

# Содержание

|          |                                     |          |
|----------|-------------------------------------|----------|
| <b>1</b> | <b>Описание системы</b>             | <b>2</b> |
| 1.1      | Слой . . . . .                      | 2        |
| 1.2      | Сервис перемещения данных . . . . . | 2        |
| <b>2</b> | <b>Реализация</b>                   | <b>3</b> |
| 2.1      | Кластер Kubernetes . . . . .        | 3        |
| 2.2      | Запуск Dcache на кластере . . . . . | 4        |
| 2.2.1    | Вспомогательные сервисы . . . . .   | 4        |
| 2.2.2    | Чарт Dcache . . . . .               | 6        |
| 2.3      | Сервис перемещения . . . . .        | 6        |
| 2.3.1    | Использование . . . . .             | 6        |
| 2.3.2    | API . . . . .                       | 6        |
| <b>3</b> | <b>Dcache + docker compose</b>      | <b>8</b> |

# 1 Описание системы

## 1.1 Слой

Каждый слой представляет из себя кластер Kubernetes из нескольких машин. Одна из машин выступает как контролер (control plane) и другие как исполнители (worker). После создания кластера посредством kubectl, требуется установить приложения, необходимые для работы системы Dcache. И после этого разворачивается система Dcache состоящая из двух составных частей: двери, обеспечивающей доступ к системе, и пула для хранения информации.

Также для обеспечения коммуникации внутри кластера Kubernetes и возможности доступа снаружи (для сервиса перемещения), требуется создание сервисов перенаправления портов.

После того как система запущена, требуется выбрать протокол коммуникации, который реализован в сервисе перемещения, в данном примере используется протокол webdav.

## 1.2 Сервис перемещения данных

Для того чтобы организовать перемещение данных между двумя системами Dcache на двух слоях, описанных выше, требуется средство способное использовать либо API системы хранения либо операции: записи, чтения и удаления. Для общности примера выбран второй вариант, так как при наличии подходящих протоколов, есть возможность использовать любую другую систему хранения.

Для реализации использовался язык программирования python. Так как вся система ориентируется на облачные технологии выбран стек, позволяющий реализовать микросервисную архитектуру. В данном примере Backend и методы записи с помощью протоколов реализованы в одном приложении, но в практической реализации следует их разделить.

Серверная часть предоставляется библиотекой flask, а реализация коммуникации через протокол webdav сделана через библиотеку requests.

Для данного прототипа реализован минимальный набор API:

- upload [POST] - загружает файл в хранилище с номером 1, а также записывает метаданные в свое локальное хранилище.
- move [POST] - перемещает файл с указанным в запросе id, в хранилище с указанным номером.
- files [GET] - возвращает список, всех файлов в формате json с описаниями места хранения и пути для получения файла.

Для загрузки файла пользователь напрямую обращается к нужному хранилищу посредством протокола webdav.

Также важно отметить, что сервису необходимо знать о портах доступных для записи поддерживаемым протоколом, которые организуются в кластере посредством сервисов перенаправления портов, описанных выше.

## 2 Реализация

### 2.1 Кластер Kubernetes

Наиболее важным источником информации в данном вопросе является: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/> - официальная документация kubernetes. Также кластер можно разворачивать любыми средствами. В данном примере используется **kubeadm**. Для того чтобы развернуть систему через kubeadm нужно:

- **Ознакомится с документацией инструмента развертывания**
- Подготовить средство запуска контейнеров containerd или docker  
Для установки containerd следует пользоваться руководством: <https://github.com/containerd/containerd/blob/main/docs/getting-started.md>  
Также следует сделать настройку cgroup описанную <https://kubernetes.io/docs/setup/production-environment/container-runtimes/#containerd>.
- Отключить swap и установить инструменты kubectl и kubelet  
Инструкции здесь <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>
- Выбрать сетевой аддон  
Список аддонов тут <https://kubernetes.io/docs/concepts/cluster-administration/addons/#networking-and-network-policy>. В данной реализации используется flannel. Для его установки следует следовать гайду <https://github.com/flannel-io/flannel#deploying-flannel-manually>. Важно отметить что нужно указать podCidr при инициализации кластера (Подробно по ссылке).
- Сеть  
Также может потребоваться отключение фаерволла. В almalinux: `sudo systemctl stop firewalld`. Либо открыть все необходимые порты?
- Создать кластер через команды `kubeadm init` и `kubeadm join`.

