Node.js: comandi da shell in tempo reale con Fastify e WebSocket

In questo tutorial eseguiremo un comando da shell in tempo reale utilizzando Fastify e WebSocket.

Fastify ha un plugin davvero efficace che gestisce le connessione WebSockets. Questo plugin è `@fastify/websocket`.

Useremo un semplice comando `ping` per vedere su una pagina web l'output proveniente direttamente dalla shell mentre il terminale sta elaborando il comando in esecuzione.

Quando il terminale produce una nuova riga di output, questa riga verrà inviata al client attraverso un WebSocket offrendo così all'utente un'esperienza in tempo reale.

Gli utenti possono fornire un nome host o un indirizzo IP, quindi è necessario convalidare questo tipo di input per evitare potenziali attacchi di shell injection.

Innanzitutto, dobbiamo installare i pacchetti NPM richiesti:

```
npm install fastify @fastify/websocket
@fastify/static validator
```

La nostra app avrà semplicemente un modulo su una pagina HTML statica e un endpoint API in cui possiamo utilizzare i WebSockets tramite il plugin di Fastify `websocket`.

La logica alla base del comando da shell può essere effettivamente racchiusa all'interno di una classe specifica.

```
'use strict';

const { spawn } = require('child_process');
const validator = require('validator');

class Ping {
    constructor(host, times = 3) {
        this.host = host;
        this.times = times;
    }

    validate() {
        if(!validator.isFQDN(this.host) &&
!validator.isIP(this.host)) {
            return false;
        }
        if(this.times < 1 || this.times > 10) {
            return false;
        }
        const times = this.times.toString();

        return validator.isInt(times);
    }

    send({ ondata = function () {}, onerror =
function () {}, onclose = function () {} }) {
        if(!this.validate()) {
            throw new Error('Invalid parameters.');
        }
        const cmd = spawn('ping', ['-c', this.times,
this.host]);

        cmd.stdout.on('data', data => {
            ondata(data.toString());
```

```
        });

        cmd.stderr.on('data', data => {
            onerror(data.toString());
        });

        cmd.on('close', code => {
            onclose(code);
        });
    }
}

module.exports = Ping;
```

Il metodo `validate()` utilizza il modulo NPM `validator` per assicurarsi che il parametro `host` fornito sia sempre un FQDN o un indirizzo IPv4/IPv6. Se questa convalida fallisce, il metodo `send()` genererà immediatamente un'eccezione. Questo metodo contiene l'effettiva esecuzione del nostro comando shell. Ci sono tre situazioni possibili:

1. L'esecuzione procede senza alcun errore, quindi l'output viene inviato a `stdout`.
2. L'esecuzione restituisce un errore, quindi l'output viene inviato a `stderr` (ad es. la risoluzione dell'host non riesce).
3. Il comando restituisce infine un codice di stato. Questo codice è gestito dall'evento `close`.

Abbiamo definito tre specifiche funzioni di callback che possiamo utilizzare quando associamo la nostra classe alla route API di Fastify che gestisce una connessione web socket:

```
'use strict';

const Ping = require('../lib/Ping');
```

```
module.exports = function (fastify, opts, done) {

    fastify.get('/ping', { websocket: true },
(connection, request) => {

        connection.socket.on('message', message => {

            const data =
JSON.parse(message.toString());

            try {
                const ping = new Ping(data.host,
data.times);
                const actions = {
                    ondata(data) {
                        connection.socket.send(data);
                    },
                    onerror(msg) {

connection.socket.send(`error: ${msg}`);
                    },
                    onclose(code) {

connection.socket.send(`process exited with code
${code}`);
                    }
                };
                ping.send(actions);
            } catch (err) {
                connection.socket.send('Request
error.');
            }
```

```
        });

    });
    done();
};
```

Un client JavaScript invia la stringa JSON contenente il nome host e il numero di tentativi. Il codice lato server ottiene questa stringa una volta stabilita una connessione WebSocket. Quindi viene eseguito il comando della shell e ogni volta che viene attivato uno dei suddetti eventi della shell, inviamo l'output restituito al client utilizzando il metodo `send()` del socket.

Il codice lato client è piuttosto semplice:

```
 const form = document.getElementById('ping-form');
            if(form !== null) {
                const response =
document.getElementById('output');
                const hostInput =
document.getElementById('host');
                const submit =
form.querySelector('input[type="submit"]');

                const wsURL = 'ws://' + location.host
+ '/api/ping';
                const wS = new WebSocket(wsURL);

                const handleWSMsg = (data, target) =>
{
                    if(data.includes('Error') ||
data.includes('error:')) {
                        target.innerHTML = data;
                        return false;
```

```
                    }
                    if(!data.includes('process')) {
                        let line =
document.createElement('div');
                        line.innerText = data;
                        target.appendChild(line);
                    } else {
                        let close =
document.createElement('div');
                        close.innerText = data;
                        target.appendChild(close);

submit.removeAttribute('disabled', 'disabled');
                    }
                };

                wS.onmessage = evt => {
                    handleWSMsg(evt.data, response);
                };


                form.addEventListener('submit', e =>
{
                    e.preventDefault();
                    response.innerHTML = '';
                    submit.setAttribute('disabled',
'disabled');
                    const host = hostInput.value;
                    const times = 3;

                    const data = {
                        host,
                        times
                    };
```

```
                wS.send(JSON.stringify(data));
            }, false);
        }
```

Il nostro client stabilisce la connessione iniziale aprendo un nuovo WebSocket sull'endpoint di Fastify. Quando il modulo viene inviato, a Fastify viene inviata una stringa JSON contenente il nome host e il numero di tentativi. Il gestore dell'evento `message` riceve ogni messaggio inviato dal server e aggiorna il DOM di conseguenza. Ciò avviene leggendo la proprietà `data` dell'evento `message`.

Conclusione

Un plugin è generalmente una soluzione migliore quando si tratta di mantenere il nostro codice il più semplice e pulito possibile. Invece di utilizzare un server WebSocket a livello globale se abbiamo solo bisogno è un endpoint dedicato, Fastify ci fornisce uno strumento più flessibile che può essere ulteriormente personalizzato per soddisfare le nostre esigenze.

Codice sorgente

Node.js: ping command with Fastify and Web Socket