Санкт-Петербургский политехнический университет Петра Великого
Институт Компьютерных Наук и Технологий
Кафедра «Высшая школа программной инженерии»

Курсовой проект

по дисциплине: «Микропроцессорные системы»

Выполнил
студент гр.33534/21 _____ С.А. Фомин

Проверил
преподаватель _____ С.К. Круглов

Санкт-Петербург
2018

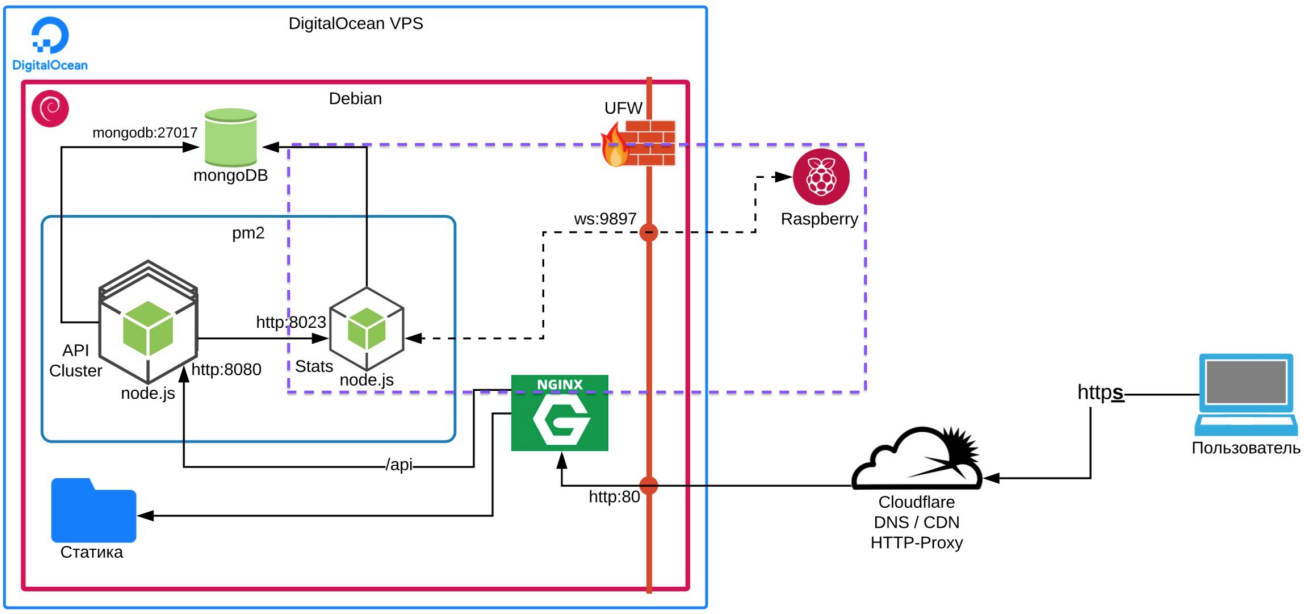

Рис. 1: Схема архитектуры крупным планом.

Постановка задачи

Разработать клиент-серверное приложение для вывода на внешний жидкокристаллический дисплей (с помощью Raspberry Pi) количества сокращенных ссылок из базы данных сервиса short.taxnuke.ru в режиме реального времени.

Ход работы

На рисунке 1 изображена схема сервиса short.taxnuke.ru в целом, а также интегрированного в него разрабатываемого клиент-серверного приложения (в пунктирной фиолетовой рамке).

В качестве программной среды было принятно решение использовать Node.js - кроссплатформенное решение, использующее в качестве JavaScript-движка V8, написанный на C. Для Node.js имеется множество готовых библиотек на JavaScript с привязками к нативным модулям на C++ и C и с многопоточностью. Выбор обусловлен тем, что изначально сервис short.taxnuke.ru тоже написан Node.js и использоание одной программной среды и языка позволяет минимизировать количество зависимостей и упростить интерфейс между компонентами. Также Node.js позволяет порождать дочерние процессы и передавать в них аргументы командной строки, управляющие последовательности и использовать потоки ввода/вывода.