После создания кластера команды `kubectl get nodes` и `kubectl get pods` должны работать и не выдавать ошибки. Иначе следует пересоздать кластер и следовать инструкции (более тщательно?). Пример работы:

```
[root@anmstor02 ~]# kubectl get nodes -o wide
```

| NAME              | STATUS | ROLES         | AGE | VERSION | INTERNAL-IP   | EXTERNAL-IP | OS-IMAGE                       | KERNEL-VERSION               | CONTAINER-RUNTIME  |
|-------------------|--------|---------------|-----|---------|---------------|-------------|--------------------------------|------------------------------|--------------------|
| anmstor02.jinr.ru | Ready  | control-plane | 20d | v1.32.3 | 10.220.18.166 | <none>      | AlmaLinux 9.4 (Seafoam Ocelot) | 5.14.0-427.22.1.el9_4.x86_64 | containerd://2.0.4 |
| anmstor03.jinr.ru | Ready  | <none>        | 20d | v1.32.3 | 10.220.18.167 | <none>      | AlmaLinux 9.4 (Seafoam Ocelot) | 5.14.0-427.22.1.el9_4.x86_64 | containerd://2.0.4 |

```
[root@anmstor02 ~]# kubectl get pods -o wide
```

| NAME                       | READY | STATUS  | RESTARTS | AGE | IP           | NODE              | NOMINATED NODE | READINESS GATES |
|----------------------------|-------|---------|----------|-----|--------------|-------------------|----------------|-----------------|
| dc-dcache-door-0           | 1/1   | Running | 0        | 13d | 10.244.1.170 | anmstor03.jinr.ru | <none>         | <none>          |
| dc-pool-a-0                | 1/1   | Running | 0        | 13d | 10.244.1.171 | anmstor03.jinr.ru | <none>         | <none>          |
| kafka-0                    | 1/1   | Running | 0        | 20d | 10.244.1.4   | anmstor03.jinr.ru | <none>         | <none>          |
| post-69d75fd5c7-bjbvb      | 1/1   | Running | 0        | 13d | 10.244.1.166 | anmstor03.jinr.ru | <none>         | <none>          |
| zookeeper-5db6799b45-6swk4 | 1/1   | Running | 0        | 20d | 10.244.1.2   | anmstor03.jinr.ru | <none>         | <none>          |

Рис. 1: Тест кластера

Также следует понимать для организации перемещения **потребуется несколько кластеров dcache** В данном примере используются два отдельных кластера kubernetes на разных машинах, но данное решение можно реализовать и в разных пространствах имен одного кластера (потребуется перенаправить сервисами порты).

## 2.2 Запуск Dcache на кластере

Репозиторий с кодом <https://github.com/sxvxxmx/dcachelayer>.

### 2.2.1 Вспомогательные сервисы

После того как **кластер развернут** необходимо подготовить приложения для запуска dcache а именно:

- PostgreSQL с пользователем chimera для хранения метаданных
- Zookeeper для коммуникации
- Kafka для обработки сообщений
- Volume Claim для выдачи дискового пространства под pool. В данном примере используется локальное хранение.
- Сервисы доступа к frontend dcache и протоколам (webdav).

Для развертывания следует использовать команду `kubectl apply -f [file path]`.

1. Для развертывания PostgreSQL требуется подготовленный контейнер, который уже загружен в docker hub. Или собрать свой (dockerfile в dapp/party/postgres). После этого требуется развернуть сервис и саму базу данных dapp/party/postgres/post.yaml. Сервис имеет название chimera-postgresql, которое определено в dcache чарте.
2. Zookeeper и его сервис разворачиваются посредством dapp/party/zoo.yaml. Сервис также имеет название определенное в системе dcache.

