Введение в Docker

Docker — это платформа для разработки, доставки и запуска приложений в контейнерах. Контейнеризация позволяет изолировать приложения и их зависимости, обеспечивая консистентность и портативность при развертывании. В этой документации мы рассмотрим основные концепции Docker, процесс установки Docker на НАЙС ОС, создание и управление Docker-образами, а также основные команды Docker CLI. В примерах будет использоваться пакетный менеджер tdnf, который является стандартным для НАЙС ОС.

Основные концепции Docker: образы, контейнеры, Dockerfile

Прежде чем погрузиться в детали установки и использования Docker, важно понять основные концепции, на которых основана эта платформа.

Образы Docker

Docker-образ — это шаблон, содержащий все необходимое для запуска приложения: код, библиотеки, зависимости, конфигурационные файлы и т. д. Образы являются неизменяемыми и состоят из нескольких слоев, каждый из которых представляет собой изменение по сравнению с предыдущим слоем. Образы можно создавать, распространять и использовать для создания контейнеров.

Контейнеры Docker

Контейнер — это экземпляр Docker-образа. Контейнеры изолированы друг от друга и от хостовой системы, но могут взаимодействовать через настроенные сети. Они обеспечивают консистентную среду выполнения, независимо от окружения, в котором они запускаются.

Dockerfile

Dockerfile — это файл конфигурации, содержащий инструкции для сборки Docker-образа. Он определяет, какие компоненты будут включены в образ и как их настроить. Используя Dockerfile, можно автоматизировать процесс создания образов, делая его воспроизводимым и управляемым.

Установка Docker на НАЙС ОС

Для установки Docker на НАЙС ОС используйте пакетный менеджер tdnf:

```
sudo tdnf install docker
sudo systemctl enable --now docker
```

Создание и управление Docker-образами

Создание и управление Docker-образами является ключевым аспектом использования Docker. Рассмотрим, как создать Dockerfile, собрать образ и управлять им.

Создание Dockerfile

Для создания Docker-образа необходимо создать Dockerfile. Пример простого Dockerfile для Node.js приложения:

```
# Используем базовый образ Node.js
FROM node:14

# Создаем директорию для приложения
WORKDIR /usr/src/app

# Копируем package.json и устанавливаем зависимости
COPY package*.json ./
RUN npm install

# Копируем исходный код приложения
COPY . .

# Определяем команду для запуска приложения
CMD ["node", "app.js"]

# Указываем порт, который будет использовать приложение
EXPOSE 8080
```

Сборка Docker-образа

После создания Dockerfile можно собрать образ с помощью команды `docker build`:

```
docker build -t mynodeapp .
```

Эта команда создаст образ с именем `mynodeapp`, используя инструкции из Dockerfile в текущем каталоге.

Запуск Docker-контейнера

Для запуска контейнера из созданного образа используйте команду `docker run`:

```
docker run -d -p 8080:8080 mynodeapp
```

Эта команда запустит контейнер в фоновом режиме, пробросив порт 8080 на хосте к порту 8080 в контейнере.

Управление Docker-образами

Основные команды для управления Docker-образами включают:

- `docker images`: отображает список доступных образов.
- `docker rmi`: удаляет указанный образ.
- `docker tag`: присваивает новый тег существующему образу.

- `docker push`: загружает образ в Docker Registry.

Основные команды Docker CLI

Docker CLI предоставляет множество команд для управления образами, контейнерами, сетями и т. д. Рассмотрим основные команды Docker CLI, используемые в повседневной работе.

Работа с контейнерами

- `docker ps`: отображает список запущенных контейнеров.
- `docker ps -a`: отображает список всех контейнеров.
- `docker start`: запускает остановленный контейнер.
- `docker stop`: останавливает запущенный контейнер.
- `docker restart`: перезапускает контейнер.
- `docker rm`: удаляет контейнер.
- `docker logs`: отображает логи контейнера.
- `docker exec`: выполняет команду внутри запущенного контейнера.

Работа с образами

- `docker build`: собирает образ из Dockerfile.
- `docker pull`: загружает образ из Docker Registry.
- `docker push`: загружает образ в Docker Registry.
- `docker images`: отображает список доступных образов.
- `docker rmi`: удаляет указанный образ.

