













Collabnix

Чит-Лист Docker [Обновлено В 2023 Году]

Контрольная таблица - это краткое изложение важной информации, предназначенной для использования в качестве краткого справочника. Чит-листы часто используются в виде списка или таблицы, и они обычно охватывают определенную тему или предметную область. В контексте Docker контрольная таблица Docker представляет собой краткое изложение часто используемых команд Docker и их опций, а также другую полезную информацию, связанную с Docker.

Контрольные таблицы могут быть особенно полезны при изучении нового инструмента или технологии, поскольку они предоставляют удобный способ быстрого поиска и напоминания о ключевых концепциях и командах. Они также могут быть полезны опытным пользователям, которым необходимо вспомнить определенную команду или параметр, но они могут не помнить всех деталей.

Содержание

- Категории
 -  Базовая панель управления Docker
 -  Интерфейс управления контейнерами
 -  Проверка контейнера
 -  Взаимодействие с контейнером
 -  Команды управления изображениями
 -  Команды передачи изображений
 -  Основные команды конструктора
 -  Интерфейс командной строки Docker
 -  Безопасность Docker (Scout, SBOM)
-  Участники
-  Поддержка и сообщество
-  Ссылки

Базовая панель управления Docker

Команда	Описание
<code>docker run</code>	Run a command in a new container
<code>docker build</code>	Build an image from a Dockerfile
<code>docker push</code>	Push an image to a registry
<code>docker pull</code>	Pull an image from a registry
<code>docker stop</code>	Stop a running container
<code>docker rm</code>	Remove one or more containers

Базовая панель управления Docker

Интерфейс управления контейнерами

Команда	Описание
<code>docker create image[command]</code> <code>docker run image[command]</code>	create the container = create + start
<code>docker start container...</code> <code>docker stop container...</code> <code>docker kill container...</code> <code>docker restart container...</code>	start the container graceful ² stop kill (SIGKILL) the container = stop + start
<code>docker pause container...</code> <code>docker unpause container...</code>	Suspend the container Resume the container
<code>docker wait container...</code>	Block until one or more containers stop, then print their exit codes
<code>docker top container...</code>	Display the running processes of a container
<code>docker rm[-f³] container...</code>	destroy the container

Интерфейс управления контейнерами

²отправьте SIGTERM основному процессу + SIGKILL через 10 секунд

^{-f 3} позволяет удалять запущенные контейнеры (= `docker kill` + `docker rm`)

Давайте разберем каждую из этих команд и приведем примеры для лучшего понимания.

1. Создайте контейнер из изображения с помощью `docker run` :

Прежде чем вы сможете выполнять команды внутри контейнера, вам обычно необходимо создать контейнер из изображения. Обычно это делается с помощью `docker run` команды. Вот простой пример:

```
docker run -d --name my_container_name my_image:tag
```

Замените `my_container_name` желаемое имя для вашего контейнера, `my_image:tag` на имя и тег изображения Docker, которое вы хотите использовать.

2. Выполнять команды в запущенном контейнере с помощью `docker exec` :

Вы можете выполнять команды в запущенном контейнере с помощью `docker exec` команды. Синтаксис следующий:

```
docker exec [options] CONTAINER COMMAND
```

Вот пример:

```
docker exec -it my_container_name ls /app
```

Это приведет к выполнению `ls /app` команды внутри контейнера с именем `my_container_name` .

3. Запустите остановленный контейнер с помощью `docker start` :

Если у вас есть остановленный контейнер, вы можете запустить его снова с помощью `docker start` команды. Вот пример:

```
docker start my_container_name
```

При этом запустится контейнер с именем `my_container_name` .

4. Остановите запущенный контейнер с помощью `docker stop` :

Вы можете остановить запущенный контейнер с помощью `docker stop` команды. Вот пример:

```
docker stop my_container_name
```

Это изящно остановит контейнер с именем `my_container_name`.

