



Mikroservisi i Node.js

Servisno-orijentisane arhitekture

JavaScript aplikacije

- Usložnjavanje aplikacija napisanih u JavaScript-u često dovodi do nekontrolisanog povećanja koda
- Organizacija po modulima i paketima često nije dovoljna da se pojednostavi aplikacija
- Da bi se aplikacija nesmetano razvijala, neophodno je velike, homogene strukture konvertovati u manje, nezavisne delove programa.
- Kompleksnost aplikacije može biti kontrolisana ako je ista razvijena na principu mikroservisa i još bolje korišćenjem Node.js ekosistema

Mikroservisi - podsetnik

- Stil servisno-orijentisane arhitekture gde se aplikacija struktura kao skup povezanih servisa
- Sama arhitektura oslanja se na lightweight protokole
- Mikroservisi služe da se aplikacija dezintegriše u manje delove čime se obezbeđuje modularnost iste
- Kreiranje JavaScript aplikacija korišćenjem mikroservisne arhitekture omogućava programerima razvoj monofunkcionalnih modula koji imaju jasno definisane operacije i interfejse
- Sam razvojni proces postaje agilniji, što doprinosi boljoj testabilnosti celog sistema
- Jedini zahtev koji mikroservisna arhitektura nameće jesu REST-ful API-ji za komunikaciju i druge servise

Node.js platforma

- Kada se radi o razvoju JavaScript mikroservisa, većina programera preferira Node.js platformu
- Node.js je cross-platform, open-source runtime environment (RTE) koji se koristi za razvoj mrežnih i server-side aplikacija
- Go truly full stack
- Napisan je u JavaScript-u
- Neblokirajuća arhitektura
- Zajedno sa Guglovim V8 JavaScript engine-om, ima neverovatnu brzinu
- Jednostavno deljenje podataka između servisa
- Ogroman community

npm

- **npm** (eng. *Node Package Manager*) je menadžer paketa za JavaScript programski jezik.
- Takođe je podrazumevani menadžer paketa za Node.js radno okruženje
- U potpunosti napisan u JavaScript-u
- Sastoji se od npm konzole i onlajn baze podataka javnih i plaćenih paketa (registara)
- Privatnim paketima (registrima) pristupa se preko klijentske konzole, dok se dostupni javni paketi mogu pretraživati na npm veb sajtu.

Osnovne potrebe mikroservisa

- Komunikacija
 - Obrada zahteva
 - Generisanje zahteva
 - Procesiranje događaja
 - Generisanje događaja
- Skladištenje
 - Baza podataka
 - File storage

Razvoj aplikacija

- Instalacija Node.js platforme
 - Preuzmite poslednju node.js verziju sa zvaničnog sajta nodejs.org
 - npm može biti koristan (Node.js instalacija uključuje i npm)

Kreiranje *heroes* servisa - 1

- Otvorite direktorijum gde želite da kreirate projekat
- Kreirajte folder heroes i fajl heroes.js unutar njega
 - ./heroes/heroes.js
- Ukoliko koristite neki source code control alat, sada je dobro vreme da inicijalizujete repozitorijum
 - Ukoliko koristite Git, ne zaboravite i .gitignore datoteku

Kreiranje *heroes* servisa - 2

- Inicijalizacija npm projekta unutar ./heroes direktorijuma
 - npm init -y
- Instalacija zavisnosti
 - npm install express body-parser

Implementacija *heroes* servisa

```
1  const express = require('express');
2  const path = require('path');
3  const bodyParser = require('body-parser');
4
5  const port = process.argv.slice(2)[0];
6  const app = express();
7  app.use(bodyParser.json());
8
9  const powers = [
10     { id: 1, name: 'flying' },
11     { id: 2, name: 'teleporting' },
12     { id: 3, name: 'super strength' },
13     { id: 4, name: 'clairvoyance' },
14     { id: 5, name: 'mind reading' }
15 ];
16
```

