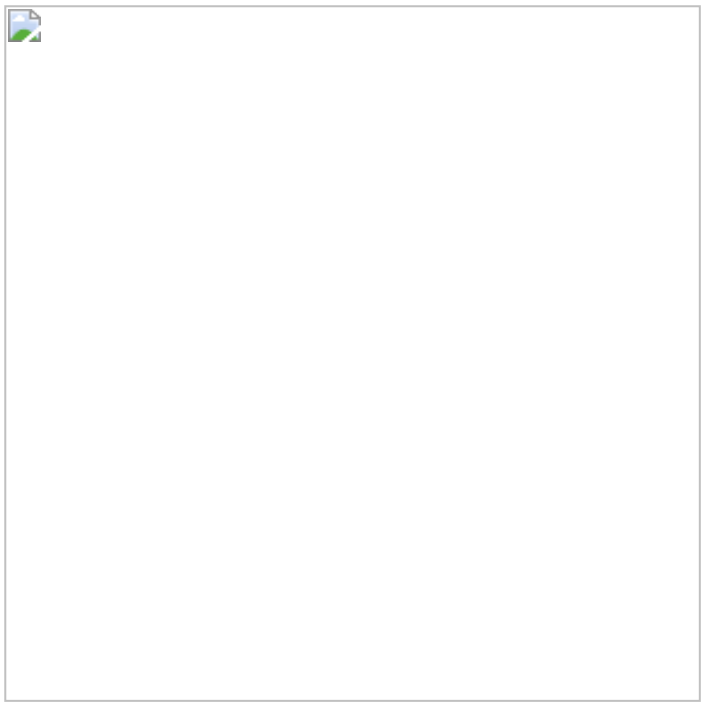
Как использовать Node.js с Docker

dev-notes.ru/articles/devops/node-js-docker

Алексей Лоскутов December 1, 2023



Node.js позволяет создавать быстрые и масштабируемые веб-приложения используя JavaScript как на сервере, так и на клиенте. Ваше приложение может прекрасно работать на машине разработчика, но можете ли вы быть уверены, что оно будет работать на устройствах ваших коллег или на рабочих серверах?

Рассмотрим следующие сценарии:

- Возможно, вы используете macOS, в то время как другие используют Windows, а сервер работает под управлением Linux.
- У вас установлен Node.js 20, а остальные используют другие версии среды исполнения.
- Вы используете такие зависимости, как базы данных, которые могут отличаться или быть недоступны на других платформах.
- Вы уверены, что ваш новый код не может сделать ничего опасного в другой операционной системе (ОС)?

Docker доставляет

Docker помогает решить перечисленные выше проблемы, связанные с “но это работает на моей машине”. Вместо того чтобы устанавливать приложение локально, вы запускаете его в легковесной, изолированной среде, похожей на виртуальную машину, называемой **контейнером**.

Настоящая виртуальная машина эмулирует аппаратное обеспечение компьютера, что позволяет установить ОС. Docker эмулирует ОС для установки приложений. Как правило, на один контейнер с Linux устанавливается одно приложение, и они соединяются виртуальной сетью, чтобы взаимодействовать по портам HTTP.

Преимущества:

- Ваша Docker-установка может либо эмулировать рабочий Linux-сервер, либо развёртываться с помощью контейнеров.
- Загрузка, установка и настройка зависимостей занимает считанные минуты.
- Ваше приложение с контейнерами работает одинаково на всех устройствах.
- Это безопаснее. Ваше приложение может повредить ОС контейнера, но это не повлияет на ваш ПК, и вы сможете перезапустить его заново за считанные секунды.

При использовании Docker нет необходимости устанавливать Node.js на ПК или использовать такие варианты управления средой выполнения, как nvm.