Проработка архитектуры и регламента взаимодействия между сервисами

В первую очередь необходимо было продумать, какие механизмы и средства будут использоваться для взаимодействия сервисом между собой и сервиса статистики и его потребителей. В качестве протокола для общения сервиса статистики и его клиентов было решено использовать websocket, чтобы избежать опрашивания основного сервиса (даже "длинного"опрашивания), которое неизбежно повлекло бы за собой ухудшение производительности основного сервиса при количестве клиентов (потребителей статистики оного), стремящемся к бесконечности.

В качестве механизма взаимодействия сервисов между собой был выбран webhook, то есть один сервис будет отправлять http-запрос любым методом в любой эндпоинт другого сервиса с целью просто "дернуть"его и уведомить о том, что данные обновились. Было решено также не передавать с webhook никаких данных от одного сервиса к другому, это позволило избежать лишних развертываний основного сервиса по причинам изменений в интерфейсе сервиса статистики или формата данных.

Вместо этого, сервис статистики открывает свое личное соединение с базой данных и по пришествии webhook сам параллельно отправляет к базе запрос и реализует там своими средствами всю бизнес логику, инкапсулируя ее от основного сервиса.

Создание фундамента

Было принято решение использовать монорепозиторий git для управления контролем версий проекта, ввиду того, что логически это один т.н. микросервис, в дополнение к тому, что клиент и сервер изоморфны и используют одни библиотеки и написаны на одном языке.

В качестве линтера (утилиты контроля стиля кода) используется eslint, его конфигурационный файл можно увидеть в приложении 4.

В качестве системы доставки и развертки (деплоя) используется стандартная для Node.js связка из pm2 и git. pm2 на машине-разработчике подключается к машине-исполнителю по ssh, выполняет git clone, разворачивает проект из исходого кода и переставляет символьную ссылку на последний релиз, а так же берет на себя управление переменными окружения. Конфигурационный файл pm2 можно увидеть в приложении 3, он декларативен и не вызывает затруднений для понимания.

Разработка сервиса-сервера

В качестве библиотеки для работы с websocket была выбрана самая популярная - "ws". Драйвер для общения с базой данных MongoDB было решено использовать официальный - "mongodb".

При первом старте, сервер пытается подключиться к базе данных, если все произошло успешно, то создается http-сервер, который слушает на порт из переменной окружения, и аналогично далее создается websocket-сервер.

При установлении нового соединения, сервер websocket отпрвляет тестовые данные для подтверждения соединения, и через 3 секунды отправляет клиенту актуальную информацию о статистике. Далее, websocket-сервер оповещает своих клиентов только после получения webhook от основного сервиса, при этом обновляя информацию о статистике.

Исходный код сервера находится в приложении 1.

@0 - подключение библиотек, оформленных как CommonJS-модули, синтаксис ES2015-моделй не используется по причине отсутствия поддержки такого синтаксиса более старыми версиями Node.

@1 - установление соединения с базой данных MongoDB посредствам официального нативного драйвера, функциональная природа JavaScript позволяет в одну строку подключить библиотеку, обратиться к синглтону клиента БД, и вызвать функцию подключения, которая в свою очередь возвращает Promise - специальный объект для упрощения работы с асинхронным кодом (чтобы не использовать вложенные коллбэки).

Далее после успешного подключения происходит создание экземпляра http-сервера, и websocket-сервера, @2, первый используется для прослушивания вебхуков (легковесных http-запросов в роли уведомлений). Порты оба сервера получают из переменных окружения, выставлением которых занимается утилита развертывания, встроенная в pm2.

На этапе @3 происходит добавление метода вещания для сервера websocket. Эта функция по сути рассылает всем активным клиентам данные, которые в нее передали.

В @4 происходит создание нового обработчика события при подключении нового клиента. Клиенту отправляется тестовое сообщение и через 3 секунды последние актуальные данные.

На @5 выполняет навешивание обработчика запросов любым методом в любой эндпоинт http-сервера.

