🐳 Руководство по развертыванию DHT сети

Версия: 1.0
Дата: 6 июня 2025
Проект: Distributed Hash Table (DHT) на основе Kademlia

📋 Содержание

🚀 Быстрый старт

Для тех, кто хочет запустить DHT сеть за 5 минут:

```
# 1. Клонируйте проект
git clone <repository-url>
cd dht

# 2. Запустите сеть
./dht-network.sh start

# 3. Откройте дашборд
open http://localhost:3000
```

Готово! У вас работает DHT сеть из 6 узлов.

💻 Требования к системе

Минимальные требования:

- **ОС:** Linux, macOS, Windows 10+ (с WSL2)
- **RAM:** 4 GB свободной памяти
- **CPU:** 2 ядра
- **Диск:** 2 GB свободного места
- **Сеть:** Доступ к интернету для загрузки образов

Рекомендуемые требования:

- **RAM:** 8 GB
- **CPU:** 4 ядра
- **Диск:** 5 GB свободного места

Порты:

Убедитесь, что следующие порты свободны: - **8080-8085:** DHT узлы - **3000:** Веб-дашборд - **9090:** Prometheus (опционально) - **3001:** Grafana (опционально)

🔧 Установка зависимостей

Ubuntu/Debian:

```
# Обновление системы
sudo apt update && sudo apt upgrade -y

# Установка Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER

# Установка Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/latest/
download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/
bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Перезагрузка для применения изменений
sudo reboot
```

CentOS/RHEL/Fedora:

```
# Установка Docker
sudo dnf install -y docker docker-compose
sudo systemctl enable --now docker
sudo usermod -aG docker $USER

# Перезагрузка
sudo reboot
```

macOS:

```
# Установка через Homebrew
brew install docker docker-compose

# Или скачайте Docker Desktop с официального сайта
# https://www.docker.com/products/docker-desktop
```

Windows:

1. Установите **Docker Desktop** с официального сайта
2. Включите **WSL2** интеграцию
3. Используйте **WSL2 терминал** для команд

Проверка установки:

```
# Проверка Docker
docker --version
docker-compose --version

# Тест Docker
docker run hello-world
```

🌐 Развертывание сети

Структура проекта:

```
dht/
├── Dockerfile                 # Образ DHT узла
├── docker-compose.yml         # Конфигурация сети
├── dht-network.sh            # Скрипт управления
```

```
├── src/                          # Исходный код
├── monitoring/                   # Мониторинг
│   ├── dashboard/               # Веб-дашборд
│   ├── nginx.conf               # Конфигурация Nginx
│   ├── prometheus.yml           # Конфигурация Prometheus
│   └── grafana/                 # Настройки Grafana
└── README.md
```

Способ 1: Автоматический запуск (рекомендуется)

```bash
# Переход в директорию проекта
cd dht

# Запуск всей сети одной командой
./dht-network.sh start
```

Что происходит: 1. 🔨 Сборка Docker образов 2. 🚀 Запуск 6 DHT узлов 3. 🌐 Настройка сети и маршрутизации 4. 📊 Запуск мониторинга 5. ✅ Проверка готовности

Способ 2: Ручной запуск

```bash
# Сборка образов
docker-compose build

# Запуск основных сервисов
docker-compose up -d

# Запуск с мониторингом (Prometheus + Grafana)
docker-compose --profile monitoring up -d

# Проверка статуса
docker-compose ps
```

Способ 3: Пошаговый запуск

```bash
# 1. Запуск bootstrap узла
docker-compose up -d dht-bootstrap

# 2. Ожидание готовности (30 сек)
sleep 30

# 3. Запуск остальных узлов
docker-compose up -d dht-node-2 dht-node-3 dht-node-4 dht-
node-5 dht-node-6
```

```
# 4. Запуск мониторинга
docker-compose up -d dht-monitor
```

Проверка развертывания:

```
 # Проверка статуса контейнеров
docker-compose ps

# Проверка логов
docker-compose logs dht-bootstrap

# Проверка здоровья узлов
./dht-network.sh status

# Тест сети
./dht-network.sh test
```

Ожидаемый результат:

```
✅ Узел на порту 8080: ЗДОРОВ
✅ Узел на порту 8081: ЗДОРОВ
✅ Узел на порту 8082: ЗДОРОВ
✅ Узел на порту 8083: ЗДОРОВ
✅ Узел на порту 8084: ЗДОРОВ
✅ Узел на порту 8085: ЗДОРОВ

Статус сети: 6/6 узлов здоровы
🎉 Вся сеть работает отлично!
```

🧪 Использование и тестирование

Веб-интерфейс:

- **Дашборд:** http://localhost:3000
- **Prometheus:** http://localhost:9090
- **Grafana:** http://localhost:3001 (admin/admin)

API тестирование:

Сохранение данных:

```
# Сохранение через любой узел
curl -X PUT "http://localhost:8080/api/store/my-key" \
  -H "Content-Type: application/json" \
  -d '{"value": "Hello DHT!"}'
```

Поиск данных:

```
# Поиск через любой узел
curl "http://localhost:8081/api/find/my-key"
```

Получение статистики:

```
# Статистика узла
curl "http://localhost:8080/api/stats"

# Список всех ключей
curl "http://localhost:8080/api/keys"

# Информация о сети
curl "http://localhost:8080/api/network"
```

Автоматические тесты:

```
# Тест репликации данных
./dht-network.sh test

# Мониторинг в реальном времени
./dht-network.sh monitor

# Очистка всех данных
./dht-network.sh clean
```

Примеры использования:

Пример 1: Сохранение пользовательских данных

```
# Сохранение профиля пользователя
curl -X PUT "http://localhost:8080/api/store/user:123" \
  -H "Content-Type: application/json" \
```

```
  -d '{
    "name": "John Doe",
    "email": "john@example.com",
    "created": "2025-06-06"
  }'

# Поиск профиля
curl "http://localhost:8082/api/find/user:123"
```

Пример 2: Кэширование данных

```
# Сохранение кэша
curl -X PUT "http://localhost:8081/api/store/
cache:session:abc123" \
  -H "Content-Type: application/json" \
  -d '{
    "userId": 123,
    "permissions": ["read", "write"],
    "expires": "2025-06-07T12:00:00Z"
  }'
```

Пример 3: Конфигурационные данные

```
# Сохранение конфигурации
curl -X PUT "http://localhost:8083/api/store/
config:app:settings" \
  -H "Content-Type: application/json" \
  -d '{
    "theme": "dark",
    "language": "ru",
    "notifications": true
  }'
```

📊 Мониторинг

Веб-дашборд (http://localhost:3000)

Возможности: - 📈 Статистика сети в реальном времени - 🔍 Статус каждого узла - 📊 Количество ключей и соединений - ⚡ Время отклика узлов - 🧪 Встроенные тесты сети - 📝 Добавление тестовых данных

Интерфейс: - Автоматическое обновление каждые 10 секунд - Цветовая индикация статуса узлов - Кнопки для быстрых действий - Адаптивный дизайн для мобильных устройств

Prometheus (http://localhost:9090)

Метрики DHT: - `dht_nodes_total` - Общее количество узлов - `dht_keys_total` - Количество ключей в сети - `dht_connections_active` - Активные соединения - `dht_operations_total` - Счетчик операций - `dht_response_time_seconds` - Время отклика

Полезные запросы:

```
# Средняя нагрузка на узлы
avg(dht_node_load_percent)

# Количество ключей по узлам
sum by (node) (dht_keys_total)

# Успешность операций
rate(dht_operations_total{status="success"}[5m])
```

Grafana (http://localhost:3001)

Логин: admin / admin

Готовые дашборды: - DHT Network Overview - Node Performance - Data Distribution - Network Health

Командная строка:

```
# Статус всех узлов
./dht-network.sh status

# Мониторинг в реальном времени
./dht-network.sh monitor

# Просмотр логов
./dht-network.sh logs

# Логи конкретного узла
./dht-network.sh logs dht-bootstrap
```

⚙️ Управление сетью

Основные команды:

```
# Запуск сети
./dht-network.sh start

# Остановка сети
./dht-network.sh stop

# Перезапуск сети
./dht-network.sh restart

# Проверка статуса
./dht-network.sh status

# Тестирование
./dht-network.sh test

# Очистка данных
./dht-network.sh clean

# Мониторинг
./dht-network.sh monitor

# Просмотр логов
./dht-network.sh logs [service]

# Справка
./dht-network.sh help
```

Docker Compose команды:

```
# Просмотр статуса
docker-compose ps

# Просмотр логов
docker-compose logs -f

# Остановка сервиса
docker-compose stop dht-node-2

# Запуск сервиса
docker-compose start dht-node-2

# Перезапуск сервиса
docker-compose restart dht-bootstrap
```

```
# Масштабирование (добавление узлов)
docker-compose up -d --scale dht-node-2=2

# Обновление образов
docker-compose pull && docker-compose up -d
```

Управление данными:

```
# Резервное копирование данных
docker-compose exec dht-bootstrap tar -czf /tmp/backup.tar.gz /
app/data

# Восстановление данных
docker-compose exec dht-bootstrap tar -xzf /tmp/backup.tar.gz -
C /

# Очистка данных конкретного узла
curl -X DELETE "http://localhost:8080/api/clear"

# Экспорт всех ключей
curl "http://localhost:8080/api/export" > dht_backup.json

# Импорт ключей
curl -X POST "http://localhost:8080/api/import" \
  -H "Content-Type: application/json" \
  -d @dht_backup.json
```