Ваш первый скрипт

Установите Docker Desktop на Windows, macOS или Linux, затем создайте небольшой скрипт с именем `version.js` и следующим кодом:

```
console.log(`Node.js version: ${ process.version }`);
```

Если у вас локально установлен Node.js, попробуйте запустить скрипт. Вы увидите результат, подобный этому, если у вас установлена версия 18:

```
$ node version.js
Node.js version: v18.18.2
```

Теперь этот же скрипт можно запустить внутри контейнера Docker. В приведённой ниже команде используется последняя версия Node.js с долгосрочной поддержкой (LTS). Перейдите в каталог сценария и запустите его на macOS или Linux:

```
$ docker run --rm --name version \
  -v $PWD:/home/node/app \
  -w /home/node/app \
  node:lts-alpine version.js

Node.js version: v20.9.0
```

Пользователи Windows Powershell могут использовать аналогичную команду со скобками `{}` вокруг `PWD`:

```
> docker run --rm --name version -v ${PWD}:/home/node/app -w /home/node/app
node:lts-alpine version.js

Node.js version: v20.9.0
```

Первый запуск может занять минуту или две, пока Docker загружает зависимости. Последующие запуски происходят мгновенно.

Попробуем использовать другую версию Node — например, последний релиз версии 21. На macOS или Linux:

```
$ docker run --rm --name version \
  -v $PWD:/home/node/app \
  -w /home/node/app \
  node:21-alpine version.js

Node.js version: v21.1.0
```

В Windows Powershell:

```
> docker run --rm --name version -v ${PWD}:/home/node/app -w /home/node/app
node:21-alpine version.js

Node.js version: v21.1.0
```

Помните, что скрипт выполняется внутри контейнера Linux, в котором установлена определённая версия Node.js.

Пояснение аргументов

Для любознательных аргументы команды следующие:

- `docker run` запускает новый контейнер из *образа* — подробнее об этом далее.
- `--rm` удаляет контейнер при завершении его работы. Нет необходимости сохранять контейнеры, если у вас нет веских причин для их повторного запуска.
- `--name version` присваивает контейнеру имя для более простого управления.
- `-v $PWD:/home/node/app` (или `-v ${PWD}:/home/node/app`) монтирует том. В этом случае текущий том, находящийся непосредственно на хост-компьютере, монтируется внутри контейнера по адресу `/home/node/app`.
- `-w /home/node/app` задаёт рабочий каталог Node.js.
- `node:lts-alpine` — *образ*, в данном случае LTS-версия Node.js, работающая в Alpine Linux. Образ содержит ОС и файлы, необходимые для работы приложения. Рассматривайте его как снимок диска. Из одного и того же образа можно запустить любое количество контейнеров: все они ссылаются на один и тот же набор файлов, поэтому каждый контейнер требует минимум ресурсов.
- `version.js` — это команда для выполнения (из рабочего каталога).

Образы Docker можно получить с Docker Hub, они доступны для приложений и режимов выполнения, включая Node.js. Образы часто доступны в нескольких версиях, обозначаемых тегами, например: `lts-alpine`, `20-bullseye-slim` или просто `latest`.

Обратите внимание, что Alpine — это небольшой дистрибутив Linux с размером базового образа около 5 МБ. Он не содержит большого количества библиотек, но его вполне достаточно для простых проектов, подобных тем, что представлены в данном руководстве.

Запуск сложных приложений

Приведённый выше сценарий `version.js` прост и не содержит зависимостей и этапов сборки. Большинство приложений Node.js используют `npm` для установки и управления модулями в каталоге `node_modules`. Вы не можете использовать приведённую выше команду, потому что:

- Невозможно запустить `npm` на хост-компьютере (возможно, не установлен Node.js или его правильная версия).
- Для некоторых модулей требуются двоичные файлы, специфичные для конкретной платформы. Нельзя установить двоичный файл Windows на хост-компьютер и ожидать, что он будет работать в контейнере Linux.