Implementacija *heroes* servisa

```
16
17  const heroes = [
18    {
19      id: 1,
20      type: 'spider-dog',
21      displayName: 'Cooper',
22      powers: [1, 4],
23      img: 'cooper.jpg',
24      busy: false
25    },
26    {
27      id: 2,
28      type: 'flying-dogs',
29      displayName: 'Jack & Buddy',
30      powers: [2, 5],
31      img: 'jack_buddy.jpg',
32      busy: false
33    },
```

```
34    {
35      id: 3,
36      type: 'dark-light-side',
37      displayName: 'Max & Charlie',
38      powers: [3, 2],
39      img: 'max_charlie.jpg',
40      busy: false
41    },
42    {
43      id: 4,
44      type: 'captain-dog',
45      displayName: 'Rocky',
46      powers: [1, 5],
47      img: 'rocky.jpg',
48      busy: false
49    }
50  ];
51
```

Implementacija *heroes* servisa

```
51
52 app.get('/heroes', (req, res) => {
53     console.log('Returning heroes list');
54     res.send(heroes);
55 });
56
57 app.get('/powers', (req, res) => {
58     console.log('Returning powers list');
59     res.send(powers);
60 });
61
```

Implementacija *heroes* servisa

```
62 app.post('/hero/**', (req, res) => {
63   const heroId = parseInt(req.params[0]);
64   const foundHero = heroes.find(subject => subject.id === heroId);
65
66   if (foundHero) {
67     for (let attribute in foundHero) {
68       if (req.body[attribute]) {
69         foundHero[attribute] = req.body[attribute];
70         console.log(`Set ${attribute} to ${req.body[attribute]} in hero: ${heroId}`);
71       }
72     }
73     res.status(202).header({Location: `http://localhost:${port}/hero/${foundHero.id}`}).send(foundHero);
74   } else {
75     console.log(`Hero not found.`);
76     res.status(404).send();
77   }
78 });
79
```

Implementacija *heroes* servisa

```
79  
80 app.use('/img', express.static(path.join(__dirname, 'img')));  
81  
82 console.log(`Heroes service listening on port ${port}`);  
83 app.listen(port);
```

Img folder

- U okviru heroes direktorijuma napraviti poddirektorijum img koji uključuje sledeće slike:
 - cooper.jpg
 - jack_buddy.jpg
 - max_charlie.jpg
 - rocky.jpg

Testiranje *heroes* servisa

- Servis pokrecemo komandom **node ./heroes/heroes.js 8081**
- Rad servisa možemo proveriti bilo kojim alatom
 - Postman, curl, PowerShell, browser, ...

Testiranje *heroes* servisa - curl

- `curl -i --request GET localhost:8081/heroes`
- Ako servis radi korektno, dobija se rezultat sličan ovome:

HTTP/1.1 200 OK

X-Powered-By: Express

Content-Type: application/json; charset=utf-8

Content-Length: 424

ETag: W/"1a8-BIZzoIRo/ZugcWv+LFVGSU1qIZU"

Date: Thu, 01 Apr 2021 12:07:07 GMT

Connection: keep-alive

```
[{"id":1,"type":"spider-dog","displayName":"Cooper","powers":[1,4],"img":"cooper.jpg","busy":false}, {"id":2,"type":"flying-dogs","displayName":"Jack & Buddy","powers":[2,5],"img":"jack_buddy.jpg","busy":false}, {"id":3,"type":"dark-light-side","displayName":"Max & Charlie","powers":[3,2],"img":"max_charlie.jpg","busy":false}, {"id":4,"type":"captain-dog","displayName":"Rocky","powers":[1,5],"img":"rocky.jpg","busy":false}]
```

Testiranje *heroes* servisa - Postman

The screenshot displays the Postman application interface. The top navigation bar includes 'Home', 'Workspaces', 'Reports', and 'Explore'. A search bar is present, along with 'Sign In' and 'Create Account' buttons. The left sidebar shows 'Scratch Pad' with 'New' and 'Import' buttons, and a list of collections (APIs, Environments, Mock Servers, Monitors, History). The main workspace shows a GET request to 'localhost:8081/heroes'. The response is a JSON array of two objects, displayed in 'Pretty' format. The status bar at the bottom shows '200 OK' and '19 ms'.

