Масштабируемость

ВВЕДЕНИЕ

Первоначально платформа Node.js задумывалась как неблокирующий веб-сервер, более того, она и называлась web.js. Но ее создатель Райан Даль (Ryan Dahl) вскоре осознал широкие возможности платформы и начал дополнять ее инструментами для создания серверных приложений произвольного типа.

Характерные особенности платформы Node.js делают ее идеальной для реализации распределенных систем.

ВВЕДЕНИЕ

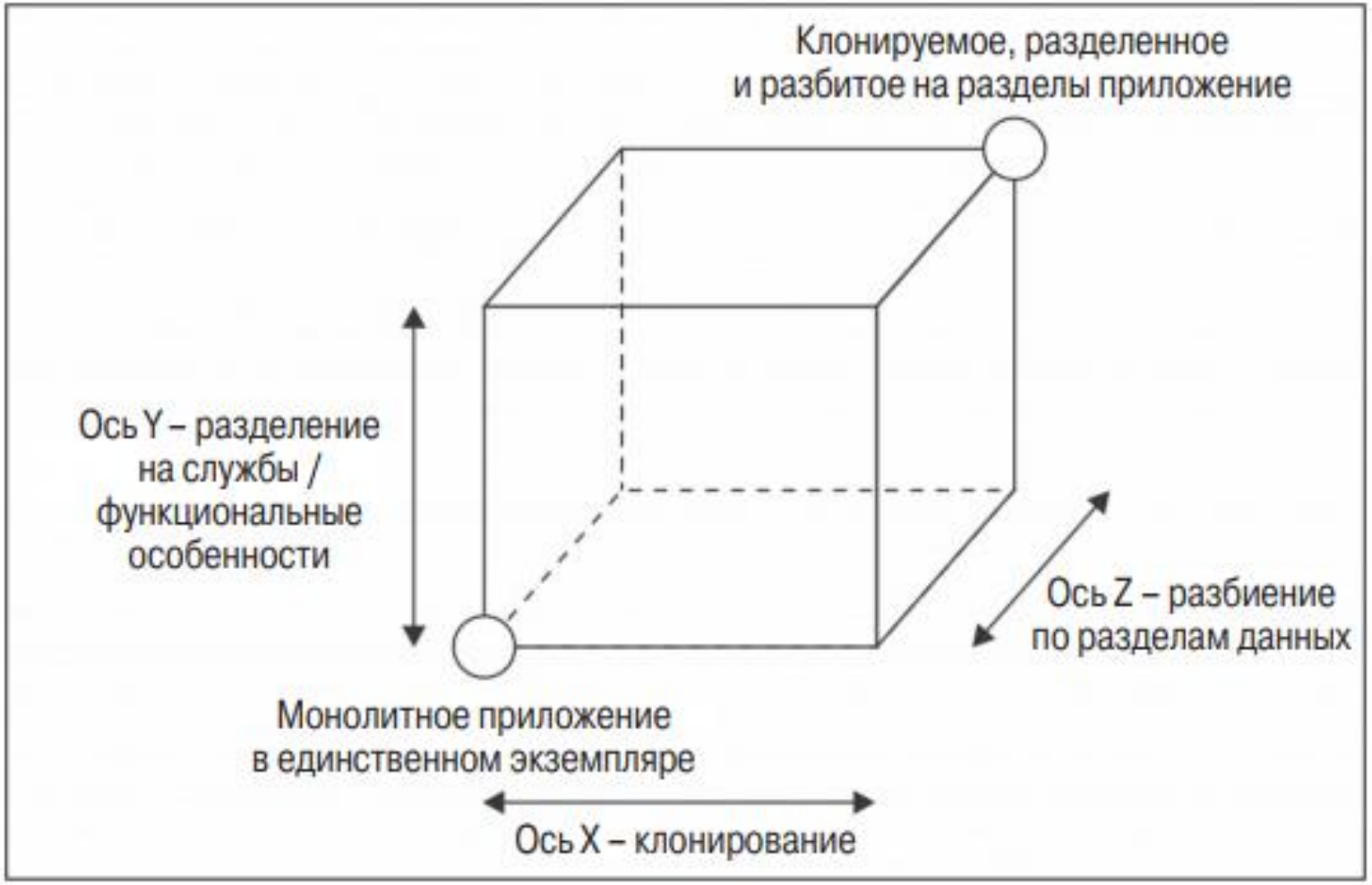
Каким бы мощным ни было аппаратное обеспечение, пропускная способность единственного потока выполнения все равно ограничена (обычно несколько сотен запросов в секунду), поэтому если вы желаете использовать Node.js для создания высоконагруженных приложений, вы вынуждены будете прибегнуть к масштабированию с использованием нескольких процессов и машин.