Работа с сетями

- `docker network ls`: отображает список сетей Docker.
- `docker network create`: создает новую сеть.
- `docker network connect`: подключает контейнер к сети.
- `docker network disconnect`: отключает контейнер от сети.
- `docker network rm`: удаляет сеть.

Работа с хранилищами (Volumes)

- `docker volume ls`: отображает список томов.
- `docker volume create`: создает новый том.
- `docker volume rm`: удаляет том.
- `docker run -v`: монтирует том к контейнеру при его запуске.

Docker Compose

Docker Compose — это инструмент для определения и запуска многоконтейнерных Docker-приложений. Он использует файл `docker-compose.yml` для описания конфигурации приложения и позволяет запускать его с помощью одной команды.

Создание файла docker-compose.yml

Пример файла `docker-compose.yml` для многоконтейнерного приложения:

```
version: '3'
services:
  web:
    image: mynodeapp
    ports:
      - "8080:8080"
    volumes:
      - .:/usr/src/app
  redis:
    image: "redis:alpine"
```

Запуск приложения с Docker Compose

Для запуска приложения используйте команду `docker-compose up`:

```
docker-compose up -d
```

Эта команда запустит все службы, определенные в файле `docker-compose.yml`, в фоновом режиме.

Основные команды Docker Compose

- `docker-compose up`: запускает приложение.
- `docker-compose down`: останавливает и удаляет контейнеры, сети и тома, созданные с помощью `up`.
- `docker-compose ps`: отображает статус контейнеров.
- `docker-compose logs`: отображает логи всех контейнеров.
- `docker-compose exec`: выполняет команду внутри запущенного контейнера.
- `docker-compose build`: собирает образы, определенные в файле `docker-compose.yml`.
- `docker-compose pull`: загружает образы из Docker Registry.
- `docker-compose push`: загружает образы в Docker Registry.

Docker Swarm

Docker Swarm — это нативный оркестратор контейнеров для Docker, позволяющий управлять кластером Docker-движков как единой сущностью. Он обеспечивает балансировку нагрузки, автоматическое масштабирование и управление состоянием контейнеров.

Настройка Docker Swarm

Для инициализации Docker Swarm выполните команду `docker swarm init`:

```
docker swarm init --advertise-addr
```

Эта команда инициализирует текущий Docker-движок как менеджер Swarm-кластера. Для добавления воркеров и менеджеров в кластер используйте команды, предоставленные Docker после инициализации.

Управление службами в Docker Swarm

- `docker service create`: создает новую службу.
- `docker service ls`: отображает список служб.
- `docker service ps`: отображает задачи, связанные со службой.
- `docker service scale`: масштабирует службу.
- `docker service rm`: удаляет службу.

Расширенные возможности Docker

Docker предоставляет множество расширенных возможностей для управления контейнерами и интеграции с другими инструментами и сервисами.

Секреты и конфигурации

Docker позволяет безопасно управлять секретами и конфигурациями, такими как пароли, сертификаты и ключи.

Создание и использование секретов

Для создания секрета используйте команду `docker secret create`:

```
echo "my_secret_password" | docker secret create my_secret -
```

Для использования секрета в службе добавьте его в конфигурацию службы:

```
docker service create --name my_service --secret my_secret my_image
```

Интеграция с CI/CD

Docker интегрируется с различными CI/CD инструментами, такими как Jenkins, GitLab CI и другие, обеспечивая автоматизацию сборки, тестирования и развертывания приложений.

Пример использования Docker в Jenkins

Добавьте следующий шаг в ваш Jenkins Pipeline для сборки Docker-образа:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                script {
                    docker.build('my_image')
                }
            }
        }
    }
}
```

Docker и Kubernetes

Docker часто используется в сочетании с Kubernetes для оркестрации контейнеров в масштабируемых и отказоустойчивых кластерах. Kubernetes управляет развертыванием, масштабированием и обновлением контейнеризованных приложений.

Пример манифеста Kubernetes для развертывания Docker-контейнера

Создайте файл `deployment.yaml` с содержимым:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: my_image
        ports:
        - containerPort: 8080
```

Для развертывания используйте команду `kubectl apply`:

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

Docker предоставляет мощные инструменты для контейнеризации приложений, обеспечивая их консистентность, портативность и легкость развертывания. Изучение основных концепций Docker, установка и настройка Docker на НАЙС ОС, а также создание и управление Docker-образами и контейнерами помогут вам эффективно использовать эту технологию в вашей ИТ-инфраструктуре.