Postman

File Edit View Help

Home Workspaces Reports Explore

Search Postman

Sign In Create Account

Scratch Pad New Import

Collections

APIs

Environments

Mock Servers

Monitors

History

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Create a new Collection

Overview GET localhost:8081/her...

localhost:8081/heroes

Save

GET localhost:8081/heroes

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (7) Test Results

200 OK 19 ms 661 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "type": "spider-dog",
4   "displayName": "Cooper",
5   "powers": [
6     1,
7     4
8   ],
9   "img": "cooper.jpg",
10  "busy": false
11 },
12 {
13   "id": 2,
14   "type": "flying-dog"
15 }
```

Find and Replace Console

GET http://localhost:8081/heroes

200 19 ms

Runner

Kreiranje *threats* servisa - 1

- Otvorite direktorijum projekta
- Kreirajte direktorijum threats i fajl threats.js unutar njega
 - `./ threats / threats.js`
- Kreirati direktorijum img unutar threats
 - `./threats/img/`
 - `tower.jpg`
 - `mess.jpg`
 - `joke.png`
- `npm init -y`
- `npm install express body-parser request`

Implementacija *threats* servisa

```
1  const express = require('express');
2  const path = require('path');
3  const bodyParser = require('body-parser');
4  const request = require('request');
5
6  const port = process.argv.slice(2)[0];
7  const app = express();
8
9  app.use(bodyParser.json());
10
11 const heroesService = 'http://localhost:8081';
12
```

Implementacija *threats* servisa

```
13 const threats = [  
14   {  
15     id: 1,  
16     displayName: 'Pisa tower is about to collapse.',  
17     necessaryPowers: ['flying'],  
18     img: 'tower.jpg',  
19     assignedHero: 0  
20   },  
21   {  
22     id: 2,  
23     displayName: 'Engineer is going to clean up server-room.',  
24     necessaryPowers: ['teleporting'],  
25     img: 'mess.jpg',  
26     assignedHero: 0  
27   },
```

```
28   {  
29     id: 3,  
30     displayName: 'John will not understand the joke',  
31     necessaryPowers: ['clairvoyance'],  
32     img: 'joke.jpg',  
33     assignedHero: 0  
34   }  
35 ];
```

Implementacija *threats* servisa

```
36  
37 app.get('/threats', (req, res) => {  
38   console.log('Returning threats list');  
39   res.send(threats);  
40 });  
41
```

Implementacija *threats* servisa

```
42 app.post('/assignment', (req, res) => {
43   request.post({
44     headers: {'content-type': 'application/json'},
45     url: `${heroesService}/hero/${req.body.heroId}`,
46     body: `{
47       "busy": true
48     }`
49   }, (err, heroResponse, body) => {
50     if (!err) {
51       const threatId = parseInt(req.body.threatId);
52       const threat = threats.find(subject => subject.id === threatId);
53       threat.assignedHero = req.body.heroId;
54       res.status(202).send(threat);
55     } else {
56       res.status(400).send({problem: `Hero Service responded with issue ${err}`});
57     }
58   });
59 });
60
```

Implementacija *threats* servisa