Помните, что вам нужно заменить `my_container_name` фактическое имя вашего контейнера. Кроме того, вы можете добавить в `-i` команду такие флаги, как `-t` (интерактивный) и `docker exec (tty)`, чтобы сделать взаимодействие более удобным для пользователя, особенно при выполнении команд, требующих пользовательского ввода.

Для более расширенного использования и опций вы можете обратиться к официальной документации Docker:

- [Запуск Docker](#)
- [Docker exec](#)
- [Запуск Docker](#)
- [Остановка Docker](#)

Проверка контейнера

Вот список основных команд Docker, которые помогут вам легко проверять контейнеры:

Команда	Описание
<code>docker ps</code>	<code>list running container</code>
<code>docker ps -a</code>	<code>list all containers</code>
<code>docker logs [-f⁶] container</code>	<code>Show the container output(stdout+stderr)</code>
<code>docker diff container</code>	<code>Show the differences with the image (modified files)</code>
<code>docker top container...</code>	<code>Display the running processes of a container</code>
<code>docker inspect --format '{{.Config.Image}}' my_container_name..</code>	<code>Inspect a container and extract specific information using --format :</code>

Интерфейс управления контейнерами

Взаимодействие с контейнером

Вы хотите знать, как получить доступ к контейнерам? Ознакомьтесь с этой основной командой ниже

Команда	Описание
<code>docker attach container</code>	attach to a running container(stdin/stdout/stderr)
<code>docker cp container:path hostpath/</code>	copy files from the container
<code>docker cp hostpath/ - container: path</code>	copy files into the container
<code>docker export container</code>	export the content of the container (<i>tar archive</i>)
<code>docker exec container args ...</code>	run a command in an existing container (useful for debugging)
<code>docker wait container...</code>	wait until the container terminates and return the exit code
<code>docker commit container image</code>	commit a new docker image (snapshot of the image)

Взаимодействие с контейнером

Команды управления изображениями

Docker предоставляет набор команд для эффективного управления образами Docker. Вот список некоторых важных команд управления образами Docker:

Команда	Описание
<code>docker images</code>	list all local images
<code>docker history image</code>	show the image history(list of ancestors)
<code>docker inspect image...</code>	show level-info's(in json format)
<code>docker tag image tag</code>	tag an image
<code>docker commit container image</code>	Create an image(from a container)
<code>docker import url/ -[tag]</code>	Create an image(from a tarball)
<code>docker rmi image.. .</code>	delete images

Команды управления изображениями

Команды передачи изображений

Для передачи образов Docker между различными средами или системами у вас есть несколько вариантов. Вот несколько часто используемых методов передачи образов Docker:

Использование registry API	Описание
<code>docker push repo[:tag]...</code>	push an image /repository from a registry
<code>docker pull repo[:tag]..</code>	pull an image /repository from a registry
<code>docker search text</code>	search an image on the official registry
<code>docker login ...</code>	login to a registry
<code>docker logout ...</code>	logout from a registry
<code>docker export <container_id> > image.tar</code>	exporting: You can export a Docker image as a tarball file using the <code>docker export</code> command
<code>docker import image image.tar <image_name>:<tag></code>	importing: On the target system, you can import the Docker image from the tarball using the <code>docker import</code> command

Команды передачи изображений

Перенос вручную	Описание
<code>docker save repo/ [:tag]...</code>	export an image/repo as a tarball
<code>docker load</code>	load images from a tarball
<code>docker-ssh¹⁰</code>	proposed script to transfer images between two daemons over ssh

Продолжение '

Основные команды конструктора

Хотите знать, как создать образ Docker? Ознакомьтесь со списком команд для создания образа ниже:

Команда	Описание
<code>FROM image/scratch</code>	base image for the build
<code>MAINTAINER email</code>	name of the maintainer(metadata).
<code>COPY path dst</code>	copy path from the context into the container at location destination(dst)
<code>ADD src dst</code>	same as <code>COPY</code> but untar archives and accepts https urls
<code>RUN args..</code>	run an arbitrary command inside the container
<code>USER name</code>	set the default username
<code>WORKDIR path</code>	set the default working directory
<code>CMD args..</code>	set the default command