Используя те же приемы масштабирования, можно улучшить и другие характеристики приложения, такие как надежность и устойчивость к сбоям.

Кроме того, понятие «масштабируемость» относится также к размеру и сложности приложения, поскольку создание архитектуры, способной расширяться, играет важную роль при разработке программного обеспечения.

ТРИ ИЗМЕРЕНИЯ МАСШТАБИРУЕМОСТИ

Куб масштабирования - эта модель описывает масштабируемость с точки зрения трех размерностей:



ТРИ ИЗМЕРЕНИЯ МАСШТАБИРУЕМОСТИ

Ось x: Клонирование.

Является простым, и как правило, не дорогим (с точки зрения стоимости разработки) и весьма эффективным способом.

Суть этого способа заключается в клонировании одного и того же приложения n раз, что позволяет каждому экземпляру обрабатывать 1/n-часть рабочей нагрузки.

ТРИ ИЗМЕРЕНИЯ МАСШТАБИРУЕМОСТИ

Ось y: Разделение на службы/функциональные особенности.

Обозначает декомпозицию приложений по функциональным особенностям, службам или вариантам использования.

В данном случае декомпозиция предполагает создание разных автономных приложений, каждое из которых обладает собственной базой кода, а иногда и выделенной базой данных и даже отдельным пользовательским интерфейсом (например, выделение службы аутентификации пользователей, создание выделенного сервера аутентификации).

ТРИ ИЗМЕРЕНИЯ МАСШТАБИРУЕМОСТИ

Ось z: Разбиение на разделы данных.

Обозначает такое разбиение приложения, что каждый из его экземпляров отвечает только за часть всего массива данных).

Этот метод используется в основном для баз данных и по-другому называется горизонтальным разделением, или шардингом (sharding).

Например, можно разделить пользователей приложения по странам (списочное разделение) или на основе начальных букв их фамилий (диапазонное раз деление) или же позволить хеш-функциям определить принадлежность пользователя к определенному разделу (хешевое разделение).

Учитывая сложность, масштабирование приложения вдоль оси z следует применять только после того, как исчерпаны все воз-можности масштабирования по осям x и y куба масштабирования.