```
61 app.use('/img', express.static(path.join(__dirname, 'img')));  
62  
63 console.log(`Threats service listening on port ${port}`);  
64 app.listen(port);
```


Testiranje *threats* servisa

- Servis pokrecemo komandom **node threats/threats.js 8082**
- Rad servisa možemo proveriti bilo kojim alatom
 - Postman, curl, PowerShell, browser, ...

Testiranje *threats* servisa - curl

- `curl -i --request POST --header "Content-Type: application/json" --data '{"herold": 1, "threatId": 1}' localhost:8082/assignment`
- Ako servis radi korektno, dobija se rezultat sličan ovome:
HTTP/1.1 202 Accepted
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 121
ETag: W/"79-ER1WRPW1305+Eomgfjq/A/Cgkp8"
Date: Thu, 04 Apr 2019 19:32:56 GMT
Connection: keep-alive

`{"id":1,"displayName":"Pisa tower is about to collapse.","necessaryPowers":["flying"],"img":"tower.jpg","assignedHero":1}`



Moleculer

- microservices framework for Node.js -

Moleculer

- Framework za kreiranje mikroservisa u Node.js-u.
- Relativno je jednostavan za korišćenje, efikasan je i aktivno se razvija.
- Event-driven arhitektura.

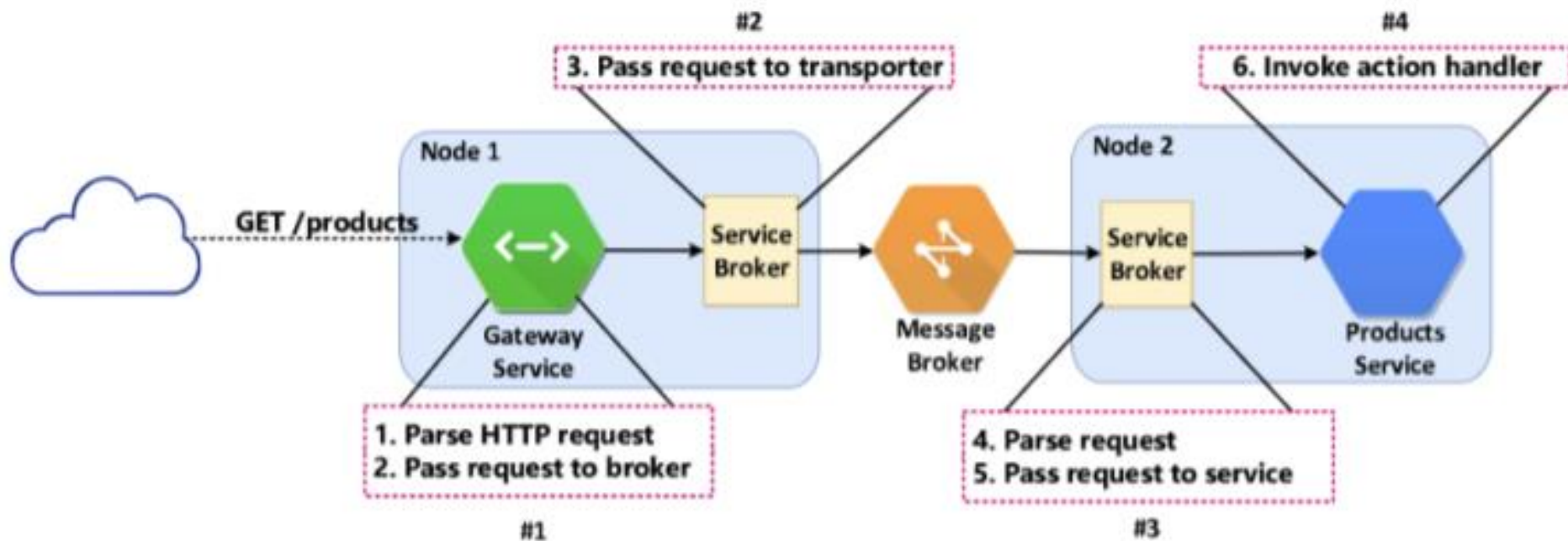
Molecular – ključni koncepti

- Servis
- Node
- Lokalni Servisi
- Udaljeni Servisi
- Servis Broker
- Transporter
- Gateway

Moleculer – primer 1: Online store

- Product servis odgovoran za čuvanje i upravljanje proizvodima
- Gateway servis za primanje klijentskih zahteva i rutiranje tih zahteva ka products servisu.
- Zasebni nodovi za servise
 - Da bismo izvršavali servise na različitim node-ovima neophodan nam je transporter koji će da obavlja komunikaciju između servisa.
 - Većina transportera koji se mogu koristiti u Moleculer-u se oslanjaju na message brokere za međuservisnu komunikaciju (u ovom primeru će biti korišćen NATS).

Online store – ciklus pribavljanja proizvoda



Online store – implementacija

- kreirati direktorijum (npr. Moleculer-store) – `mkdir moleculer-store`
- - `cd moleculer-store`
- - `npm init -y`
- - `npm install moleculer moleculer-web nats`
- - Sledeći korak je konfiguracija brokera – treba kreirati `index.js` fajl