Секция @6 отвечает за обращение к базе с целью подсчета сокращенных ссылок, далее формируется объект и он передается всем клиентам websocket-сервера.

На шаге @7 происходит начало просшуливания http-сервером веб-хуков на порту из переменной окружения HOOK_PORT.

Разработка сервиса-клиента

В качестве библиотеки для работы с websocket была также выбрана "ws". При старте, сервис-клиент пытается установить соединение с сервисом-сервером по протоколу websocket и далее выводит информацию на жидкокристаллический экран с помощью библиотеки "lcdi2c которая позволяет абстрагироваться от низкоуровневой логики общения по протоколу I2C. Данную бибиотеку представляется возможным использовать благодаря её поддержки расширителя портов PCF8574, который был преобретен вместе с самим жидкокристаллическим экраном (16x2) на Hitachi HD44780 контроллере. Все библиотеки (пакеты) для Node.js можно найти на ресурсе npmjs.com, далее они подключаются как модули через функцию require.

Исходный код клиента находится в приложении 2.

@0 - подключение библиотек, оформленных как CommonJS-модули, синтаксис ES2015-моделй не используется по причине отсутствия поддержки такого синтаксиса более старыми версиями Node.

@1 - задание константы с адресом I2C чипа, для определения этого адреса, нужно подключиться к Raspberry Pi по SSH или другим удобным способом и дать команду i2cdetect 1.

@2 - создание экземпляра соединения с дисплеем, передаются аргументы: поколение Raspberry Pi, адрес I2C, ширина экрана и количество строк на нем.

@3 - создание нового вебсокет-соединения, подключение к хосту и порту из переменных окружения WS_HOST и WS_PORT соответственно.

@4 - навешивание обработчика на событие открытия соединения ws.

@5 - навешивание обработчика на событие закрытия соединения ws.

@6 - навешивание обработчика на событие получения нового сообщения по ws. В случае успешного разбора сериализованных данных, они выводятся на экран, иначе в поток ошибок и на экран идет сообщение об ошибке.

Деплой (развертка) проекта на конечных устройствах

При внесении каких-либо изменений в проект, для их вступления в силу на реальных устройствах-носителях, необходимо выполнить процедуру деплоя, или "развертки". В качестве деплоера используется pm2, также обеспечивающий кластерный режим работы приложения и демонизацию (daemon режим работы).

В первую очередь необходимо создать новый коммит с изменениями и выполнить пуш в удаленный репозиторий. Важно, чтобы конечные устройства имели свои SSH-ключи, которые добавлены в список доверенных для работы с удаленным репозиторием. После того, как эти шаги выполнены, сценарий деплоя различается в зависимости от того, деплой чего нужно произвести, клиентского сервиса или серверного.