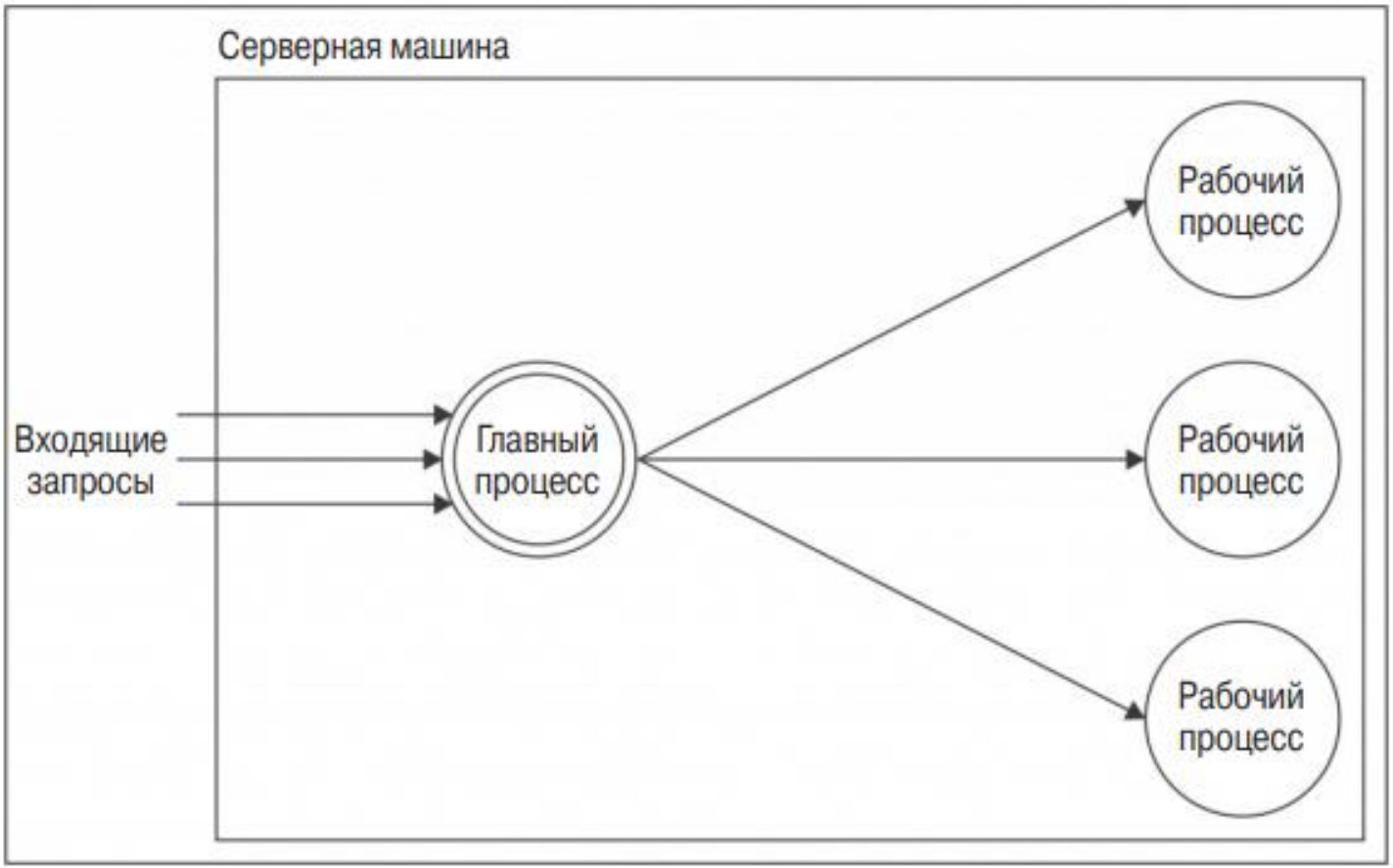
КЛОНИРОВАНИЕ

Единственному процессу Node.js доступен один поток и ему по умолчанию доступен ограниченный объем памяти 1,7 ГБ на 64-битных машинах.

Это значит, что приложения на платформе Node.js обычно начинают нуждаться в масштабировании гораздо раньше, чем традиционные веб-серверы, даже в контексте одной машины, чтобы иметь возможность пользоваться всеми ее ресурсами.

Не стоит считать это недостатком. Напротив, практически вынужденное масштабирование оказывает благоприятное воздействие на другие характеристики приложения, в частности на надежность, отказоустойчивость и заставляет разработчиков рассматривать возможность масштабирования на самых ранних этапах разработки приложения.

ШАБЛОН МАСШТАБИРОВАНИЯ ВЕТВЛЕНИЕМ ПРОЦЕССА



Главный процесс отвечает за запуск группы рабочих процессов, каждый из которых представляет экземпляр масштабируемого приложения. Затем все входящие соединения распределяются между рабочими процессами, что обеспечивает разделение нагрузки между ними.