Команда	Описание
<code>ENV name value</code>	set an environment variable

Основные команды конструктора

В Docker “шаблон построения” – это метод, используемый для создания оптимизированных и эффективных образов Docker путем отделения среды сборки от среды выполнения. Она включает в себя использование многоступенчатого процесса сборки, где один этап используется для сборки приложения, а другой этап используется для упаковки приложения для выполнения. Этот шаблон помогает уменьшить размер конечного образа Docker и гарантирует, что будут включены только необходимые зависимости времени выполнения.

Вот основные команды, используемые в шаблоне builder в Docker:

1. **ОТ: FROM** Инструкция определяет базовый образ для этапа сборки. Она настраивает среду сборки и содержит все инструменты и зависимости, необходимые для сборки приложения.

```
FROM base_image_for_building AS build_stage
```

2. **ВЫПОЛНИТЬ: RUN** инструкция выполняет команды внутри контейнера этапа сборки. Она используется для установки зависимостей, компиляции кода или выполнения любых других задач, связанных со сборкой.

```
RUN apt-get update && apt-get install -y build-essential
RUN npm install
RUN go build -o my_app
```

3. **КОПИРОВАТЬ / ДОБАВЛЯТЬ: COPY** or **ADD** используется для копирования файлов из локального каталога (context) в контейнер этапа сборки.

```
COPY . /app
```

4. **WORKDIR: WORKDIR** инструкция устанавливает рабочий каталог внутри контейнера этапа сборки, где будут выполняться последующие команды.

```
WORKDIR /app
```

5. **ИЗ (второго этапа)**: после завершения этапа сборки вы используете другую **FROM** инструкцию для запуска нового этапа для среды выполнения. Этот новый этап начинается с базового образа меньшего размера, обычно содержащего только необходимые зависимости времени выполнения.

```
FROM base_image_for_runtime AS runtime_stage
```

6. **КОПИРОВАТЬ из**: **COPY --from** инструкция копирует файлы из контейнера этапа сборки в контейнер этапа выполнения.

```
COPY --from=build_stage /app/my_app /app/my_app
```

7. **CMD / ENTRYPOINT**: В инструкции **CMD** от **ENTRYPOINT** указывается команда по умолчанию или точка входа, которая будет выполняться при запуске контейнера. Она используется на этапе выполнения для определения приложения, которое должно выполняться при запуске контейнера.

```
CMD ["/app/my_app"]
```

Вот полный пример многоступенчатого файла Dockerfile с использованием шаблона builder:

```
# Build stage
FROM golang:1.16 AS build_stage
WORKDIR /app
COPY . .
RUN go build -o my_app

# Runtime stage
FROM alpine:latest AS runtime_stage
WORKDIR /app
COPY --from=build_stage /app/my_app /app/my_app
CMD ["/app/my_app"]
```


В этом примере на первом этапе используется образ языка программирования Go в качестве базового образа для сборки приложения, компилируется код Go и генерируется `my_app` исполняемый файл. На втором этапе образ Alpine Linux используется в качестве базового образа для среды выполнения, а `my_app` двоичный файл копируется с первого этапа. Окончательный образ будет включать только скомпилированный `my_app` двоичный файл, а не всю среду сборки, в результате чего получится уменьшенный по размеру и более оптимизированный для выполнения образ.

Интерфейс командной строки Docker

Ниже приведено краткое описание некоторых основных команд Docker CLI, связанных с управлением изображениями и операциями с контейнерами. Ниже я расширил описания каждой команды:

1. Сборка Docker:

```
docker build [options] -t "app/container_name" .
```

Эта команда используется для создания образа Docker из Dockerfile, который представляет собой текстовый файл, содержащий инструкции по определению образа. `-t` Опция используется для указания тега или имени для созданного изображения. `.` В конце команды ссылается на контекст сборки, то есть на текущий каталог, в котором находится файл Dockerfile.