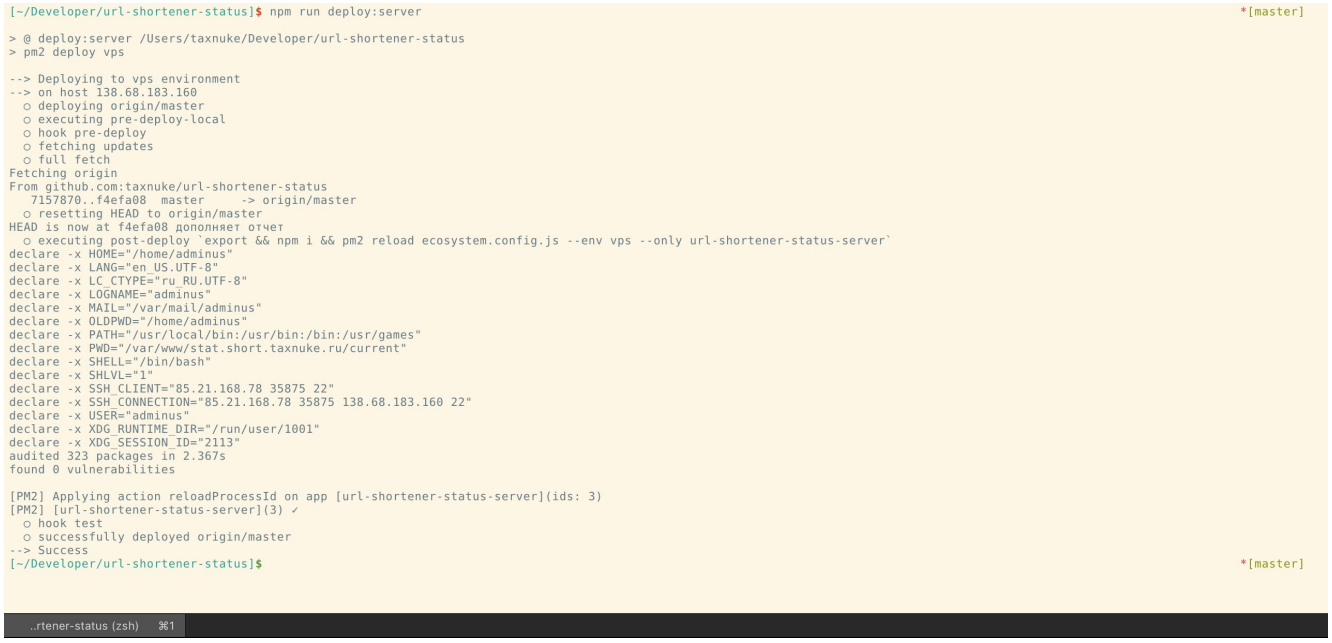

Рис. 2: Ввод команды для деплоя и результат.

VPS-сервер DigitalOcean

В случае, если нужно задеплоить серверный сервис, из директории проекта нужно дать команду npm run deploy:server, или pm2 deploy vps, одна является псевдонимом другой. В результате выполнения этой команды, pm2 считает конфигурационный файл ecosystem.config.js, по содержимому которого будет определено, под каким пользователем нужно подключиться на VPS-сервер, из какого репозитория и куда сделать клонирование, и что сделать для самой развертки. Так же будут автоматически выставлены переменные окружения и выполнен бесшовный перезапуск приложения.

Пример деплоя на сервер со снимками экрана приведен на рисунке 2.

Raspberry Pi

В случае, если нужно задеплоить клиентский сервис, то из директории проекта нужно дать команду npm run deploy:client, или pm2 deploy raspberry, одна является псевдонимом другой. Далее будут выполнены те же действия, что и в результате деплоя на VPS, с тем лишь различием, что деплой на Raspberry в данном случае возможен лишь из локальной сети, так как ip в конфигурационном файле указан локальный, это сделано из соображений безопасности. Доступ к websocket-порту возможен также только с ip моего домашнего NAT, это контролируется фаерволом UFW.

Результат работы

В результате выполнения курсового проекта было разработано серверное и клиентское приложения, работающие в паре и выполняющие поставленную задачу.

ПРИЛОЖЕНИЕ 1. Код сервера

```
/**
 * @0
 */
const WebSocket = require('ws')
const hookPort = process.env.HOOK_PORT

/**
 * @1
 */
const url = `mongodb://localhost:${process.env.MONGO_PORT}`
require('mongodb').MongoClient
  .connect(url)
  .then(mongoConnection => {
    let latestData = null

    /**
     * @2
     */
    const httpServer = require('express')()
    const websocketServer = new WebSocket.Server({
      port: process.env.WS_PORT
    })

    /**
     * @3
     */
    websocketServer.broadcast = function broadcast(data) {
      console.info(`broadcasting ${JSON.stringify(data)}...`)

      websocketServer.clients.forEach(function each(client) {
        if (client.readyState === WebSocket.OPEN) {
          client.send(data)
        }
      })
    }

    /**
     * @4
     */
    websocketServer.on('connection', function connection(ws) {
      ws.send('hi')

      setTimeout(() => {
        ws.send(latestData)
      }, 3000)
    })

    /**
```

```
    * @5
    */
  httpServer.use('*', (req, res) => {

    /**
     * @6
     */
    mongoConnection.db('url-shortener').collection('aliases')
      .countDocuments()
      .then(count => {
        res.end()

        const data = JSON.stringify({
          link_count: count
        })

        websocketServer.broadcast(data)

        latestData = data
      })
      .catch(console.error)
  })

  /**
   * @7
   */
  httpServer.listen(hookPort)
    .on('listening', () => {
      console.log('proverka')
      console.log(`HTTP server listening on port ${hookPort}`)
    })
})
.catch(console.error)
```