МОДУЛЬ CLUSTER

Реализация этого простого шаблона основана на модуле cluster, входящем в состав библиотеки ядра.

Начиная с версии 0.11.2 NodeJS главный процесс реализует циклический алгоритм, гарантирующий равномерное распределение нагрузки между всеми рабочими процессами. Этот алгоритм включен по умолчанию на всех платформах, кроме Windows, что, впрочем, можно исправить, присвоив переменной cluster.schedulingPolicy константу cluster.SCHED_RR (циклический алгоритм) или cluster.SCHED_NONE (распределение нагрузки осуществляется операционной системой) до первого создания рабочего процесса.

СОЗДАНИЕ ПРОСТОГО HTTP-СЕРВЕРА

app.js:

```
const http = require('http');
const pid = process.pid;
var i = 0;
http.createServer((req, res) => {
        for (let i = 1e7; i>0; i--) {}
        console.log(`Handling request from ${pid} numder
${i++}`);
        res.end(`Hello from ${pid}\n`);
}).listen(8080, () => {
        console.log(`Started ${pid}`);
});
```

СОЗДАНИЕ ПРОСТОГО HTTP-СЕРВЕРА

Протестируем созданное приложение нагрузочным тестом. Измерим, сколько запросов в секунду сможет обработать сервер с использованием единственного процесса.

Инструмент для тестирование:
- Для Windows: siege-windows (https://github.com/ewwink/siege-windows);
- Для Linux: siege (https://www.joedog.org/siege-home/).

Пример параметров запуска siege:
> siege -c50 -t60S http://localhost:8080
//будет создано 50 параллельных соединений с сервером в течении 60 секунд

СОЗДАНИЕ ПРОСТОГО HTTP-СЕРВЕРА

server.js:

```
const cluster = require('cluster');
const os = require('os');
if(cluster.isMaster) {
        cluster.schedulingPolicy = cluster.SCHED_RR;
        const cpus = os.cpus().length;
        console.log(`Clustering to ${cpus} CPUs`);
        for (let i = 0; i<cpus; i++) {
                cluster.fork();
        }
} else {
        require('./app');
}
```

СОЗДАНИЕ ПРОСТОГО HTTP-СЕРВЕРА

Экземпляры рабочих процессов доступны через переменную cluster.workers, поэтому, например, для отправки всем им сообщения достаточно следующих строк кода:

```
Object.keys(cluster.workers).forEach(id => {
        cluster.workers[id].send('Hello from the master');
});
```

ОБЕСПЕЧЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ

Изменим модуль app.js:

```
// В конце модуля app.js
setTimeout(() => {
        throw new Error('Ooops');
}, Math.ceil(Math.random() * 3) * 1000);
```

А в server.js в ветку кода создания рабочих процессов:

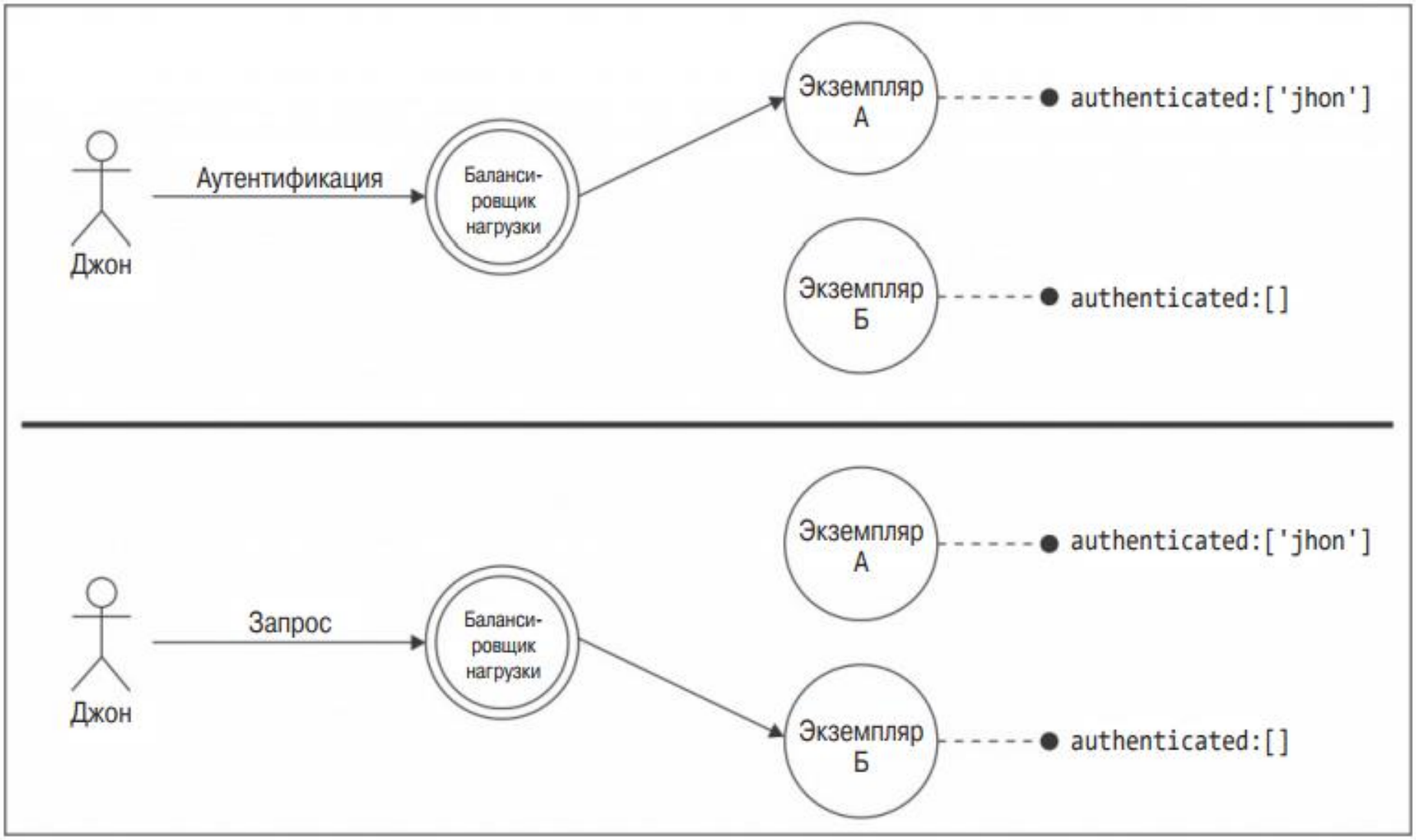
```
cluster.on('exit', (worker, code) => {
        if(code != 0 && !worker.exitedAfterDisconnect) {
                console.log('Worker crashed. Starting a new
worker');
                cluster.fork();
        }
});
```

ОБЕСПЕЧЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ

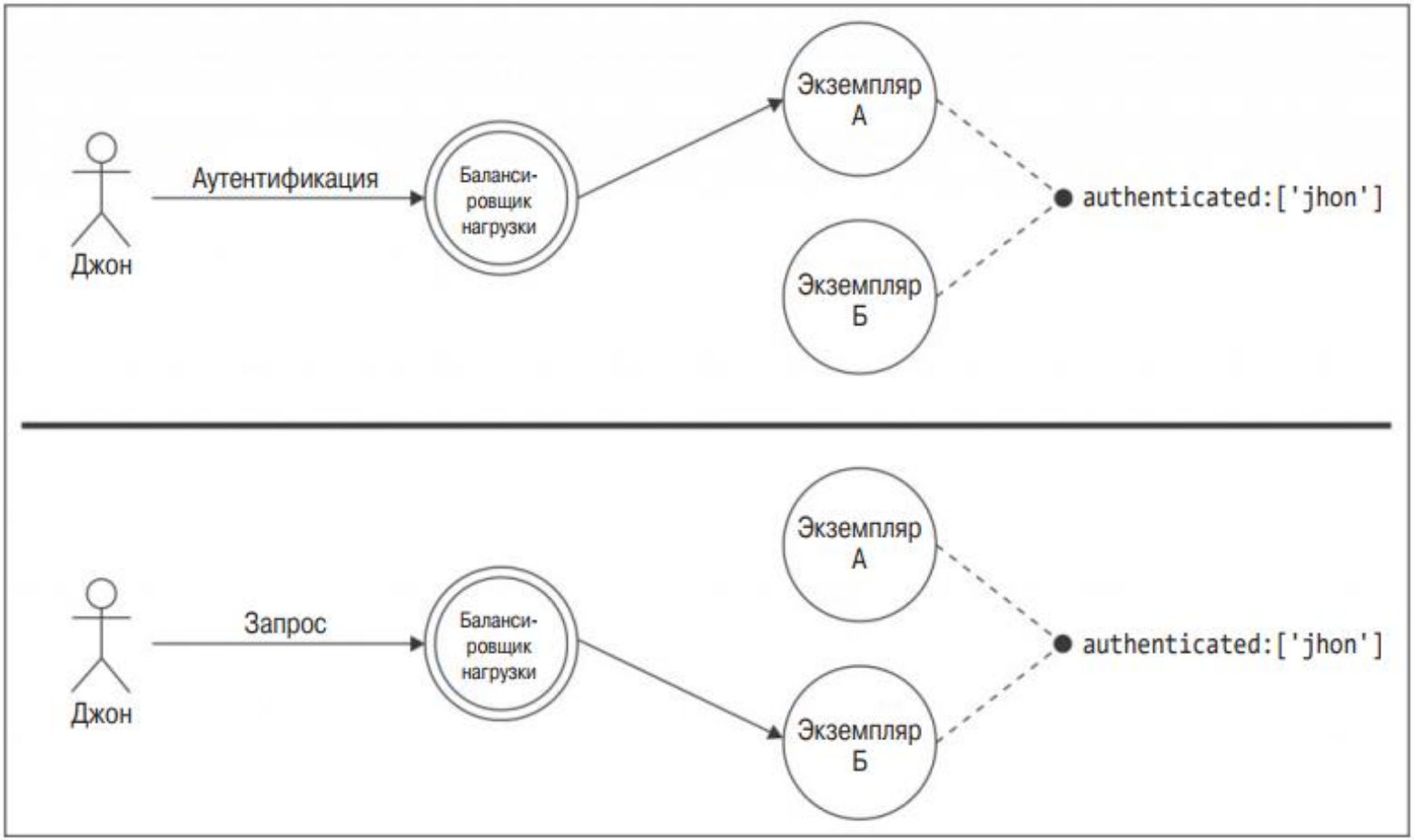
pm2 (https://github.com/Unitech/pm2) – небольшая утилита на основе модуля cluster, которая обеспечивает распределение нагрузки, мониторинг процессов, перезапуск без простоя и другие полезные возможности.

forever (https://npmjs.org/package/forever) – простой инструмент для обеспечения того, чтобы данный скрипт работал постоянно (т. е. вечно)