Решение заключается в создании собственного образа Docker, содержащего:

- соответствующую версия среды выполнения Node.js
- установленную версия вашего приложения со всеми необходимыми модулями

В следующей демонстрации создаётся простое приложение Node.js с использованием фреймворка Express.js. Создайте новый каталог с именем `simple` и добавьте в него файл `package.json` со следующим содержимым:

```
{
  "name": "simple",
  "version": "1.0.0",
  "description": "simple Node.js and Docker example",
  "type": "module",
  "main": "index.js",
  "scripts": {
    "debug": "node --watch --inspect=0.0.0.0:9229 index.js",
    "start": "node index.js"
  },
  "license": "MIT",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

Добавьте файл `index.js` с кодом JavaScript:

```
// приложение Express
import express from 'express';

// конфигурация
const cfg = {
  port: process.env.PORT || 3000
};

// инициализация Express
const app = express();

// маршрут домашней страницы
app.get('/:name?', (req, res) => {
  res.send(`Hello ${ req.params.name || 'World' }!`);
});

// запуск сервера
app.listen(cfg.port, () => {
  console.log(`server listening at http://localhost:${ cfg.port }`);
});
```

Не пытайтесь устанавливать зависимости или запускать это приложение на хост-компьютере!

Создайте файл с именем `Dockerfile` со следующим содержимым:

```
# базовый образ Node.js LTS
FROM node:lts-alpine

# определение переменных среды
ENV HOME=/home/node/app
ENV NODE_ENV=production
ENV NODE_PORT=3000

# создание папки приложения и назначение прав пользователю node
RUN mkdir -p $HOME && chown -R node:node $HOME

# установка рабочего каталога
WORKDIR $HOME

# установка активного пользователя
USER node

# копирование файла package.json с хоста
COPY --chown=node:node package.json $HOME/

# установка модулей приложения
RUN npm install && npm cache clean --force

# копирование оставшихся файлов
COPY --chown=node:node . .

# выставление порта на хосте
EXPOSE $NODE_PORT

# команда запуска приложения
CMD [ "node", "./index.js" ]
```

В нем определены шаги, необходимые для установки и выполнения приложения. Обратите внимание, что `package.json` копируется в образ, затем выполняется `npm install` перед копированием остальных файлов. Это более эффективно, чем копирование всех файлов сразу, поскольку Docker создаёт слой образа при каждой команде. Если файлы приложения (`index.js`) изменятся, Docker нужно будет выполнить только последние три шага; повторный запуск `npm install` не требуется.

В качестве опции можно добавить файл `.dockerignore`. Он аналогичен `.gitignore` и предотвращает копирование ненужных файлов в образ командой `COPY ...` Например:

```
Dockerfile

.git
.gitignore

.vscode
node_modules
README.md
```

Создайте образ Docker с именем `simple`, выполнив следующую команду (*обратите внимание на точку `.` в конце — она означает, что вы используете файлы в текущем каталоге*):

```
docker image build -t simple .
```

Образ должен собраться в течение нескольких секунд, если использованный ранее Docker-образ `node:lts-alpine` не был удалён из вашей системы.

Если сборка прошла успешно, запустите контейнер из образа:

```
docker run -it --rm --name simple -p 3000:3000 simple

server listening at http://localhost:3000
```

Параметр `-p 3000:3000` публикует или выставляет `<хост-порт>` на `<контейнер-порт>`, поэтому порт `3000` на вашем хост-компьютере маршрутизируется на порт `3000` внутри контейнера.

Откройте браузер и введите URL `http://localhost:3000/`, чтобы увидеть "Hello World!".

Попробуйте добавить к URL имена — например, `http://localhost:3000/Craig` — чтобы увидеть альтернативные сообщения.

Наконец, остановите работу приложения, нажав на иконку **stop** на вкладке **Containers** в Docker Desktop или выполнив следующую команду в другом окне терминала:

```
docker container stop simple
```

Улучшенный рабочий процесс разработки Docker

Приведённый выше процесс имеет ряд досадных недостатков:

- Любое изменение кода (в `index.js`) требует остановки контейнера, пересборки образа, перезапуска контейнера и повторного тестирования.
- Нельзя подключить отладчик Node.js, подобный тому, который имеется в VS Code.

Docker может улучшить рабочий процесс разработки, сохранив существующий образ рабочего уровня, но запустив контейнер с переопределениями для выполнения следующих действий:

- Установите переменные среды, такие как `NODE_ENV`, на значение `development`.
- Смонтируйте локальный каталог в контейнер.

- Запустите приложение с помощью команды `npm run debug`. При этом запускается `node --watch --inspect=0.0.0.0:9229 index.js`, который перезапускает приложение при изменении файлов (новое в Node.js 18) и запускает отладчик с запросами, разрешёнными извне контейнера.
- На хосте будут открыты порт приложения `3000` и порт отладчика `9229`.

Это можно сделать с помощью одной длинной команды `docker run`, но я предпочитаю использовать Docker Compose. Он устанавливается вместе с Docker Desktop и часто используется для запуска нескольких контейнеров. Создайте новый файл с именем `docker-compose.yml` со следующим содержимым:

```
version: '3'

services:

  simple:
    environment:
      - NODE_ENV=development
    build:
      context: ./
      dockerfile: Dockerfile
    container_name: simple
    volumes:
      - ./:/home/node/app
    ports:
      - "3000:3000"
      - "9229:9229"
    command: /bin/sh -c 'npm install && npm run debug'
```

Запустите приложение в режиме отладки с помощью:

```
docker compose up

[+] Building 0.0s
[+] Running 2/2
 ✔ Network simple_default  Created
 ✔ Container simple        Created
Attaching to simple
simple  |
simple  | up to date, audited 63 packages in 481ms
simple  |
simple  | > simple@1.0.0 debug
simple  | > node --watch --inspect=0.0.0.0:9229 index.js
simple  |
simple  | Debugger listening on ws://0.0.0.0:9229/de201ceb-5d00-1234-8692-
8916f5969cba
simple  | For help, see: https://nodejs.org/en/docs/inspector
simple  | server listening at http://localhost:3000
```

Обратите внимание, что старые версии Docker Compose представляют собой Python-скрипты, запускаемые с помощью `docker-compose`. В новых версиях функциональность Compose интегрирована в основной исполняемый файл, поэтому он запускается с помощью `docker compose`.

Перезапуск приложений в реальном времени

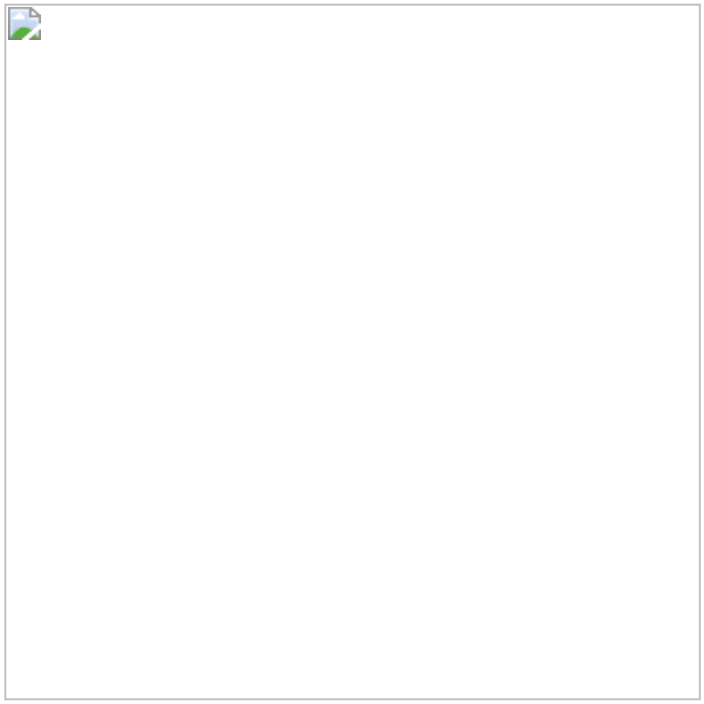
Откройте файл `index.js`, внесите изменения (например, строку в строке 14) и сохраните файл, чтобы увидеть автоматический перезапуск приложения:

```
simple  | Restarting 'index.js'
simple  | Debugger listening on ws://0.0.0.0:9229/acd16665-1399-4dbc-881a-
8855ddf9d34c
simple  | For help, see: https://nodejs.org/en/docs/inspector
simple  | server listening at http://localhost:3000
```

Для просмотра обновления откройте или обновите браузер по адресу `https://localhost:3000/`.

Отладка в VS Code

Откройте панель VS Code **Run and Debug** и кликните кнопку **create a launch.json file**.

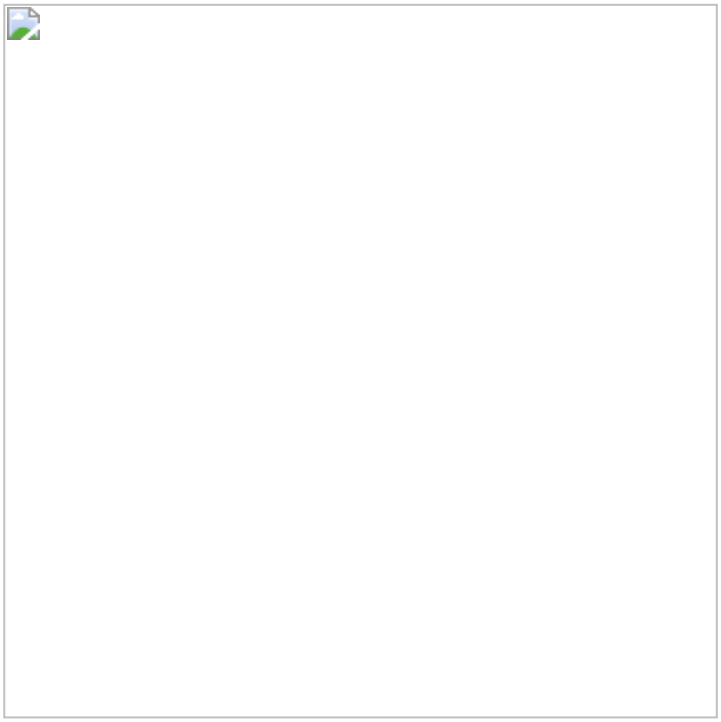


Панель выполнения и отладки VS Code

В выпадающем списке выберите **Node.js**, после чего будет создан и открыт в редакторе файл `.vscode/launch.json`. Добавьте следующий код, который присоединяет отладчик к запущенному контейнеру:

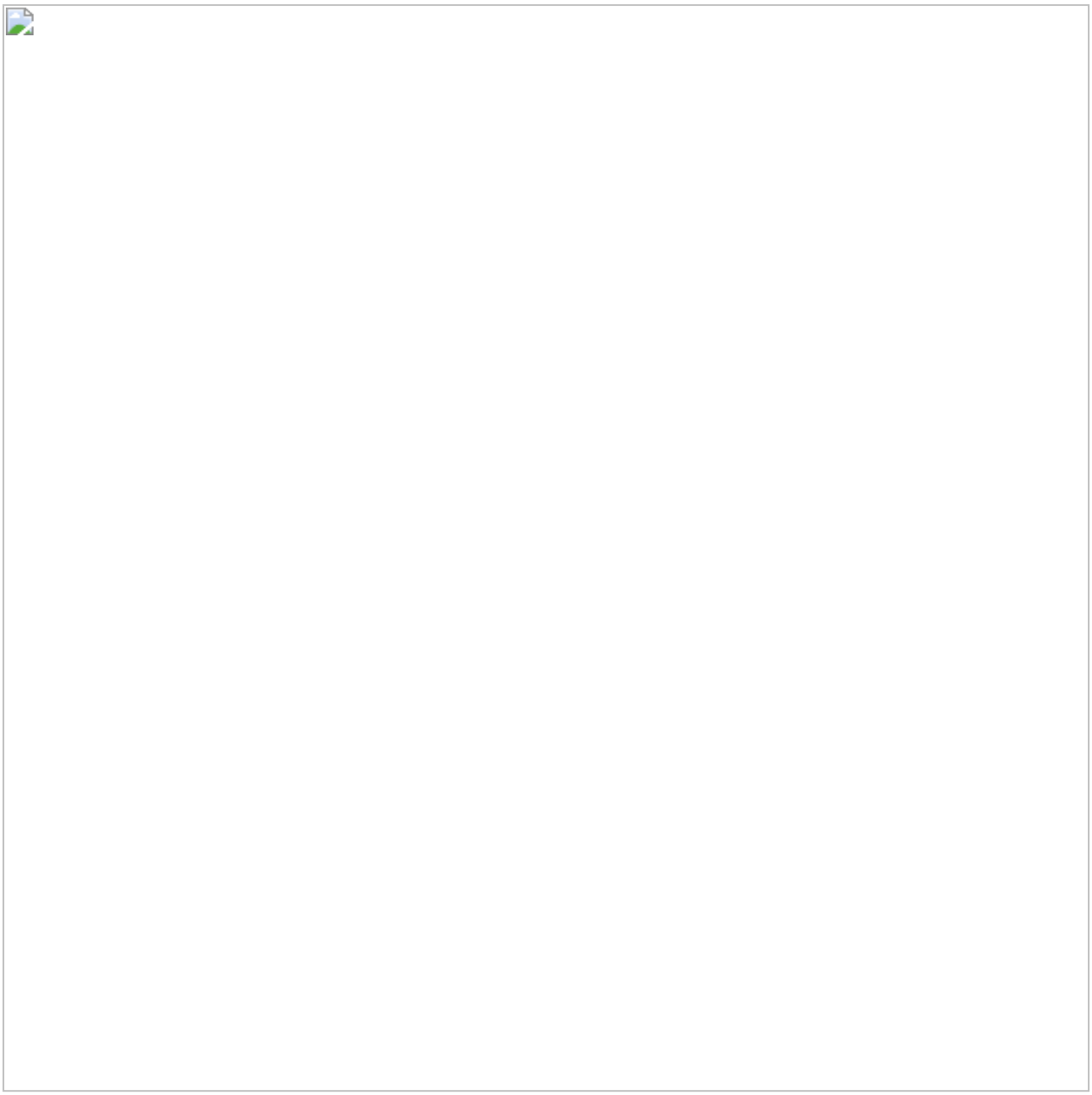
```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "attach",
      "name": "Attach to Container",
      "address": "localhost",
      "port": 9229,
      "localRoot": "${workspaceFolder}",
      "remoteRoot": "/home/node/app",
      "skipFiles": [
        "<node_internals>/**"
      ]
    }
  ]
}
```

Сохраните файл и кликните **Attach to Container** в верхней части панели Debug, чтобы начать отладку.



Запуск и отладка VS Code

Появится панель инструментов отладки. Переключитесь на файл `index.js` и добавьте точку останова на строке 14, щёлкнув по каёмке, чтобы появилась красная точка.



Установка точки останова в VS Code

Обновите `https://localhost:3000/` в браузере, и VS Code остановит выполнение в точке останова, и покажет состояние всех переменных приложения. Кликните на пиктограмму на панели инструментов отладки, чтобы продолжить выполнение, пройти по коду или отключить отладчик.

Остановка контейнера

Остановите работающий контейнер, открыв другой терминал. Перейдите в каталог приложений и введите:

```
docker compose down
```

Заключение

Хотя Docker требует некоторого времени на первоначальную настройку, долгосрочные преимущества надёжного, распространяемого кода с лихвой перекрывают эти усилия. Docker приобретает неоценимое значение при добавлении дополнительных зависимостей, таких как базы данных.