

# Мануал по развертыванию уязвимого приложения

14 февраля 2024 г.

## Содержание

1	Концепция приложения	2
2	Перечень необходимых приложений и их функционал	3
3	Debian	3
4	Работа с Kubernetes	7
5	Kafka	11
6	Grafana	11
7	Clickhouse	12
8	Sorl	13

## 1. Концепция приложения

Было разработано заведомо уязвимое приложение, принцип работы которого похож на книжный учёт. Приложение предоставляет информацию о книгах и использует следующие инструменты: Kubernetes, Kafka, Solr, Clickhouse, Grafana и Prometheus.

Логика работы приложения описывается в двух сценариях:

### Сценарий 1 - Загрузка файла книги пользователем:

- 1) Пользователь при помощи интерфейса App1 загружает json файл книги.
- 2) App1 через Kafka, отправляет пользовательский файл к App2.
- 3) App2 забирает из файла нужную информацию и направляет в solr на индексацию

### Сценарий 2 - Получение информации о книге:

- 1) Пользователь при помощи интерфейса App1 отправляет запрос к Solr. Затем следует тот же путь до App2.
- 2) App2 обращается к Solr, и получает данные о всех книгах, подходящих по запросу.
- 3) Данные направляются к App1 через Kafka и отображаются пользователю.

Использование Kubernetes обусловлено необходимостью масштабирования производительности, а Grafana применяется для аналитики и мониторинга производительности Kafka, с целью выяснения причин возможных задержек и улучшения работы системы.

