Micro Servizi Base

Slide a cura di Mirko Campari

Cos'è un Micro servizio

Un microservizio ma più in generale i microservizi sono un approccio di sviluppo e organizzazione dell'architettura dei software, secondo cui quest'ultimi, sono composti di servizi indipendenti di piccole dimensioni che comunicano tra loro tramite API ben definite.

I vantaggi a livello di strutturazione del carico e dei team di lavoro sono molteplici e molto sfaccettate sopratutto in agile.

Le architetture dei microservizi permettono di scalare e sviluppare costole dellle applicazioni in modo più rapido e semplice, permettendo di promuovere l'innovazione e accelerare il time-to-market di nuove funzionalità e la capacità di versionamento.

Cos'è un Micro servizio

Un microservizio ma più in generale i microservizi sono un approccio di sviluppo e organizzazione dell'architettura dei software, secondo cui quest'ultimi, sono composti di servizi indipendenti di piccole dimensioni che comunicano tra loro tramite API ben definite.

I vantaggi a livello di strutturazione del carico e dei team di lavoro sono molteplici e molto sfaccettate sopratutto in agile.

Le architetture dei microservizi permettono di scalare e sviluppare costole dellle applicazioni in modo più rapido e semplice, permettendo di promuovere l'innovazione e accelerare il time-to-market di nuove funzionalità e la capacità di versionamento.

Architettura monolitica vs. architettura di microservizi

Nelle architetture monolitiche tutti i processi sono strettamente collegati tra loro e vengono eseguiti come un singolo servizio.
Ciò significa che se un processo dell'applicazione sperimenta un picco nella richiesta, è necessario ridimensionare l'intera architettura.
Aggiungere o migliorare una funzionalità dell'applicazione monolitica diventa più complesso, in quanto sarà necessario aumentare la base di codice. Tale complessità limita la sperimentazione e rende più difficile implementare nuove idee.
Le architetture monolitiche rappresentano un ulteriore rischio per la disponibilità dell'applicazione, poiché la presenza di numerosi processi dipendenti e strettamente collegati aumenta l'impatto di un errore in un singolo processo.

Architettura monolitica vs. architettura di microservizi prt 2

- **Architettura monolitica:** è la diretta conseguenza dell'era dei mainframe IBM e del periodo di monopolio del sistema operativo Microsoft Windows, tradizionalmente adottata negli ambienti IT aziendali.

- **Microservizi:** inizialmente originati da community Open Source, sviluppatori di terze parti e start-up, dove programmatori indipendenti hanno messo a disposizione codici destinati ad estendere la funzionalità di base delle piattaforme di server web più diffuse. Ora, la maggior parte delle aziende IT più importanti rilascia i propri microservizi e i propri contributi ai progetti Open Source dove l'adozione degli standard è estesa a una varietà di mercati verticali e team di diversi settori sulla base di elementi fondamentali univoci. I microservizi si basano sullo stesso principio di innovazione degli sviluppatori tramite soluzioni di codice Open Source per applicazioni cloud, sebbene attualmente siano comuni anche i microservizi con licenza proprietaria.

Vantaggi dei microservizi

- **Innovazione rapida**: aziende e start-up possono immettere sul mercato soluzioni innovative più rapidamente rispetto a quanto sia possibile fare con un'architettura monolitica quando occorre creare nuove funzionalità per le applicazioni software. I clienti che utilizzano applicazioni web e mobili richiedono nuove caratteristiche. Le tecnologie innovative ricevono fondi attraverso l'utilizzo diffuso da parte degli utenti e l'adozione da parte delle aziende. Rimanere all'avanguardia sul fronte programmazione e sviluppo mediante l'integrazione di microservizi comporta vantaggi sia per le aziende IT che per le start-up.

- **Livelli superiori di automazione del data center**: gli sviluppatori preferiscono usare determinate piattaforme o determinati standard per il loro lavoro, incluso il supporto dei linguaggi di programmazione e dei database nelle applicazioni web/mobili con i microservizi. La connessione dei microservizi avviene mediante processi di scripting, ad esempio le API, che possono incrementare i livelli di automazione del data center.