ПРИЛОЖЕНИЕ 2. Код клиента

```
/**
 * @0
 */
const I2CLCDConnection = require('lcdi2c')
const WebSocket = require('ws')

/**
 * @1
 */
const I2C_ADDR = 0x3f

/**
 * @2
 */
const lcdConnection = new I2CLCDConnection(1, I2C_ADDR, 16, 2)
lcdConnection.println('short.taxnuke.ru', 1)

/**
 * @3
 */
const ws = new WebSocket(
  `ws://${process.env.WS_HOST}:${process.env.WS_PORT}`
)

/**
 * @4
 */
ws.on('open', () => {
  lcdConnection.println('connected', 2)
  console.info('WebSocket connection established')
})

/**
 * @5
 */
ws.on('close', function close() {
  console.warn('WebSocket connection closed')
  lcdConnection.println('conn closed', 2)
})

/**
 * @6
 */
ws.on('message', data => {
  console.log(data)

  lcdConnection.clear()
```

```
  try {
    data = JSON.parse(data)
  } catch (e) {
    console.error(data)
  } finally {
    lcdConnection.println('short.taxnuke.ru', 1)
    lcdConnection.println(
      `links: ${data.link_count || 'Error =('}`,
      2
    )
  }
})
```

ПРИЛОЖЕНИЕ 3. Содержимое конфигурационного файла для развертки

```
module.exports = {
  // Options reference: https://pm2.io/doc/en/runtime/reference/ecosystem-f
  apps: [
    {
      /**
       * SERVER
       */
      name: 'url-shortener-status-server',
      script: 'source/server.js',
      env: {
        watch: true,
        HOOK_PORT: 8813,
        WS_PORT: 9897,
        MONGO_PORT: 28017,
        NODE_ENV: 'development'
      },
      env_vps: {
        watch: false,
        HOOK_PORT: 8813,
        WS_PORT: 9897,
        MONGO_PORT: 27017,
        NODE_ENV: 'production'
      }
    },
    {
      /**
       * CLIENT
       */
      name: 'url-shortener-status-client',
      script: 'source/client.js',
      env: {
        watch: true,
        WS_PORT: 9897,
        WS_HOST: 'localhost',
        NODE_ENV: 'development'
      },
      env_raspberry: {
        watch: false,
        WS_PORT: 9897,
        WS_HOST: '138.68.183.160',
        NODE_ENV: 'production'
      }
    }
  ],

  deploy: {
    vps: {
```

```
      user: 'adminus',
      host: '138.68.183.160',
      ref: 'origin/master',
      repo: 'git@github.com:taxnuke/url-shortener-status.git',
      path: '/var/www/stat.short.taxnuke.ru',
      'post-deploy': 'npm i && pm2 reload ecosystem.config.js --env vps --c
    },
    raspberry: {
      user: 'pi',
      host: '192.168.1.70',
      ref: 'origin/master',
      repo: 'git@github.com:taxnuke/url-shortener-status.git',
      path: '/var/www/stat.short.taxnuke.ru',
      'post-deploy': 'npm i && pm2 reload ecosystem.config.js --env raspber
    }
  }
}
```

ПРИЛОЖЕНИЕ 4. Конфигурация линтера

```
module.exports = {
  'env': {
    'es6': true,
    'node': true
  },
  'extends': 'eslint:recommended',
  'parserOptions': {
    'ecmaVersion': 2015
  },
  'rules': {
    'indent': [
      'error',
      2
    ],
    'linebreak-style': [
      'error',
      'unix'
    ],
    'quotes': [
      'error',
      'single'
    ],
    'semi': [
      'error',
      'never'
    ],
    'no-console': 0
  }
};
```