. 2. Любое содержимое в указанной директории приведет к сбою операции.
Поддерживаемые файловые системы	xfs ext4

4. Рекомендации по конфигурации:

Конфигурация	**Топология**	**Особенности использования**

Конфигурация для окружений разработки (dev)	Минимальная конфигурация с 1 узлом, совмещающим роли client и storage. Это позволяет снизить затраты на инфраструктуру при сохранении базовой функциональности.	Конфигурация предназначена для разработки, в связи с чем можно пренебречь отказоустойчивостью, а также использовать менее мощные ресурсы.
Конфигурация для продуктивных окружений (prod)	Разделение client и storage узлов. Минимум 3 storage-узла для обеспечения отказоустойчивости данных.	Конфигурация предназначена для работы в продуктивных окружениях, где важно обеспечить высокую отказоустойчивость. В данной архитектуре пользовательская нагрузка и хранилище данных разделены, что предотвращает взаимное влияние и обеспечивает стабильную работу системы даже при повышенных нагрузках.

Пользовательские сценарии

Для того, чтобы иметь возможность подключать *Persistent Volumes* из хранилища Longhorn, необходимо добавить узлы, на которых будут запускаться пользовательские Pod-ы, в список `clientNodes` в `LonghornClusterConfig` или сделать их частью storagePool.

На узлах, перечисленных в списке `clientNodes`, не должно быть никаких *Taints*.

После успешного применения конфигурации нужно создать *Persistent Volumes* на хранилище Longhorn.

1. Создание тома с доступом RWO и его подключение к приложению

1. Создайте том.

   Пример манифеста Kubernetes для создания тома (Persistent Volume Claim (PVC)):

   ```yaml
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: my-rwo-volume
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 1Gi
     storageClassName: longhorn-storage-single-replica
   ```

2. Подключите созданный выше PVC к приложению. Пример манифеста для Deployment:

   ```yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: app-rwo-1
   spec:
     replicas: 1
     selector:
       matchLabels:
         app: app-rwo-1
     template:
       metadata:
   ```

```
      labels:
        app: app-rwo-1
    spec:
      containers:
      - name: app-container
        image: nginx
        volumeMounts:
        - mountPath: "/data"
          name: data-volume
      volumes:
      - name: data-volume
        persistentVolumeClaim:
          claimName: my-rwo-volume
```

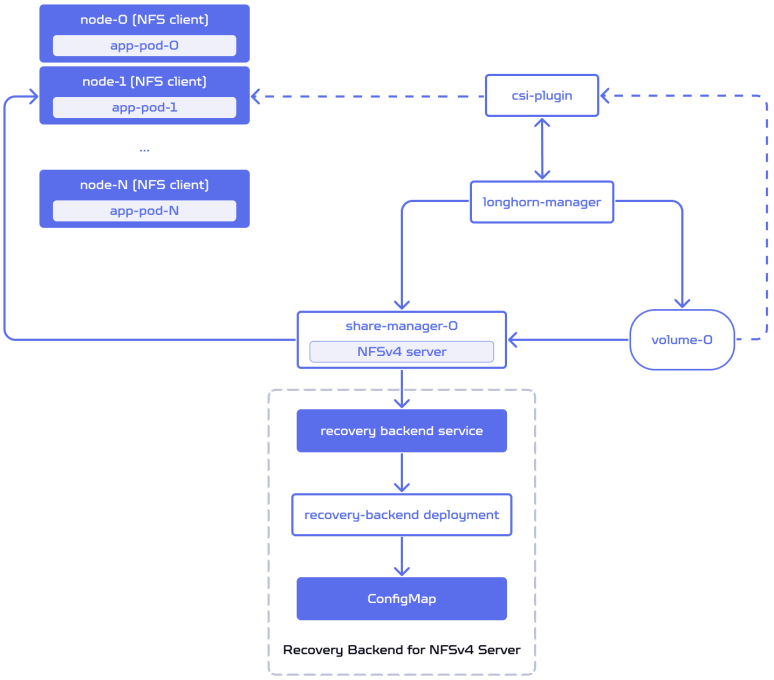
Пример StatefulSet, у которого каждый Pod будет использовать отдельный том. Для этого не требуется создавать отдельные PVC, так как это обеспечивается механизмом *VolumeClaimTemplates*.

YAML

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: app-rwo
spec:
  selector:
    matchLabels:
      app: app-rwo
  serviceName: "app-rwo"
  replicas: 3
  template:
    metadata:
      labels:
        app: app-rwo
    spec:
      containers:
      - name: app-container
        image: nginx
        volumeMounts:
        - name: data-volume
          mountPath: "/data"
  volumeClaimTemplates:
  - metadata:
      name: data-volume
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi
      storageClassName: longhorn-storage-single-replica
```

2. Создание тома с доступом RWX и подключение к приложению

Конфигурация Longhorn поддерживает автоматическое создание RWX томов, используя NFS. При создании *Persistent Volumes* в режиме RWX будет создан Pod с share-manager, в который будет подключен том с Longhorn, а сам он будет выступать в роли NFS-сервера.



1. Создайте том.

   Пример манифеста Kubernetes для создания тома:

   ```yaml
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: my-rwx-volume
   spec:
     accessModes:
       - ReadWriteMany
     resources:
       requests:
         storage: 1Gi
     storageClassName: longhorn-storage-single-replica
   ```

2. Подключите том к приложению.

   Том RWX может быть подключен одновременно к нескольким Pod’ам. Пример Deployment, использующего RWX:

YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-rwx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: app-rwx
  template:
    metadata:
      labels:
        app: app-rwx
    spec:
      containers:
      - name: app-container
        image: nginx
        volumeMounts:
        - mountPath: "/shared-data"
          name: data-volume
      volumes:
      - name: data-volume
        persistentVolumeClaim:
          claimName: my-rwx-volume
```