🔧 Troubleshooting

Частые проблемы и решения:

Проблема: Контейнеры не запускаются

```
# Проверка логов
docker-compose logs

# Проверка портов
netstat -tulpn | grep :8080

# Освобождение портов
sudo lsof -ti:8080 | xargs kill -9

# Очистка Docker
docker system prune -f
```

Проблема: Узлы не видят друг друга

```
# Проверка сети Docker
docker network ls
docker network inspect dht_dht-network

# Проверка DNS
docker-compose exec dht-bootstrap nslookup dht-node-2

# Перезапуск сети
./dht-network.sh restart
```

Проблема: Медленная работа

```
# Проверка ресурсов
docker stats

# Увеличение памяти для Docker Desktop
# Settings -> Resources -> Memory -> 4GB+

# Проверка дискового пространства
df -h
docker system df
```

Проблема: Данные не реплицируются

```
# Проверка связности узлов
./dht-network.sh test

# Проверка логов репликации
docker-compose logs | grep -i replication

# Ручная проверка репликации
for port in 8080 8081 8082; do
  echo "Node $port:"
  curl -s "http://localhost:$port/api/keys" | jq length
done
```

Проблема: Веб-дашборд не работает

```
# Проверка Nginx
docker-compose logs dht-monitor

# Проверка доступности узлов
curl -f "http://localhost:8080/api/health"
```

```
# Перезапуск мониторинга
docker-compose restart dht-monitor
```

Диагностические команды:

```
# Полная диагностика системы
echo "=== Docker версия ==="
docker --version
docker-compose --version

echo "=== Статус контейнеров ==="
docker-compose ps

echo "=== Использование ресурсов ==="
docker stats --no-stream

echo "=== Сетевая конфигурация ==="
docker network inspect dht_dht-network

echo "=== Проверка портов ==="
for port in 8080 8081 8082 8083 8084 8085 3000; do
  echo -n "Port $port: "
  nc -z localhost $port && echo "OPEN" || echo "CLOSED"
done

echo "=== Тест API ==="
for port in 8080 8081 8082; do
  echo -n "API $port: "
  curl -s -f "http://localhost:$port/api/health" && echo "OK"
|| echo "FAIL"
done
```

Логи и отладка:

```
# Детальные логи всех сервисов
docker-compose logs -f --tail=100

# Логи конкретного узла
docker-compose logs -f dht-bootstrap

# Логи с временными метками
docker-compose logs -f -t

# Поиск ошибок в логах
docker-compose logs | grep -i error
```

```
# Экспорт логов в файл
docker-compose logs > dht_logs.txt
```

Восстановление после сбоев:

```
# Полная перезагрузка
./dht-network.sh stop
docker system prune -f
./dht-network.sh start

# Восстановление отдельного узла
docker-compose restart dht-node-2
sleep 10
./dht-network.sh status

# Восстановление данных из резервной копии
# (если настроено резервное копирование)
docker-compose exec dht-bootstrap restore-backup.sh
```

📚 API документация

Базовые операции:

Сохранение данных

```
PUT /api/store/{key}
Content-Type: application/json

{
  "value": "any JSON data",
  "ttl": 3600,
  "replicas": 3
}
```

Поиск данных

```
GET /api/find/{key}

Response:
{
  "key": "my-key",
  "value": "stored data",
  "node": "node-8080",
  "replicas": ["node-8080", "node-8081", "node-8082"],
```

```
  "timestamp": "2025-06-06T12:00:00Z"
}
```

Удаление данных

```
DELETE /api/delete/{key}

Response:
{
  "deleted": true,
  "key": "my-key",
  "nodes": ["node-8080", "node-8081"]
}
```

Информационные API:

Статус узла

```
GET /api/health

Response:
{
  "status": "healthy",
  "node_id": "node-8080",
  "uptime": 3600,
  "connections": 5,
  "keys": 150
}
```

Статистика узла

```
GET /api/stats

Response:
{
  "node_id": "node-8080",
  "keys": 150,
  "connections": 5,
  "operations": {
    "store": 1000,
    "find": 2500,
    "delete": 50
  },
  "performance": {
    "avg_response_time": 25,
    "success_rate": 0.99
```

```
  }
}
```

Информация о сети

```
GET /api/network

Response:
{
  "total_nodes": 6,
  "online_nodes": 6,
  "network_size": 6,
  "replication_factor": 3,
  "nodes": [
    {
      "id": "node-8080",
      "address": "dht-bootstrap:8080",
      "status": "online",
      "keys": 150
    }
  ]
}
```