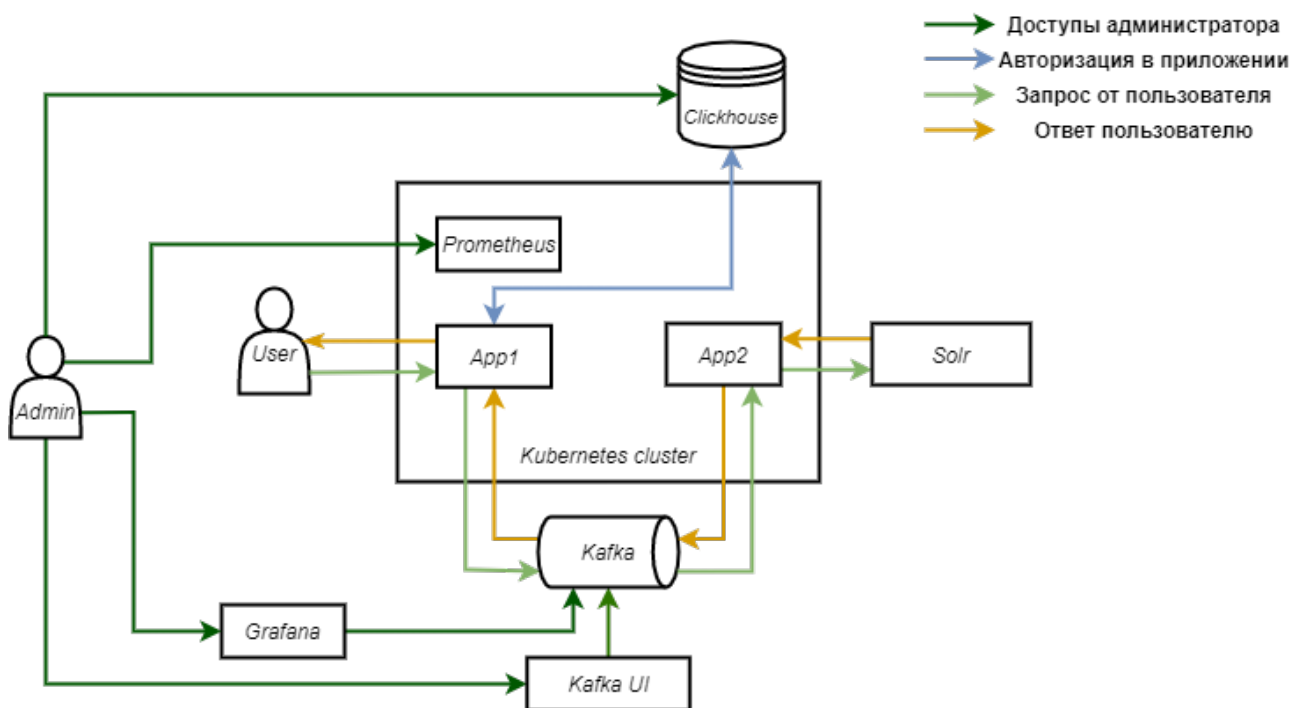


Рис. 1. Схема приложения

## 2. Перечень необходимых приложений и их функционал

### Grafana:

**Описание:** Grafana - это открытое программное обеспечение для мониторинга и визуализации данных. Он обеспечивает гибкие и интерактивные панели для отображения метрик и графиков.

**Назначение:** Используется для создания красочных и информативных дашбордов, позволяющих вам мониторить различные аспекты системы и приложений в реальном времени.

### Prometheus:

**Описание:** Prometheus - это система мониторинга с открытым исходным кодом. Он собирает метрики с различных источников, хранит их и предоставляет возможность выполнения запросов и создания графиков.

**Назначение:** Используется для сбора и анализа временных рядов метрик, таких как производительность приложений и состояние системы.

### Kubernetes:

**Описание:** Kubernetes - это открытая система управления контейнерами, обеспечивающая автоматизацию развертывания, масштабирования и управления контейнеризированными приложениями.

**Назначение:** Используется для оркестрации и управления контейнерами, обеспечивая автоматизацию процессов развертывания и управления ресурсами.

### Kafka:

**Описание:** Apache Kafka - это платформа для потоковой обработки данных и передачи сообщений. Он предоставляет механизм для эффективной передачи сообщений между различными компонентами системы.

**Назначение:** Используется для создания стойких и масштабируемых систем обмена сообщениями, что делает его подходящим для стриминга и обработки данных в реальном времени.

### Kafka UI:

**Описание:** Kafka User Interface - инструмент для мониторинга топиков Kafka.

**Назначение:** Используется для манипуляций с топиками в Kafka и просмотра сообщений, проходящих через Kafka.

### ClickHouse:

**Описание:** ClickHouse - это открытая колоночная система управления базами данных для аналитики. Он разработан для эффективного анализа больших объемов данных.

**Назначение:** Используется для выполнения аналитических запросов на больших объемах данных, что делает его подходящим для систем, требующих высокой производительности и быстрого доступа к данным.

Эти инструменты могут использоваться в совокупности для построения комплексных систем мониторинга, аналитики и обработки данных в среде Kubernetes.

## 3. Debian

### Установка необходимого софта

```
1 sudo apt install apt-transport-https ca-certificates curl gnupg2  
   software-properties-common
```

## Загрузка docker-ce

```
1 curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key  
   add -  
2 sudo add-apt-repository "deb [arch=amd64]  
   https://download.docker.com/linux/debian $(lsb_release -cs) stable"  
3 sudo apt update  
4 apt-cache policy docker-ce #Здесь получим список доступных версий для  
   Debian 10
```

## Установка и запуск docker

```
1 sudo apt install docker-ce=5:19.03.5~3-0~debian-buster  
2 sudo systemctl enable docker  
3 sudo systemctl start docker
```

## Установка kubelet, kubeadm и kubectl

```
1 sudo apt-get update && sudo apt-get install -y apt-transport-https curl  
2 curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo  
   apt-key add -  
3 echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee  
   /etc/apt/sources.list.d/kubernetes.list  
4 sudo apt-get update  
5 sudo apt-get install -y kubelet=1.20.0-00 kubeadm=1.20.0-00  
   kubectl=1.20.0-00  
6 sudo apt-mark hold kubelet kubeadm kubectl
```

## Попытка инициализации Docker

```
1 sudo kubeadm init --kubernetes-version=1.20.0
```

## Решение проблем:

### Проблема 1 - Отключаем swap

```
1 sudo swapoff -a
```

Данную команду необходимо использовать каждый раз перед запуском кластера

**Проблема 2 - Используем рекомендуемый драйвер "systemd" вместо "cgroupfs"**

```
1 sudo mkdir -p /etc/systemd/system/docker.service.d
2 sudo nano /etc/systemd/system/docker.service.d/override.conf
```