```
package.json X
package.json > ...
1  {
2    "name": "moleculer-store",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "Teodora",
11   "license": "ISC",
12   "dependencies": {
13     "moleculer": "^0.14.6",
14     "moleculer-web": "^0.9.1",
15     "nats": "^1.4.8"
16   }
17 }
```


Online store – brokerNode1

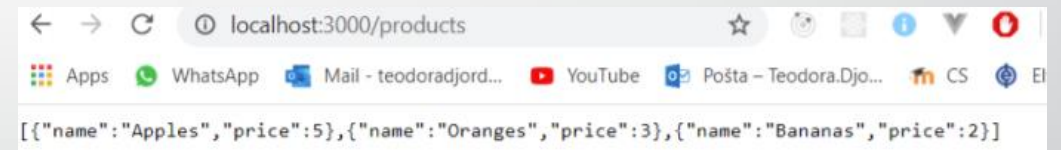
```
1 // index.js
2 const { ServiceBroker } = require("moleculer");
3 const HTTPServer = require("moleculer-web");
4
5 // Create the broker for node-1
6 // Define nodeID and set the communication bus
7 const brokerNode1 = new ServiceBroker({
8   nodeID: "node-1",
9   transporter: "nats://localhost:4222"
10 });
11
12 // Create the "gateway" service
13 brokerNode1.createService({
14   // Define service name
15   name: "gateway",
16   // Load the HTTP server
17   mixins: [HTTPServer],
18
19   settings: {
20     routes: [
21       {
22         aliases: {
23           // When the "GET /products" request is made the "listProducts" action of "products"
24           // service is executed
25           "GET /products": "products.listProducts"
26         }
27       }
28     ]
29   }
30 });
31
```

Online store – brokerNode2 i startovanje brokera

```
33 // Create the broker for node-2
34 // Define nodeID and set the communication bus
35 const brokerNode2 = new ServiceBroker({
36   nodeID: "node-2",
37   transporter: "nats://localhost:4222"
38 });
39
40 // Create the "products" service
41 brokerNode2.createService({
42   // Define service name
43   name: "products",
44
45   actions: {
46     // Define service action that returns the available products
47     listProducts(ctx) {
48       return [
49         { name: "Apples", price: 5 },
50         { name: "Oranges", price: 3 },
51         { name: "Bananas", price: 2 }
52       ];
53     }
54   }
55 });
56
57 // Start both brokers
58 Promise.all([brokerNode1.start(), brokerNode2.start()]);
59
```

Online store – Testiranje

- Pre pokretanja i testiranja online store aplikacije treba pokrenuti nats. To je moguće učiniti komandom nats-server
- Sledi pokretanje aplikacije komandom node index.js. Nakon ovog koraka aplikaciju je moguće testirati na <http://localhost:3000/products>.



Moleculer – primer 2: Movies

- Kreiranje većeg broja servisa, gde će se svaki od servisa izvršavati u zasebnom kontejneru
 - Prvi servis će biti gateway koji treba da prihvata zahteve na registrovanim rutama
 - Drugi servis će čuvati i upravljati filmovima
 - Treći servis će imati registrovane event-ove koji će "slati mejl" kada se neki događaj koji to zahteva okine.
- NATS kao message broker

Movies - implementacija

- pokretanje komande za inicijalizaciju projekta - npm init
- npm install --save moleculer nats express body-parser
- kreiranje direktorijuma *services* i kreiranje fajlova email.service.js, movies.service.js i gateway.service.js u tom direktorijumu
- kreiranje Dockerfile-a u glavnom direktorijumu
- dodavanje .dockerignore fajla za node_modules
- implementacija servisa
 - gateway.service.js
 - email.service.js
 - movies.service.js