2. Запуск Docker:

```
docker run [options] IMAGE
```

Эта команда используется для запуска команды или приложения в контейнере Docker на основе указанного образа. Она создает новый контейнер из изображения и выполняет команду по умолчанию, указанную в изображении, или любую команду, предоставленную в качестве аргументов `docker run` команды. Это один из основных способов запуска контейнера.

3. Создать Docker:

```
docker create [options] IMAGE
```

docker create Команда создает новый контейнер из указанного изображения, но не запускает контейнер. Она подготавливает контейнер к запуску с помощью **docker start** команды. Вы можете использовать эту команду для создания контейнера, а затем запустить его позже.

4. Управление изображениями:

Docker предоставляет несколько команд для управления изображениями:

- **docker images** : Перечислены все образы Docker, доступные в вашей системе.
- **docker pull** Загружает образы Docker из реестра.
- **docker push** Помещает изображения Docker в реестр (например, Docker Hub) для совместного использования с другими .
- **docker rmi** Удаляет образы Docker из вашей системы .

4. Управление контейнерами:

Docker предоставляет несколько команд для управления контейнерами:

- **docker ps** : Перечислены все запущенные контейнеры.
- **docker ps -a** Содержит список всех контейнеров (включая остановленные).
- **docker start** Запускает один или несколько остановленных контейнеров .
- **docker stop** Останавливает один или несколько запущенных контейнеров.
- **docker restart** : Останавливает, а затем запускает контейнер .
- **docker rm** Удаляет один или несколько контейнеров (используйте с осторожностью).

Эти команды представляют собой лишь основы работы с образами и контейнерами Docker. Docker предоставляет богатый набор команд и опций для взаимодействия с контейнерами, сетями, томами и многим другим. Вы всегда можете ознакомиться с официальной документацией Docker для получения более подробной информации о каждой команде и ее параметрах.

Чтобы запустить команду в контейнере Docker с помощью **docker create** команды, сначала необходимо создать контейнер из изображения. **docker create** Команда создает контейнер, но не запускает его. Она возвращает идентификатор контейнера, который вы можете использовать для управления контейнером.

Вот основной синтаксис `docker create` команды:

```
docker create [options] IMAGE [COMMAND] [ARG...]
```

Давайте разберем варианты:

- `-a` , `--attach` : Прикрепите стандартный вывод / STDERR и передайте сигналы.
- `-i` , `--interactive` : Держите стандартный интерфейс открытым, даже если он не подключен.
- `--name NAME` : Присвойте имя контейнеру.
- `-p` , `--publish 5000: 5000` : Опубликовать порт контейнера на хосте. В этом примере порт 5000 из контейнера будет сопоставлен с портом 5000 на хосте.
- `--expose 5432` : Предоставьте порт внутренне связанным контейнерам. При этом порт не публикуется на главном компьютере.
- `-P` , `--publish-all` : Опубликовать все открытые порты на хосте.
- `--link container: alias` : Свяжите контейнер с другим контейнером с необязательным псевдонимом.
- `-v` , `--volume 'pwd:/app` : Смонтировать каталог узла в контейнер. В этом примере текущий рабочий каталог подключается к `/app` каталогу в контейнере.
- `-e` , `--env NAME=hello` : Установите переменную среды внутри контейнера.

Вот пример использования `docker create` команды:

```
docker create -a -i --name my_container -p 5000:5000 --expose 5432 -P --link db_container:db_alias -v "$(pwd):/app" -e NAME=hello my_image
```

- `-a -i` : Прикрепите стандартный вывод / STDERR и оставьте STDIN открытым для интерактивного сеанса .
- `--name my_container` : Присвойте созданному контейнеру имя "my_container".
- `-p 5000:5000` : Опубликуйте порт 5000 из контейнера на порт 5000 на хосте.
- `--expose 5432` : Предоставьте внутренний порт 5432 для подключения к другим контейнерам .
- `-P` : Опубликовать все открытые порты на хосте .
- `--link db_container : db_alias` : Свяжите контейнер с другим контейнером с именем "db_container" с псевдонимом "db_alias".
- `-v "$(pwd):/app"` : Смонтируйте текущий рабочий каталог на хосте в каталог `/app` в контейнере.