Содержимое override.conf:

```
1 [Service]
2 ExecStart=
3 ExecStart=/usr/bin/dockerd --host=fd:// --add-runtime
    docker-runc=/usr/local/bin/docker-runc --add-runtime
    docker-containerd=/usr/local/bin/docker-containerd --config-file
    /etc/docker/daemon.json
```

```
1 sudo nano /etc/docker/daemon.json
```

Содержимое daemon.json:

```
1 {
2   "exec-opts": ["native.cgroupdriver=systemd"],
3   "log-driver": "json-file",
4   "log-opts": {
5     "max-size": "100m"
6   },
7   "storage-driver": "overlay2"
8 }
```

```
1 sudo systemctl daemon-reload
2 sudo systemctl restart docker
```

**Проверка драйвера**

```
1 docker info | grep -i cgroup
```

**Инициализация**

```
1 sudo kubeadm init --kubernetes-version=1.20.0
```

### Завершение инициализации Kubernetes

```
1 Your Kubernetes control-plane has initialized successfully!
2
3 To start using your cluster, you need to run the following as a regular
  user:
4
5     mkdir -p $HOME/.kube
6     sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
7     sudo chown $(id -u):$(id -g) $HOME/.kube/config
8
9 Alternatively, if you are the root user, you can run:
10
11     export KUBECONFIG=/etc/kubernetes/admin.conf
12
13 You should now deploy a pod network to the cluster.
14 Run "kubectl apply -f [podnetwork].yaml" with one of the options listed
  at:
15     https://kubernetes.io/docs/concepts/cluster-administration/addons/
16
17 Then you can join any number of worker nodes by running the following on
  each as root:
18
19 \begin{lstlisting}[language=bash]
20 kubectl join 10.0.2.15:6443 --token mcc5mr.slr9teht39was8ei \
21   --discovery-token-ca-cert-hash
    sha256:91787640cf674c5d6ab11d885a97b889cfe7ab9fb238b95e00f1001b6cd5366c
```

### Установка сетевого плагина (в данном случае calico)

```
1 kubectl apply -f
   https://docs.projectcalico.org/v3.18/manifests/calico.yaml
```

### Проверка работы кластера

```
1 kubectl get nodes
2 kubectl get pods --all-namespaces
```

### Загрузка OpenLens

Он автоматически цепляется к кластеру

```
1 https://github.com/MuhammedKalkan/OpenLens/releases
```

Команды для включения и выключения кластера Команды вкл/выкл

```
1 kubectl drain deb1 --delete-local-data --ignore-daemonsets --force;  
2 kubectl uncordon deb1 для включения
```

## 4. Работа с Kubernetes

### Шаг 0: Установка Helm

Перейдите на официальный сайт Helm по ссылке: <https://helm.sh/docs/intro/install/> и выполните инструкции по установке Helm.

### Шаг 1: Управление тэйнтами на узле

Используя kubectl, выполните команду для управления тэйнтами на узле deb1:

```
1 kubectl taint nodes deb1 node-role.kubernetes.io/master -
```

Эта команда удаляет тэйнт, связанный с ролью "мастер" на узле deb1.

### Шаг 2: Создание Traefik Deployment и Service

Создайте файл traefik-deployment.yaml:

```
1 apiVersion: apps/v1  
2 kind: Deployment  
3 metadata:  
4   name: traefik  
5   namespace: traefik  
6 spec:  
7   replicas: 1  
8   selector:  
9     matchLabels:  
10    app: traefik  
11   template:  
12     metadata:  
13       labels:  
14         app: traefik  
15     spec:  
16       containers:  
17       - name: traefik  
18         image: traefik:v2.6  
19         ports:  
20         - name: web  
21           containerPort: 80  
22         - name: websecure  
23           containerPort: 443
```

Примените деплоймент с помощью команды:

```
1 kubectl apply -f traefik-deployment.yaml
```

Создайте файл `traefik-service.yaml` и добавьте следующий контент:

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: traefik
5   namespace: traefik
6 spec:
7   selector:
8     app: traefik
9   ports:
10    - name: web
11      protocol: TCP
12      port: 80
13    - name: websecure
14      protocol: TCP
15      port: 443
```

Примените службу с помощью команды:

```
1 kubectl apply -f traefik-service.yaml
```

### Шаг 3: Создание PV и PVC для MetalLB

Создайте файл `pv.yaml` и добавьте следующий контент:

```
1 apiVersion: v1
2 kind: PersistentVolume
3 metadata:
4   name: metallb-pv
5 spec:
6   capacity:
7     storage: 1Gi
8   accessModes:
9     - ReadWriteOnce
10  hostPath:
11    path: /metallb-data
```

Примените PV с помощью команды:

```
1 kubectl apply -f pv.yaml
```

Создайте файл `pvc.yaml` и добавьте следующий контент:

```
1 pvc.yaml
2 apiVersion: v1
```



```
3 kind: PersistentVolumeClaim
4 metadata:
5   name: metallb-pvc
6   namespace: metallb-system
7 spec:
8   accessModes:
9     - ReadWriteOnce
10  resources:
11    requests:
12      storage: 1Gi
```

Примените PVC с помощью команды:

```
1 kubectl apply -f pvc.yaml
```

## Шаг 4: Установка MetalLB

Создайте файл `metallb-system.yaml` и добавьте следующий контент:

```
1 apiVersion: v1
2 kind: Namespace
3 metadata:
4   name: metallb-system
```

Примените Namespace с помощью команды:

```
1 kubectl apply -f metallb-system.yaml
```

Создайте Bash-скрипт, например, `configure-kube-proxy.sh`, и добавьте следующий контент:

```
1 #!/bin/bash
2
3 # see what changes would be made, returns nonzero returncode if different
4 kubectl get configmap kube-proxy -n kube-system -o yaml | \
5   sed -e "s/strictARP: false/strictARP: true/" | \
6   kubectl diff -f - -n kube-system
7
8 # actually apply the changes, returns nonzero returncode on errors only
9 kubectl get configmap kube-proxy -n kube-system -o yaml | \
10  sed -e "s/strictARP: false/strictARP: true/" | \
11  kubectl apply -f - -n kube-system
```

Выполните Bash-скрипт:

```
1 ./configure-kube-proxy.sh
```

Примените конфигурацию MetalLB:

```
1 kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.13.9/config/manifests/m
```

Создайте файл `metallb-config.yaml` и добавьте следующий контент:

```
1 apiVersion: metallb.io/v1beta1
2 kind: IPAddressPool
3 metadata:
4   name: default
5   namespace: metallb-system
6 spec:
7   addresses:
8   - 10.0.2.240-10.0.2.250
9   autoAssign: true
10 ---
11 apiVersion: metallb.io/v1beta1
12 kind: L2Advertisement
13 metadata:
14   name: default
15   namespace: metallb-system
16 spec:
17   ipAddressPools:
18   - default
```

Примените конфигурацию MetalLB:

```
1 kubectl apply -f metallb-config.yaml
```

## Шаг 5: Развертывание App1 и App2

Установите Maven с помощью команды:

```
1 sudo apt-get install maven
```

Установите Java и укажите его в пути:

```
1 sudo apt-get install openjdk-11-jdk
2 export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
3 export PATH=$JAVA_HOME/bin:$PATH
```

Загружаем файлы лежащие по адресу

```
1 https://github.com/DasLiberasten/Custom_Vuln_App/tree/main
2 /Ingress-to-kafka/Cluster_deploy
3 https://github.com/DasLiberasten/Custom_Vuln_App/tree/main
4 /KafkaToSolr/Cluster_deploy
```

И применяем их

```
1 kubectl apply -f <filename>.yaml
```

## 5. Kafka

Загружаем docker-compose.yml:

```
1 https://github.com/DasLiberasten/Custom_Vuln_App/tree/main
```

И запускаем Kafka, Kafka UI и Zookeeper

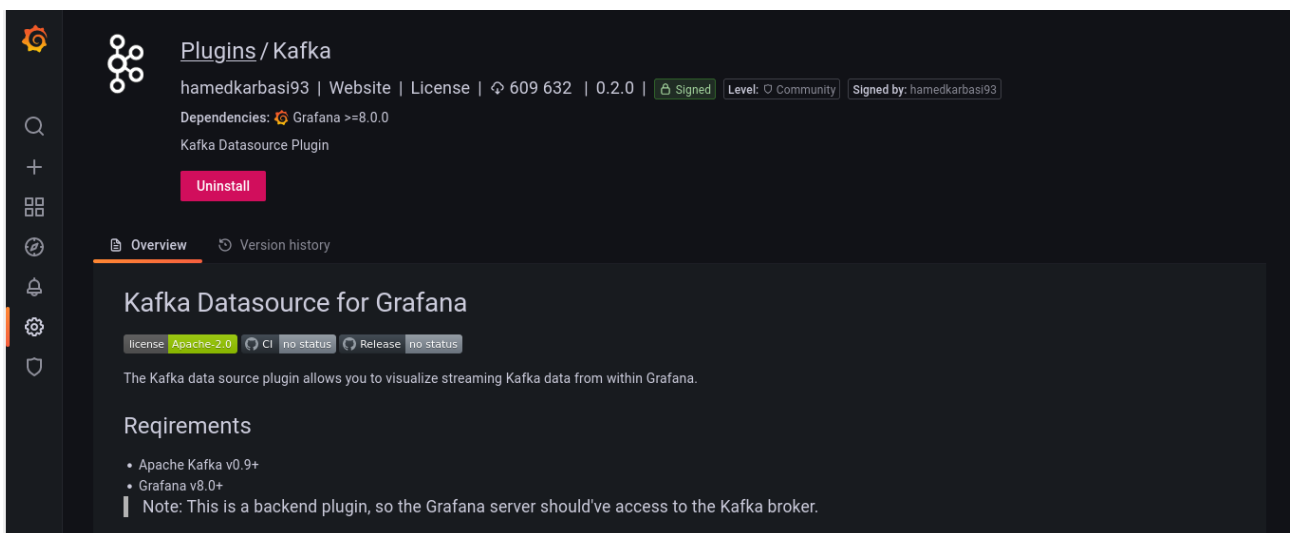
```
1 docker-compose up
```

## 6. Grafana

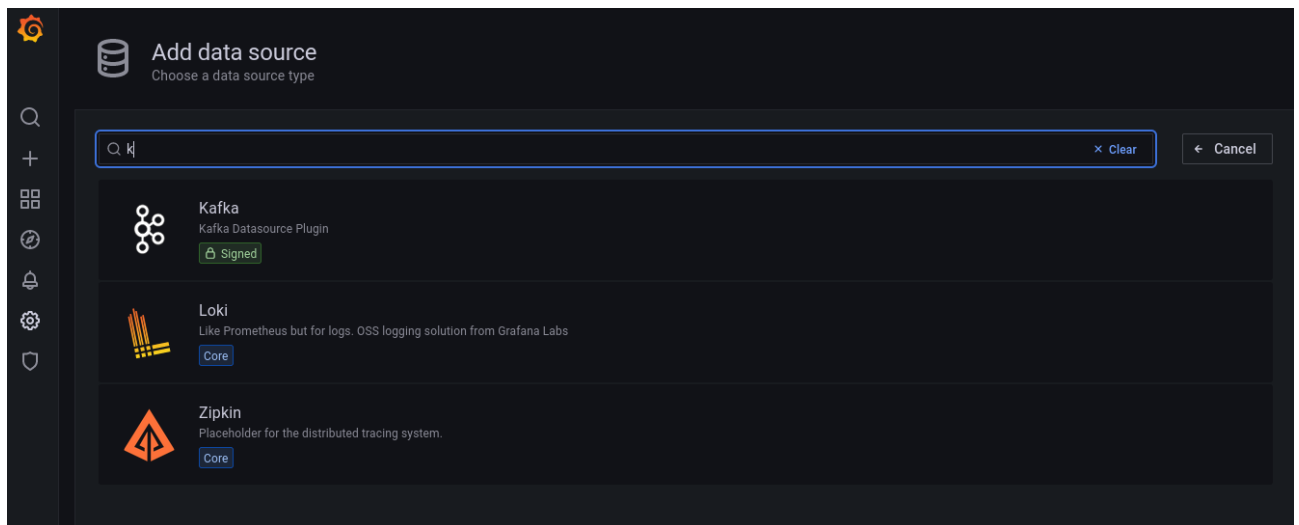
Теперь необходимо поднять Grafana:

```
1 docker run -d -p 3000:3000 --name my-grafana my-grafana:8.2.3
```

Переходим на 3000 порт. Логин и пароль - admin, после первого запуска устанавливаем свои по желанию. Внутри также нужно проставить расширение для Kafka.



Далее создаем Data Source. Выбираем Kafka



Servers: your-ip:29092, your-ip:29093

## 7. Clickhouse

Создаем docker-compose.yml

```

1 version: '2'
2
3 services:
4   clickhouse:
5     image: clickhouse/clickhouse-server:21.9.6
6     container_name: clickhouse
7     ports:
8       - "8123:8123" # HTTP порт
9       - "9000:9000" # Native порт
10      - "9009:9009" # Native SSL порт
11     environment:
12       - CLICKHOUSE_USER=default
13       - CLICKHOUSE_PASSWORD=password
14     volumes:
15       - ./clickhouse-data:/var/lib/clickhouse

```

```

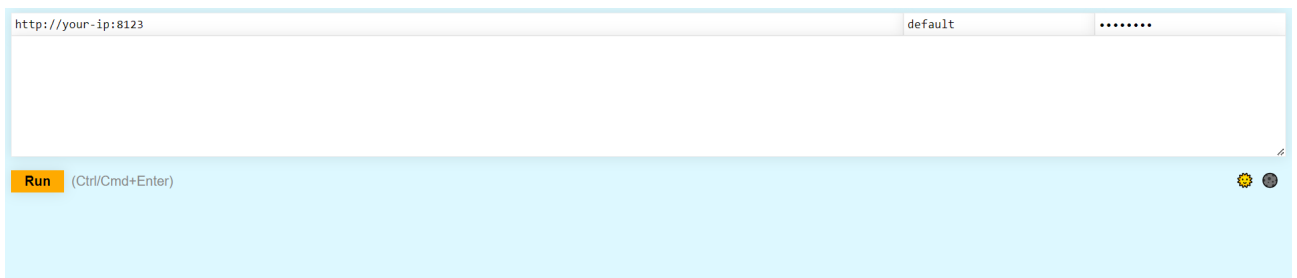
1 docker-compose up -d

```

Теперь можно через интерфейс Clickhouse <http://10.0.2.15:8123/play> взаимодействовать с базой данных.

## Sorl

---



Создаем базу данных

```
1 CREATE DATABASE IF NOT EXISTS passworddb;  
2 USE passworddb;
```

Создаем таблицу паролей

```
1 CREATE TABLE passworddb.users (  
2     username String,  
3     password String  
4 ) ENGINE = MergeTree()  
5 ORDER BY username;
```

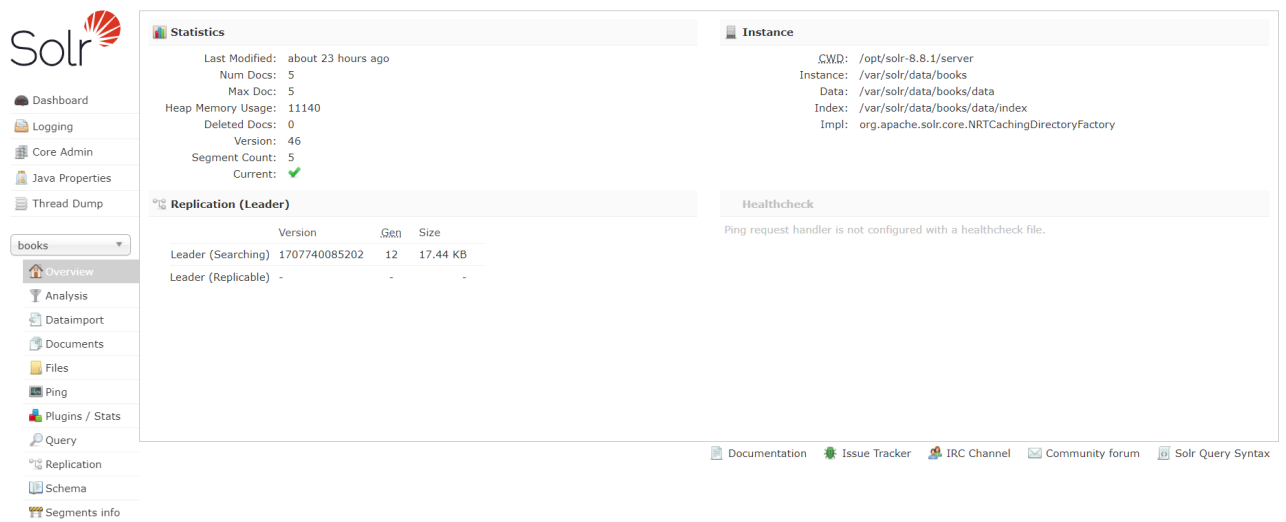
И добавляем туда пользователей каких желаем

## 8. Sorl

Необходимо теперь поднять Solr:

```
1 mkdir solrdata  
2 sudo chmod -R 777 solrdata  
3 docker run -d -v "$PWD/solrdata:/var/solr" -p 8983:8983 --name solrbooks  
   --user 8983:8983 solr:8.8.1 solr-precreate gettingstarted  
4 docker exec -it solrbooks bin/solr create -c books
```

По <http://your-ip:8983/solr/> можем посмотреть на интерфейс Solr



The screenshot shows the Solr Admin interface for a cluster named 'books'. The left sidebar contains navigation links: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, and a dropdown menu for 'books' which is currently open, showing Overview (selected), Analysis, Dataimport, Documents, Files, Ping, Plugins / Stats, Query, Replication, Schema, and Segments info. The main content area is divided into three panels: 'Statistics' showing cluster health (Last Modified: about 23 hours ago, Num Docs: 5, Max Doc: 5, Heap Memory Usage: 11140, Deleted Docs: 0, Version: 46, Segment Count: 5, Current: green), 'Instance' showing configuration (CWD: /opt/solr-8.8.1/server, Instance: /var/solr/data/books, Data: /var/solr/data/books/data, Index: /var/solr/data/books/data/index, Impl: org.apache.solr.core.NRTCachingDirectoryFactory), and 'Replication (Leader)' showing a table with columns Version, .Gen, and Size. The table has two rows: 'Leader (Searching)' with values 1707740085202, 12, and 17.44 KB, and 'Leader (Replicable)' with dashes. A 'Healthcheck' panel below shows a message: 'Ping request handler is not configured with a healthcheck file.' At the bottom right of the main area are links for Documentation, Issue Tracker, IRC Channel, Community forum, and Solr Query Syntax.

**Solr**

- Dashboard
- Logging
- Core Admin
- Java Properties
- Thread Dump
- books
  - Overview
  - Analysis
  - Dataimport
  - Documents
  - Files
  - Ping
  - Plugins / Stats
  - Query
  - Replication
  - Schema
  - Segments info

**Statistics**

Last Modified: about 23 hours ago  
Num Docs: 5  
Max Doc: 5  
Heap Memory Usage: 11140  
Deleted Docs: 0  
Version: 46  
Segment Count: 5  
Current: ✔

**Instance**

CWD: /opt/solr-8.8.1/server  
Instance: /var/solr/data/books  
Data: /var/solr/data/books/data  
Index: /var/solr/data/books/data/index  
Impl: org.apache.solr.core.NRTCachingDirectoryFactory

**Replication (Leader)**

	Version	.Gen	Size
Leader (Searching)	1707740085202	12	17.44 KB
Leader (Replicable)	-	-	-

**Healthcheck**

Ping request handler is not configured with a healthcheck file.

[Documentation](#) [Issue Tracker](#) [IRC Channel](#) [Community forum](#) [Solr Query Syntax](#)

Activate Windows  
Go to Settings to activate Windows.

Приложение готово к эксплуатации. Переходим на localhost:8081/login, авторизуемся ранее загруженными данными пользователя и работаем с приложением

[←](#) [→](#) [↺](#)

localhost:8081/index

## Upload book information

Choose a file:

Файл не выбран.

## Get info about book

Enter message:

[Logout](#)