ВЗАИМОДЕЙСТВИЯ С СОХРАНЕНИЕМ СОСТОЯНИЯ

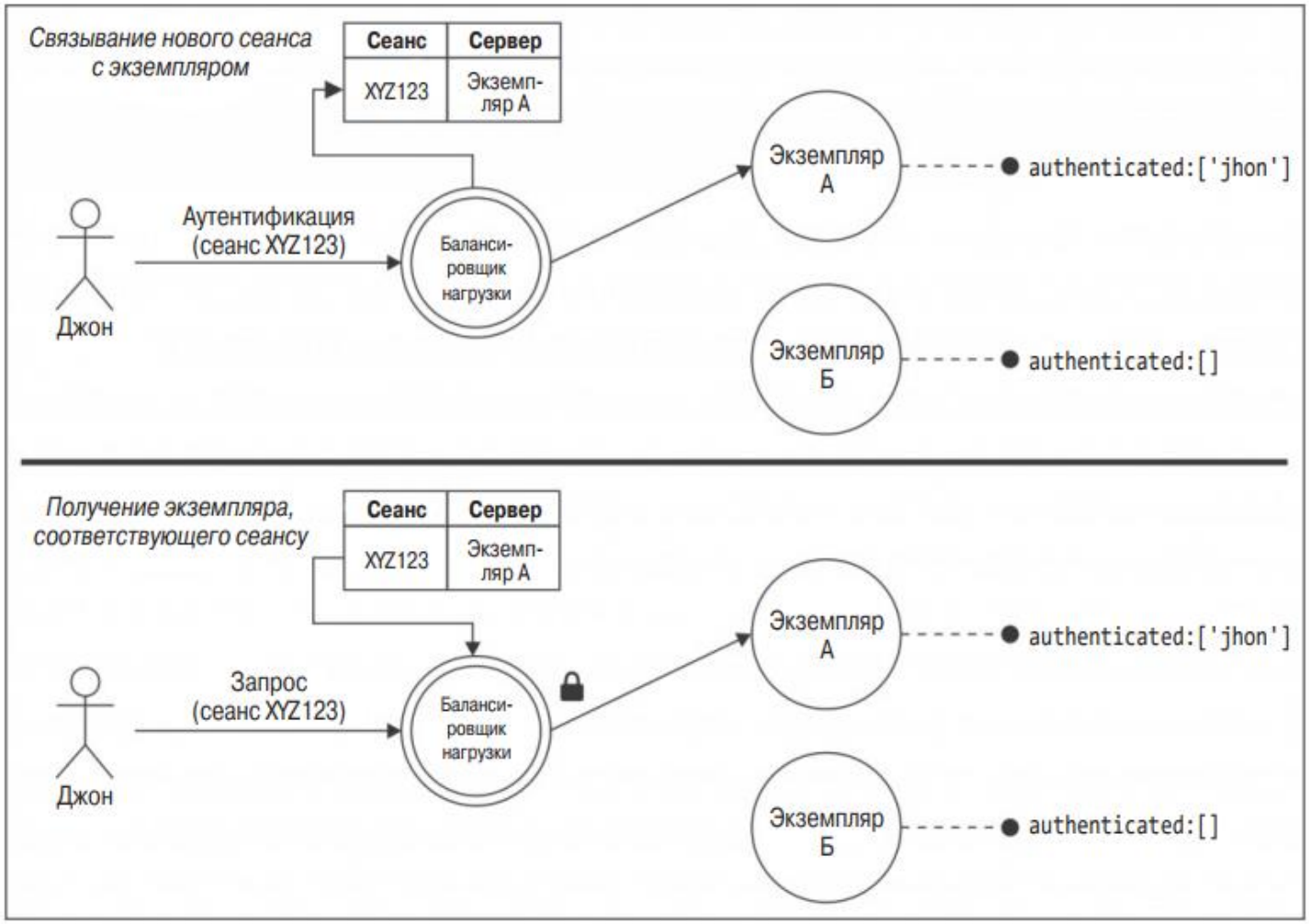


СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ СОСТОЯНИЯ НЕСКОЛЬКИМИ ЭКЗЕМПЛЯРАМИ



Для этого потребуется общее хранилище данных, такое как база данных, например PostgreSQL (http://www.postgresql.org), MongoDB (http://www.mongodb.org) или CouchDB (http://couchdb.apache.org), или, что еще лучше, хранилище в памяти, такое как Redis (http://redis.io) или Memcached (http://memcached.org).

РАСПРЕДЕЛЕНИЕ НАГРУЗКИ С ПРИВЯЗКОЙ К ЭКЗЕМПЛЯР



Распределение нагрузки с привязкой не поддерживается по умолчанию модулем cluster, но эту возможность можно добавить с помощью npm-библиотеки stickysession (https://www.npmjs.org/package/sticky-session).