```
Dockerfile > ...  
1 FROM node:latest  
2 RUN mkdir /app  
3 WORKDIR /app  
4 ADD package*.json /app/  
5 RUN npm install  
6 ADD . /app/  
7 CMD [ "npm", "start" ]
```

Movies – npm run dev

```
mol $ services
```

Service	Version	State	Actions	Events	Nodes
\$node	-	OK	7	0	1
email	-	OK	0	1	1
gateway	-	OK	0	0	1
movies	-	OK	3	0	1

```
mol $
```

Implementacija servisa - 1

```
services > gateway.service.js > ...
1  "use strict";
2  const express = require("express");
3  const bodyParser = require('body-parser');
4
5  module.exports = {
6    name: "gateway",
7    settings: {
8      port: process.env.PORT || 3000,
9    },
10   methods: {
11     initRoutes(app) {
12       app.get("/movies", this.getMovies);
13       app.get("/movies/:id", this.getMovie);
14       app.post("/movies", this.createMovie);
15     },
16     getMovies(req, res) {
17       return Promise.resolve()
18         .then(() => {
19           return this.broker.call("movies.listAll").then(movies => {
20             res.send(movies);
21           });
22         })
23         .catch(this.handleError(res));
24   },
```

Implementacija servisa - 2

```
25     getMovie(req, res) {
26         const id = req.params.id;
27         return Promise.resolve()
28             .then(() => {
29             return this.broker.call("movies.getById", {id: id}).then(movie => {
30                 res.send(movie);
31             });
32             })
33             .catch(this.handleError(res));
34     },
35     createMovie(req, res) {
36         const payload = req.body;
37         return Promise.resolve()
38             .then(() => {
39             return this.broker.call("movies.create", { payload }).then(movie =>
40                 res.send(movie)
41             );
42             })
43             .catch(this.handleError(res));
44     },
45     handleError(res) {
46         return err => {
47             res.status(err.code || 500).send(err.message);
48         };
49     }
50 },
```


Implementacija servisa - 3

```
51     created() {  
52         const app = express();  
53         app.use(bodyParser.urlencoded({ extended: false }));  
54         app.use(bodyParser.json());  
55         app.listen(this.settings.port);  
56         this.initRoutes(app);  
57         this.app = app;  
58     }
```

Implementacija servisa - 4

```
services > movies.service.js > ...
1  "use strict";
2
3  const movies = [
4    {id: 1, title: 'Sharknado'},
5    {id: 2, title: 'Roma'},
6  ];
7
8  module.exports = {
9    name: "movies",
10
11    actions: {
12      listAll(ctx) {
13        return Promise.resolve({ movies: movies });
14      },
15      getById(ctx) {
16        const id = Number(ctx.params.id);
17        return Promise.resolve(movies.find(movie => movie.id === id ));
18      },
19      create(ctx) {
20        const lastId = Math.max(...movies.map(movie => movie.id));
21        const movie = {
22          id: lastId + 1,
23          ...ctx.params.payload,
24        };
25        movies.push(movie);
26        this.broker.emit("movie.created", movie);
27        return Promise.resolve(movie);
28      }
29    },
30  };
```

Implementacija servisa - 5

```
services > 🚩 email.service.js > ...  
1  "use strict";  
2  
3  module.exports = {  
4    name: "email",  
5    events: {  
6      "movie.created": {  
7        group: "other",  
8        handler(payload) {  
9          console.log('Recieved "movie.created" event in email service with payload: ', payload);  
10       }  
11    }  
12  },  
13  };|
```

Implementacija servisa - config fajl