3. Для Kafka используется файл `dapp/party/kafka.yaml`. Аналогично Zookeeper.
4. Для локального хранения файлов:  
Установить и развернуть на кластере необходимые поды:

```
kubectl apply -f https://raw.githubusercontent.com/rancher/
local-path-provisioner/master/deploy/local-path-storage.yaml
```

После этого сделаем локальный путь по умолчанию:

```
kubectl patch storageclass local-path -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

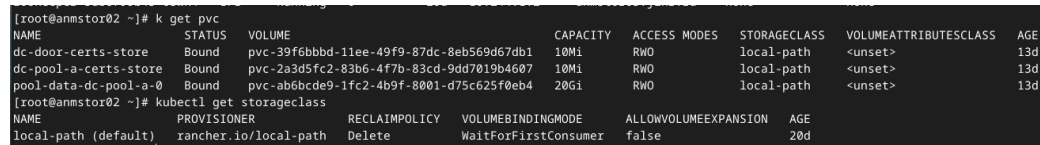
```
kubectl get pod -n local-path-storage
```

Для проверки запустились ли нужные поды.

```
kubectl get storageclass
```

Проверка того что используется локальный путь.

После этого шага при установке чарта `dcache` автоматически занимается пространством, а также показано что установлен локальный путь по умолчанию:



```
[root@anmstor02 ~]# k get pvc
NAME                                STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
dc-door-certs-store                Bound    pvc-39f6bbbd-11ee-49f9-87dc-8eb569d67db1   10Mi       RWO            local-path     <unset>                  13d
dc-pool-a-certs-store              Bound    pvc-2a3d5fc2-83b6-4f7b-83cd-9dd7019b4607   10Mi       RWO            local-path     <unset>                  13d
pool-data-dc-pool-a-0             Bound    pvc-ab6bcde9-1fc2-4b9f-8001-d75c625f0eb4   20Gi       RWO            local-path     <unset>                  13d
[root@anmstor02 ~]# kubectl get storageclass
NAME                                PROVISIONER   RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
local-path (default)              rancher.io/local-path   Delete          WaitForFirstConsumer   false                  20d
```

Рис. 2: Тест локального выделения пространства

Важно понимать что для полного удаления файлов требуется также перезапуск приложения с базой метаданных, а не только удаление локальных хранилищ.

5. Также для доступа к `dcache` из внешних источников (для сервиса перемещения) требуется создать сервис, перенаправляющий порт: для этого применяется файл `dapp/party/webdav.yaml` и `dapp/party/front.yaml` по аналогии можно подключить необходимый сервис из `dcache` вроде `admin` панели.

## 2.2.2 Чарт Dcache

Чарт `dcache` расположен `dapp/dcachelhelm` и является модификацией <https://github.com/dCache/dcachelhelm>. Данный чарт создает `pool-a` для записи и один сервис и приложение `door`. Для установки чарта потребуется `helm`. Команда для установки здесь <https://helm.sh/docs/intro/install/>. После того как все сервисы запущены (из прошлой секции) применяется команда:

```
helm install [name] dapp/dcache-helm/
```

После чего начинается разворачивание необходимых сервисов и приложений. При первом запуске также требуется загрузить контейнеры, что требует времени. На рисунке 1 показан пример запущенной системы готовой к работе.

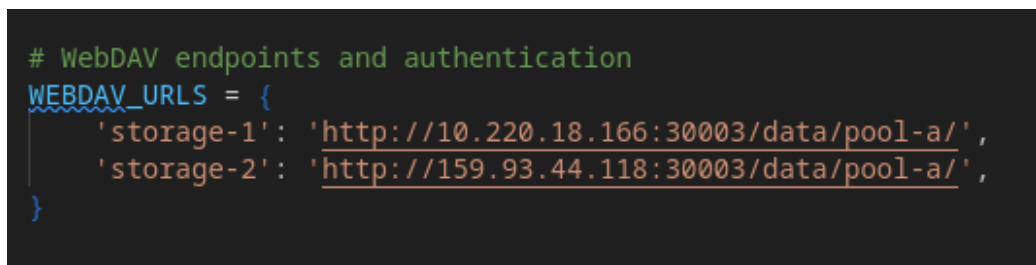
Конфигурация dcache находится в `dapp/dcache-helm/templates/configmap.yaml`. А также `dapp/dcache-helm/values.yaml` – файл позволяющий менять конфигурацию самого чарта.

## 2.3 Сервис перемещения

Сервис перемещения – микросервис написанный на python и flask (`dapp/observ`). Наиболее простым способом запуска является локальный запуск python приложения через `main` (`python main.py`) после установки зависимостей из `requirements.txt`. Также в директории находится `Dockerfile`, позволяющий собрать сервис в контейнере.

### 2.3.1 Использование

Для того чтобы добавить уровни хранения в сервис требуется изменить конфигурацию в файле `dapp/observ/control.py`. А именно блок `WEBDAV_URLS`. Для добавления новых уровней достаточно указать новое хранилище а именно его название и путь на pool в который следует записывать файлы.