Caratteristiche dei microservizi

- **Autonomi:** Ciascun servizio nell'architettura basata su microservizi può essere sviluppato, distribuito, eseguito e ridimensionato senza influenzare il funzionamento degli altri componenti. I servizi non devono condividere alcun codice o implementazione con gli altri. Qualsiasi comunicazione tra i componenti individuali avviene attraverso API ben definite.

- **Specializzati:** Ciascun servizio è progettato per una serie di capacità e si concentra sulla risoluzione di un problema specifico. Se, nel tempo, gli sviluppatori aggiungono del codice aggiuntivo a un servizio rendendolo più complesso, il servizio può essere scomposto in servizi più piccoli.

Esempi pratici

Possiamo declinare i microservizi anche a partire da una predefinita metodologia di strutturazione a servizi logici detta distribuita ma lo vedremo nelle prossime lezioni per ora andiamo ad approfondire l'argomento guardando del codice pratico ed esercitandoci sull'isolamento delle funzioni e nella distinzioni delle condizioni di Isolabilità DI UN BLOCCO DI CODICE

https://github.com/MaSTERmIKK/microserviceFORU

https://github.com/MaSTERmIKK/microservices-demoFORU

https://github.com/MaSTERmIKK/microserviceFORU2

Architetture REST e realizzazione di RESTFUL API

Prima di parlare delle architteture REST dobbiamo partire da alcuni basi comuni.

API è l'acronimo di *Application Programming Interface.*

È il modo con cui componenti *software* stabiliscono regole di comunicazione, un'interfaccia che consente il dialogo tra parti diverse di uno stesso *software* o tra parti di programmi diversi, nascono con lo scopo di permettere allo sviluppatore di usare lo stesso codice in diversi contesti e aiutano l'interoperabilità semantica.

- **Ci sono API che permettono al PC di eseguire le attività richieste chiamando le varie librerie di sistema**
- **Ci sono le API di Facebook, Twitter, Instagram, Google dette *API Oauth* e utilizzate per l'autenticazione**
- **Ci sono le API di Ebay, Amazon e dei vari *e-commerce* e altre ancora.**

Cos'è "REST"?

- **Representiational State Transfer (REST)** è un'architettura software che impone condizioni sul funzionamento di un'API. REST è stata inizialmente creata come linea guida per la gestione delle comunicazioni in una rete complessa come internet. Si può utilizzare l'architettura REST per supportare una comunicazione su vasta scala ad elevate prestazioni e affidabile. Si può facilmente implementare e modificare, apportando visibilità e portabilità multipiattaforma a qualsiasi sistema API.

- Gli sviluppatori API possono progettare API utilizzando diverse architetture. Le API che seguono lo stile architetturale REST sono chiamate API REST. I servizi Web che implementano le architetture REST sono chiamati servizi Web RESTful. Il termine RESTful API si riferisce generalmente alle API web RESTful. Comunque, è possibile utilizzare i termini REST API e RESTful API in maniera interscambiabile.

Come funzionano le API RESTful?

La funzione di base di una API RESTful è la stessa della navigazione su internet. Il client contatta il server utilizzando l'API quando richiede una risorsa. Gli sviluppatori API spiegano in che modo il client dovrebbe utilizzare l'API REST nella documentazione API dell'applicazione del server. Di seguito i passaggi generali per qualsiasi chiamata REST API:

- **Il client invia una richiesta al server. Il client segue la documentazione API per formattare la richiesta in modo che il server comprenda.**
- **Il server autentica il client e conferma che ha il diritto di effettuare la richiesta.**
- **Il server riceve la richiesta e la elabora internamente.**
- **Il server risponde al client. La risposta contiene informazioni che rivelano al client se la richiesta è avvenuta correttamente, la risposta include inoltre qualsiasi informazione richiesta dal client la dove le condizioni di accesso siano verificate e verificabili.**

La richiesta API REST e i dettagli della risposta variano leggermente in base a come gli sviluppatori progettano l'API.

Sistemi centralizati vs distributi

cos'è un sistema distribuito?