- `-e NAME=hello` : Установите переменную среды "NAME" со значением "hello" внутри контейнера.
- `my_image` : Имя образа Docker, на основе которого будет создан контейнер.

Пожалуйста, обратите внимание, что некоторые опции, такие как `--link`, устарели в последних версиях Docker, и вместо этого рекомендуется использовать пользовательские сети для контейнерной связи.:

Variable Name	Value
-----	-----
VAR1	value1
VAR2	value2
VAR3	value3

Замените `VAR1`, `VAR2` и `VAR3` фактическими именами ваших переменных среды, а `value1`, `value2` и `value3` их соответствующими значениями.

Помните, что для выполнения `docker create` команды в вашей системе должен быть установлен и запущен Docker.

Docker Compose

Базовый пример

```
version: '2'
services:
  web:
    build:
      context: ./Path
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./code

  redis:
    image: redis
```

В этой конфигурации:

- Определен **web** сервис, который будет создавать образ с использованием файла `Dockerfile`, расположенного в `./Path` каталоге. Он предоставляет доступ к порту 5000 из контейнера на порт 5000 на хосте и монтирует текущий каталог (`.`) в `/code` внутри контейнера.
- **redis** Определен сервис, который использует официальный образ Redis из Docker Hub.

Пожалуйста, убедитесь, что вы заменили `./Path` фактический путь к вашему файлу `Dockerfile` и обновили любые другие заполнители с указанием сведений о вашем конкретном проекте. Убедитесь, что отступ указан правильно, поскольку YAML чувствителен к отступам.

Ссылки

1. Создание образов:

Для создания образов с помощью Docker Compose вы можете указать **build** раздел в определении сервиса. Вы можете выполнить сборку из файла `Dockerfile` в каталоге (`context`) и при необходимости указать пользовательский файл `Dockerfile`. В качестве альтернативы вы можете напрямую использовать существующий образ.

```
version: '3'
services:
  web:
    build:
      context: ./dir
      dockerfile: Dockerfile.dev
    image: my-custom-image:tag

  database:
    image: postgres:latest
```

В этом примере **web** сервис создается с использованием пользовательского файла `Dockerfile`, расположенного в `./dir` каталоге. Затем результирующее изображение помечается как `my-custom-image:tag`. **database** Сервис использует существующий **postgres** образ.

2. Раскрытие портов:

Чтобы предоставить доступ к портам из контейнеров на хост-компьютере, вы можете использовать **ports** раздел в определении службы. Вы можете указать как гостевой (контейнерный) порт, так и порт хоста.

```
version: '3'
services:
  app:
    image: my-app-image:latest
    ports:
      - "3000"
      - "8000:80"
```

```
#Expose port to linked service (not to host)
Expose :["3000"]
```

В этом примере **app** служба предоставляет доступ к порту 3000 из контейнера к случайному порту на хосте. Она также сопоставляет порт 80 из контейнера с портом 8000 на хосте.

Пожалуйста, обратите внимание, что эти примеры предназначены для иллюстрации структуры и синтаксиса конфигураций Docker Compose. Обязательно замените **my-custom-image:tag** и **my-app-image:latest** реальными названиями изображений, которые вы собираетесь использовать.

Помните, что правильное расположение отступов YAML имеет решающее значение в файлах Docker Compose.

Расширенные возможности

1. Ярлыки:

Метки позволяют прикреплять метаданные к вашим службам Docker, контейнерам и другим объектам. Вот как вы можете определить метки в файле Docker Compose YAML:

```
version: '3'
services:
  web:
    image: my_web_app_image
    labels:
      com.example.description: "Accounting web app"
```

В этом примере **web** сервису присваивается метка **com.example.description** со значением **"Accounting web app"**.