```
# WebDAV endpoints and authentication
WEBDAV_URLS = {
    'storage-1': 'http://10.220.18.166:30003/data/pool-a/',
    'storage-2': 'http://159.93.44.118:30003/data/pool-a/',
}
```

Рис. 3: Поле ответственное за добавление хранилищ

### 2.3.2 API

Для взаимодействия сервис имеет набор из 3х эндпоинтов:

- `/upload` - загрузка файла через post
- `/move` - перемещение файла с указанием id файла и нового хранилища
- `/files` - список всех файлов на хранилищах

```

• .venvmchin@m15:~/Documents/projects/jinr$ curl -X POST -F "file=@file1.txt"
http://localhost:6000/upload
{
  "file_id": 1,
  "message": "File uploaded successfully"
}
• .venvmchin@m15:~/Documents/projects/jinr$ curl -X POST -F "file=@file2.txt"
http://localhost:6000/upload
{
  "file_id": 2,
  "message": "File uploaded successfully"
}
• .venvmchin@m15:~/Documents/projects/jinr$ curl -X POST -F "file=@file3.txt"
http://localhost:6000/upload
{
  "file_id": 3,
  "message": "File uploaded successfully"
}
• .venvmchin@m15:~/Documents/projects/jinr$ curl -X POST -H "Content-Type: app
lication/json" -d '{"file_id": 1, "target_storage": "storage-2"}' http://loc
alhost:6000/move
{
  "message": "File moved successfully",
  "path": "file1.txt"
}

```

Рис. 4: Запись и перемещение

```

• .venvmchin@m15:~/Documents/projects/jinr$ curl -X GET http://localhost:6000/files
[
  {
    "id": 3,
    "path": "file3.txt",
    "size": 14,
    "storage": "storage-1",
    "upload_time": "2025-04-04T07:23:00.894738"
  },
  {
    "id": 2,
    "path": "file2.txt",
    "size": 28,
    "storage": "storage-1",
    "upload_time": "2025-04-04T07:22:55.112442"
  },
  {
    "id": 1,
    "path": "file1.txt",
    "size": 11,
    "storage": "storage-2",
    "upload_time": "2025-04-04T07:22:49.657377"
  }
]

```

Рис. 5: Список файлов

Пример использования показан на рисунках:

В данном примере рассматривается система из двух слоев. Но предложенное решение никак не ограничено своей топологией. Пока каждая пара уровней связана сервисом, можно создавать любую структуру.

### 3 Dcache + docker compose

Также в процессе разработки системы были собраны контейнеры для разных частей dcache. Данное решение подходит для **быстрого локального развертывания** с логическим распределением.

Директория compose/ является набором блоков для построения системы в среде docker compose. Файл compose/compose.yaml позволяет запустить dcache в виде трех узлов общающихся через Zookeeper в отдельном контейнере.

Каждая отдельная поддиректория вроде compose/dcache-core содержит dockerfile для сборки контейнера для описанного узла. Также в compose/dcache-core/config/ содержатся основные файлы конфигурации dcache для узла. Для быстрого запуска требуется установить docker <https://docs.docker.com/engine/install/>. После чего выполнить в папке с compose.yaml:

```
docker compose up --build
```

Данная архитектура в точности повторяет систему описанную в книге: <https://www.dcache.org/manuals/Book-10.0/install.shtml> к моменту распределенной системы на нескольких узлах. Следовательно можно пользоваться командами из книги чтобы записать файл через webdav:

«The WebDAV door is listening on port 2880 (the default). We can upload a file into dCache using that protocol, with the curl client.

```
curl -u tester:TooManySecrets -L -T /bin/bash  
http://localhost:2880/home/tester/test-file-1
```

The file may also be downloaded:

```
curl -u tester:TooManySecrets -L -o /tmp/test-file-1  
http://localhost:2880/home/tester/test-file-1>>
```