La locuzione sistema distribuito, in informatica, indica genericamente una tipologia di sistema informatico costituito da un insieme di processi interconnessi tra loro in cui le comunicazioni avvengono solo esclusivamente tramite lo scambio di opportuni messaggi. Ogni nodo del sistema esegue un insieme di componenti che comunicano tra di loro utilizzando uno strato software detto middleware che permette all'utente di percepire il sistema come un'unica entità. Con il termine processo si indica, in genere, una qualsiasi entità capace di comunicare con un qualsiasi altro processo e di eseguire un algoritmo distribuito. A differenza di un algoritmo tradizionale è necessario includere nella definizione di algoritmo distribuito anche i messaggi che vengono scambiati tra i vari processi, poiché anch'essi sono essenziali nell'esecuzione e nella terminazione dell'algoritmo. I sistemi distribuiti nascono da esigenze sia di tipo economico che tecnologico.

Sistemi centralizati vs distributi

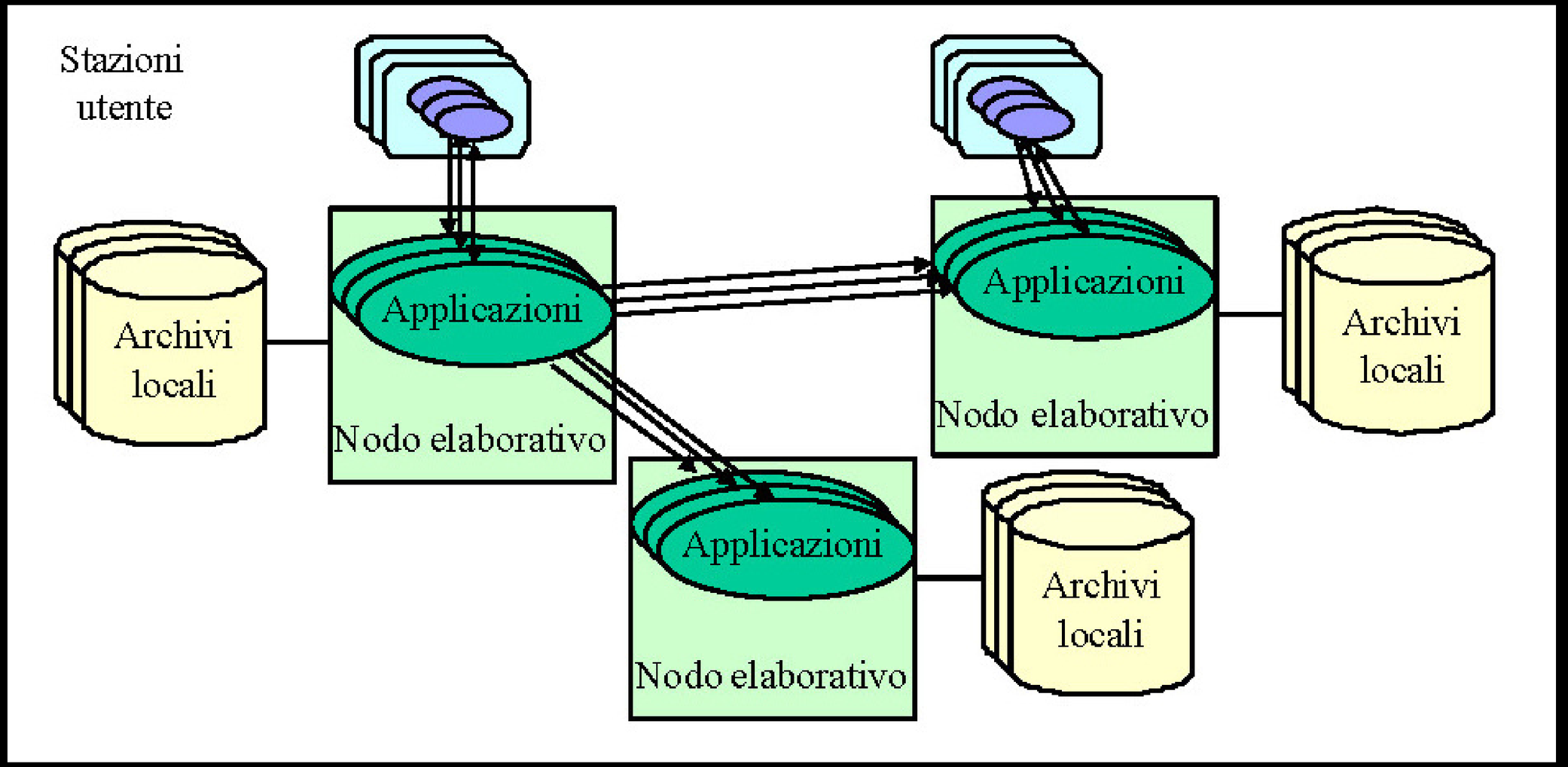
cos'è un sistema centralizato?