```
molecular.config.js > ...  
1  "use strict";  
2  const os = require("os");  
3  
4  module.exports = {  
5    nodeId: (process.env.NODEID ? process.env.NODEID + "-" : "") + os.hostname().toLowerCase(),  
6    // metrics: true  
7    // cacher: true  
8  };
```

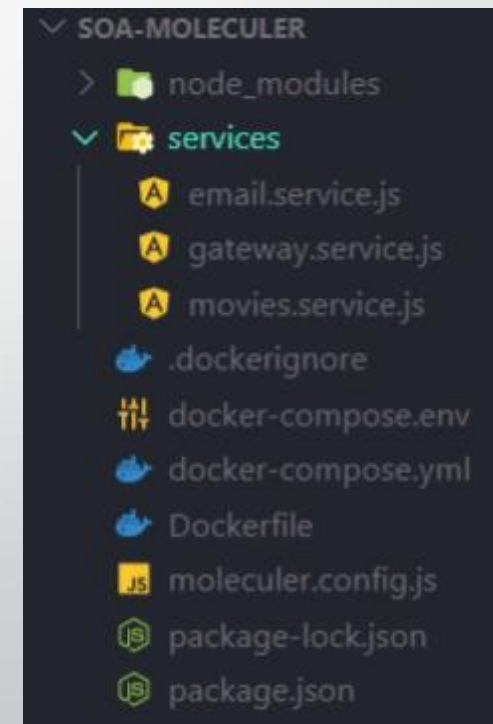
Pokretanje servisa u zasebnim kontejnerima

```
docker-compose.env
1 LOGLEVEL=info
2 TRANSPORTER=nats://nats:4222
3 SERVICEDIR=services
```

```
docker-compose.yml
1 version: '3.0'
2 services:
3   nats:
4     image: nats:latest
5   gateway:
6     build:
7       context: .
8     image: service-gateway
9     env_file: docker-compose.env
10    environment:
11      NODEID: "node-gateway"
12      SERVICES: gateway
13      PORT: 3000
14    ports:
15      - "3000:3000"
16    depends_on:
17      - nats
18    email:
19      build:
20        context: .
21      env_file: docker-compose.env
22      environment:
23        NODEID: "node-email"
24        SERVICES: email
25      depends_on:
26        - nats
27    movies:
28      build:
29        context: .
30      env_file: docker-compose.env
31      environment:
32        NODEID: "node-movies"
33        SERVICES: movies
34      depends_on:
35        - nats
```

docker-compose

- docker-compose.yml definiše servise koji čine našu aplikaciju i koji će se izvršavati zajedno.
- Potreban nam je nats koji se kreira na osnovu slike nats:latest.
- Svaki od ostalih servisa zavisi od nats-a tako da treba staviti depends_on opciju.
- Za ostale servise se navodi naziv slike, na osnovu kog dockerfile-a se pravi ta slika, environment fajl koji smo prethodno definisali i još neke promenljive koje su neophodne.
- Vrlo je važno navesti SERVICES promenljivu kojoj ćemo dodeliti ime odgovarajućeg mikroservisa da bi se znalo koji mikroservis treba pokrenuti.
- Za gateway se dodatno specificira i broj porta.




Testiranje – GET /movies

The screenshot displays a REST client interface with the following components:

- Request Bar:** Shows the method `GET` and the URL `http://localhost:3000/movies`. Buttons for `Send` and `Save` are present.
- Params Tab:** The 'Query Params' section is active, showing a table with columns for KEY, VALUE, and DESCRIPTION.
- Body Tab:** The 'Body' tab is selected, showing the response status `200 OK`, time `110 ms`, and size `277 B`. A `Save Response` button is available.
- Response Body:** The response is displayed in JSON format, showing an array of two movie objects.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```
1 {
2   "movies": [
3     {
4       "id": 1,
5       "title": "Sharknado"
6     },
7     {
8       "id": 2,
9       "title": "Roma"
10    }
11  ]
12 }
```



???