2. DNS-серверы:

Вы можете настроить DNS-серверы для своих служб, используя **dns** опцию. Вот как вы можете определить DNS-серверы в файле Docker Compose YAML:

```
version: '3'
services:
  web:
    image: my_web_app_image
    dns:
      - 8.8.8.8
      - 8.8.4.4
```

В этом примере **web** служба настроена на использование DNS-серверов Google (8.8.8.8 и 8.8.4.4).

3. Устройства:

Вы можете назначить устройства службе, используя **devices** опцию. Вот как вы можете определить устройства в файле Docker Compose YAML:

```
version: '3'
services:
  web:
    image: my_web_app_image
    devices:
      - "/dev/ttyUSB0:/dev/ttyUSB0"
```

В этом примере `web` службе назначается устройство `/dev/ttyUSB0` с хоста, и она будет доступна по тому же пути внутри контейнера.

Не забудьте заменить `my_web_app_image` фактическое название изображения, которое вы используете.

Обязательно сохраните эти конфигурации в `.yaml` файле (например, `docker-compose.yaml`) и используйте `docker-compose` команду для управления своими сервисами:

```
docker-compose up -d # to start your services
docker-compose down # to stop and remove your services
```

Пожалуйста, обратите внимание, что отступ YAML имеет решающее значение. Обязательно поддерживайте надлежащие уровни отступов, как показано в примерах.

Сервисы Docker

1. Для просмотра списка всех служб, запущенных в swarm:

```
docker service ls
```

2. Чтобы увидеть все запущенные службы в стеке Docker:

```
docker stack services STACK_NAME
```

3. Чтобы просмотреть журналы определенной службы в стеке Docker:

```
docker service logs STACK_NAME_SERVICE_NAME
```

4. Для быстрого масштабирования сервиса между квалифицированными узлами в стеке Docker:

```
docker service scale STACK_NAME_SERVICE_NAME=REPLICAS
```


Замените `STACK_NAME` , `SERVICE_NAME` и `REPLICAS` на фактические названия стека Docker, сервиса и желаемого количества реплик соответственно.

Пожалуйста, обратите внимание, что команда для просмотра журналов (`docker service logs`) требует в качестве аргументов как имя стека, так и имя службы. Команду масштабирования (`docker service scale`) следует использовать, когда вы хотите настроить количество реплик (экземпляров) определенной службы в Docker swarm.

Docker Очистить

1. Для очистки или удаления неиспользуемых (висячих) изображений:

```
docker image prune
```

2. Удалить все изображения, которые не используются в контейнерах (включая висячие изображения):

```
docker image prune -a
```

3. Для очистки всей вашей системы (включая неиспользуемые образы, контейнеры, сети и тома):

```
docker system prune
```

4. Как покинуть Docker swarm (если вы являетесь узлом в swarm):

```
docker swarm leave
```

5. Чтобы удалить Docker swarm (удалите все службы и тома, связанные со стеком):

```
docker stack rm STACK_NAME
```

6. Чтобы уничтожить все запущенные контейнеры:

```
docker kill $(docker ps -q)
```

Примечание: Команда “docker swarm покинуть” должна выполняться на рабочем или управляющем узле внутри swarm. Команда “docker stack rm” используется для удаления стека Docker (служб) из кластера swarm. Кроме того, будьте осторожны при использовании команды “docker system prune”, поскольку она удалит все неиспользуемые данные (изображения, контейнеры, сети и тома) из вашей системы, и операция не может быть отменена. Всегда просматривайте, что будет удалено, прежде чем подтверждать действие.