Список ключей

```
GET /api/keys?limit=100&offset=0

Response:
{
  "keys": [
    {
      "key": "user:123",
      "size": 256,
      "replicas": 3,
      "created": "2025-06-06T12:00:00Z"
    }
  ],
  "total": 150,
  "limit": 100,
  "offset": 0
}
```

Административные API:

Очистка данных

```
DELETE /api/clear

Response:
{
  "cleared": true,
  "keys_deleted": 150
}
```

Экспорт данных

```
GET /api/export

Response:
{
  "export_time": "2025-06-06T12:00:00Z",
  "node_id": "node-8080",
  "data": [
    {
      "key": "user:123",
      "value": {...},
      "metadata": {...}
    }
  ]
}
```

Импорт данных

```
POST /api/import
Content-Type: application/json

{
  "data": [
    {
      "key": "user:123",
      "value": {...}
    }
  ]
}
```

Метрики для Prometheus:

```
GET /api/metrics

Response:
# HELP dht_keys_total Total number of keys stored
# TYPE dht_keys_total gauge
dht_keys_total{node="node-8080"} 150

# HELP dht_operations_total Total number of operations
# TYPE dht_operations_total counter
dht_operations_total{node="node-8080",operation="store"} 1000
```

🎯 Примеры интеграции

Python клиент:

```python
import requests
import json

class DHTClient:
    def __init__(self, nodes=None):
        self.nodes = nodes or [
            "http://localhost:8080",
            "http://localhost:8081",
            "http://localhost:8082"
        ]

    def store(self, key, value):
        data = {"value": value}
        for node in self.nodes:
            try:
                response = requests.put(f"{node}/api/store/
{key}",
                                        json=data, timeout=5)
                if response.ok:
                    return True
            except:
                continue
        return False

    def find(self, key):
        for node in self.nodes:
            try:
                response = requests.get(f"{node}/api/find/
{key}",
```

```
                                        timeout=5)
                if response.ok:
                    return response.json()
            except:
                continue
        return None

# Использование
client = DHTClient()
client.store("user:123", {"name": "John", "age": 30})
user = client.find("user:123")
print(user)
```

JavaScript клиент:

```
class DHTClient {
    constructor(nodes = ['http://localhost:8080', 'http://
localhost:8081']) {
        this.nodes = nodes;
    }

    async store(key, value) {
        for (const node of this.nodes) {
            try {
                const response = await fetch(`${node}/api/store/
${key}`, {
                    method: 'PUT',
                    headers: { 'Content-Type': 'application/
json' },
                    body: JSON.stringify({ value })
                });
                if (response.ok) return true;
            } catch (e) {
                continue;
            }
        }
        return false;
    }

    async find(key) {
        for (const node of this.nodes) {
            try {
                const response = await fetch(`${node}/api/find/$
{key}`);
                if (response.ok) return await response.json();
            } catch (e) {
                continue;
            }
        }
        return null;
```

```
    }
}

// Использование
const client = new DHTClient();
await client.store('session:abc', { userId: 123, expires:
Date.now() + 3600000 });
const session = await client.find('session:abc');
console.log(session);
```

🔒 Безопасность

Рекомендации по безопасности:

1. **Сетевая безопасность:**
2. Используйте firewall для ограничения доступа
3. Настройте VPN для удаленного доступа

4. Измените стандартные порты в production

5. **Аутентификация:**

6. Добавьте API ключи для доступа
7. Используйте HTTPS в production

8. Настройте rate limiting

9. **Мониторинг:**

10. Включите логирование всех операций
11. Настройте алерты на подозрительную активность
12. Регулярно проверяйте логи

Пример конфигурации для production:

```
# docker-compose.prod.yml
version: '3.8'
services:
  dht-bootstrap:
    environment:
      - ENABLE_AUTH=true
      - API_KEY=your-secret-key
      - LOG_LEVEL=INFO
    volumes:
      - ./ssl:/app/ssl:ro
```

```
    networks:
      - dht-internal

networks:
  dht-internal:
    driver: bridge
    internal: true
```

📞 Поддержка

Получение помощи:

1. **Документация:** Читайте этот файл полностью
2. **Логи:** Всегда проверяйте логи первым делом
3. **Диагностика:** Используйте встроенные команды диагностики
4. **Сообщество:** Создавайте issues в репозитории

Полезные ссылки:

- Docker документация
- Docker Compose документация
- Kademlia протокол
- Prometheus документация

Документ подготовлен: 6 июня 2025
Версия: 1.0
Автор: Manus AI Agent
Проект: DHT Kademlia Implementation