Безопасность Docker (Scout, SBOM)

Предоставленные команды являются частью инструмента под названием “Docker Scout”, используемого для анализа программных артефактов, особенно образов Docker, на предмет уязвимостей. Ниже приведено описание каждой команды:

1. Отображение уязвимостей из архива с сохранением Docker:

Эта команда анализирует уязвимости в образе Docker, сохраненном как архив.

```
docker scout cves [OPTIONS] IMAGE | DIRECTORY | ARCHIVE
```

Пример использования:

```
docker scout cves redis.tar
```

2. Отображение уязвимостей из каталога OCI:

Эта команда анализирует уязвимости в образе Docker, хранящемся в каталоге OCI (Open Container Initiative).

```
skopeo copy --override-os linux docker://alpine oci:redis
```

Примечание: Skopeo – это инструмент для работы с изображениями контейнеров, который можно использовать для копирования изображения из одного места в другое.

3. Экспортируйте уязвимости в JSON-файл SARIF:

Эта команда экспортирует уязвимости, обнаруженные в образе Docker, в JSON-файл

SARIF (формат обмена результатами статического анализа).

```
docker scout cves --format sarif --output redis.sarif.json redis
```

Приведенная выше команда генерирует файл SARIF JSON с именем, `redis.sarif.json` содержащим уязвимости, обнаруженные в `redis` образе Docker.

4. Сравнение двух изображений:

Эта команда сравнивает два образа Docker, чтобы выявить любые различия в уязвимостях между ними.

```
docker scout compare --to redis:6.0 redis:6-bullseye
```

Приведенная выше команда сравнивает уязвимости в `redis:6.0` и `redis:6-bullseye` образах Docker.

5. Отображение краткого обзора изображения:

Эта команда предоставляет краткий обзор уязвимостей в образе Docker.

```
docker scout quickview redis:6.0
```

Приведенная выше команда отображает сводку уязвимостей, обнаруженных в `redis:6.0` образе Docker.

Пожалуйста, обратите внимание, что предоставленные команды предполагают, что у вас установлено и правильно настроено средство "Docker Scout". Это средство может не входить в стандартную команду Docker, поэтому вам может потребоваться установить его отдельно. Кроме того, параметры и возможности инструмента могут меняться со временем, поэтому всегда полезно обратиться к документации инструмента для получения самой последней информации.:

Участники

Присоединяйтесь к нашему активному сообществу из более чем 8000 инженеров DevOps, включая капитана Docker [Аджита Райну](#). Сотрудничайте и вносите свой вклад в расширение возможностей облачного образования и инноваций.

Поддержка и сообщество

Поддержка и сообщество: Взаимодействуйте с [Collabnix](#) на различных платформах. Заходите в блоги сообщества, следите за анонсами, подключайтесь к [slack](#), [Twitter](#), [Dev](#) и общайтесь в [Discord](#).

Ссылки

[Здесь](#) вы найдете множество практических лабораторных работ, руководств и списков кураторов. Изучите [DockerLabs](#), [KubeLabs](#), [Kubetools](#), [DockerTools](#) и другие. Улучшайте свои навыки разработки в DevOps и работы в облаке.

Вклад в Collabnix

Если вы в восторге от облачных технологий и любите делиться своими знаниями, мы приглашаем вас внести свой вклад в Collabnix. Являетесь ли вы опытным профессионалом или только начинаете свой путь, наше сообщество обеспечивает благоприятную среду для сотрудничества и роста.

Присоединяйтесь!

Капитан Docker [Аджит Райна](#) здесь, чтобы познакомить вас с Collabnix. Также вы можете связаться с [Адесоджи Алу](#), чтобы подключиться. И не забудьте изучить его [collabnix.com](#), чтобы отправиться в путешествие обучения и инноваций.

Присоединяйтесь ко мне и всему сообществу Collabnix, когда мы исследуем увлекательный мир DevOps, облачных технологий, искусственного интеллекта и не только. Давайте расширять возможности друг друга, делая шаг за шагом к облачным технологиям.

Поздравляем с предстоящим захватывающим путешествием! 🚀🌟👤

Присоединяйтесь к нашему серверу Discord