Si parla di sistema informatico centralizzato quando i dati e le applicazioni risiedono in un unico nodo elaborativo.
Sistema informatico centralizzato
Viceversa, si parla di sistema informatico distribuito quando almeno una delle seguenti due condizioni è verificata:

- **le applicazioni, fra loro cooperanti, risiedono su più nodi elaborativi (elaborazione distribuita);**
- **il patrimonio informativo, unitario, è ospitato su più nodi elaborativi (base di dati distribuita).**

In termini generali, quindi, un sistema distribuito è costituito da un insieme di applicazioni logicamente indipendenti che collaborano per il perseguimento di obiettivi comuni attraverso una infrastruttura di comunicazione hardware e software

Sistemi centralizati vs distributi



Cos'è Node JS

Come runtime JavaScript asincrono basato su eventi, Node.js è progettato per creare applicazioni di rete scalabili.

https://nodejs.org/en/docs/

La rete basata su thread è relativamente inefficiente e molto difficile da usare. Inoltre, gli utenti di Node.js sono liberi dalla preoccupazione di bloccare il processo in modo permanente, poiché non ci sono blocchi. Quasi nessuna funzione in Node.js esegue direttamente l'I/O, quindi il processo non si blocca mai tranne quando l'I/O viene eseguito utilizzando metodi sincroni della libreria standard Node.js. Poiché nulla blocca, i sistemi scalabili sono molto ragionevoli da sviluppare in Node.js.

Cos'è Node JS

Nel seguente esempio "hello world", molte connessioni possono essere gestite contemporaneamente. Ad ogni connessione, la richiamata viene attivata, ma se non c'è lavoro da fare, Node.js andrà in sospensione.

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World');
});

server.listen(port, hostname, () => {
console.log(`Server running at http://${hostname}:${port}/`);
 });
```

Cos'è e a cosa serve Express.js

Express.js è un framework open source per **Node.js**.

È stato progettato per creare applicazioni web e **API**. È un framework dinamico, altamente flessibile, leggero e veloce.
Inoltre è uno dei **framework più popolari** e più diffusi, consente agli sviluppatori di customizzare l'applicazione e di gestirla attraverso l'utilizzo di **rotte e middleware**.
Fornisce supporto per il **pattern MVC** per diversi **template engine** (come ejs, pug e handlebars), per estensioni che ne aumentano le funzionalità e per le **operazioni di debug**.

Express è anche un componente back-end fondamentale per le applicazioni stack MEAN, MERN e MEVN. Questi tre stack permettono di creare un'applicazione completa (front end, back end, database) utilizzando interamente Javascript e JSON.

Cos'è e a cosa serve Express.js

Andiamo ora a vedere tramite la documentazione ufficiale come utilizzare nelle sue principali funzioni il framework Express.js

https://expressjs.com/it/guide/routing.html

Qui invece andiamo ad approfondire come istallare il framework sulla nostra macchina

https://expressjs.com/it/starter/installing.html

Express.js vs le alternative

L'acronimo *MEAN* sta per MongoDB, Express, Angularjs e Node. *MERN* è una variazione che sostituisce Angular con React e *MEVN* con Vue

---

Oltre ad Express.js esistono altri framework per lo sviluppo di applicazioni web lato server con Node.js, ecco qualche esempio:

- **Adonis**
- **Koa**
- **Sails.js**
- **Next.js con React**

Continuiamo con NODE JS

Gli elementi principali in NODE.JS sono i moduli e i nodi

Che cos'è un modulo in Node.js? Considera i moduli come le stesse librerie JavaScript o più tecnicamente potremmo anche dire l'insieme di tutte le funzioni che desideri includere nella tua applicazione.
Oltre tutto i moduli possono essere di diversi tipi

Moduli integrati

Node.js ha una serie di moduli integrati che puoi utilizzare senza ulteriori installazioni che consentono a n.js nativamente di funzionare.

https://www.w3schools.com/nodejs/ref_modules.asp

Continuiamo con NODE JS

Per includere un modulo, bisogna utilizzare la KEYWORD *require()* funzione che dovrà essere accompagnata con il nome del modulo:

```
var http = require('http');
```

Ora la nostra applicazione avrà accesso al modulo HTTP ed è in grado di creare il server che poi noi andremo ad istanziare:

```
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

Continuiamo con NODE JS

Possiamo andare anche a creare dei moduli personali e includerli facilmente nelle tue applicazioni; Creiamo un modulo che restituisca la data e l'ora

```
exports.myDateTime = function () {
  return Date();
};
```

Utilizziamo la KEYWORD exports per rendere disponibili proprietà e metodi all'esterno del file del modulo.

Salviamo ora il codice sopra un file chiamato "primomodulo.js"

Continuiamo con NODE JS

Ora possiamo includere e utilizzare il modulo in uno qualsiasi dei nostri file Node.js. Usiamo il modulo "primomodulo.js" in un file Node:

```
var http = require('http');
var dt = require('./primomodulo');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("La data e l'ora al momento sono: " + dt.myDateTime());
  res.end();
}).listen(8080);
```

Si noti che utilizziamo ./ per individuare il modulo, ciò significa che il modulo si trova nella stessa cartella del file Node.js.

Continuiamo con NODE JS

Salviamo ora il nostro codice sopra un file chiamato "modulo_demo.js" e avvia il file Avvia demo_module.js:

C:\Users\Your Name> node modulo_demo.js

SIAMO APPENA ANDATI AD ESEGUIRE UN NODO LOGICO TRAMITE I MODULI, O LE SU FUNZIONI, CHE ABBIAMO DECISO DI IMPORTARE AL SUO INTERNO

Node.js come server Web

HTTP è il modulo che consente a Node.js di trasferire dati tramite Hyper Text Transfer Protocol (HTTP).Il modulo HTTP può creare un server HTTP che ascolta le porte del server e restituisce una risposta al client andiamo ora ad utilizzare il createServer() è un metodo che viene usato per creare un server HTTP:

```
var http = require('http');

http.createServer(function (req, res) {
  res.write('Serve c'è');     //Scriviamo quella che sarà la risposta del server
  res.end();      //Specifichiamo che il sistema da qui non avrà altri input client
}).listen(8080);
```

Node.js come server Web

Se la risposta dal server HTTP deve essere visualizzata come HTML, dovresti includere un'intestazione HTTP con il tipo di contenuto

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('ehy sono un testo!');
  res.end();
}).listen(8080);
```

Il primo argomento del res.writeHead() è il codice di stato, 200, significa che tutto è OK, il secondo argomento è un oggetto contenente le intestazioni di risposta che possono variare in base al contesto .

Node.js come server Web

Ricordiamoci però che a ogni funzione passata a http.createServer() ci sarà un argomento che rappresenta la richiesta del client, come esempio prendiamo l'oggetto http.IncomingMessage
Questo oggetto ha una proprietà chiamata "url" che contiene la parte dell'url che segue il nome del dominio

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

Node.js come server Web

Possiamo anche dividere una stringa di query e per farlo esistono moduli integrati per suddividere facilmente la stringa di query in parti leggibili, come il modulo URL.

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

Node.js system

```
var fs = require('fs');
```

Uso comune per il modulo File System: Leggi i file, Crea file, Aggiorna i file, Cancella file e Rinominare i fileIl fs.readFile() viene utilizzato per leggere i file sul tuo computer, creiamo ora il seguente file HTML ,che si trova nella stessa cartella di Node.js:

```
<html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

Node.js system

Crea un file Node.js che legge il file HTML

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
    fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
 });
}).listen(8080);
```

Node.js system

Creiamo un nuovo file usando il metodo appendFile():

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err) throw err;
 console.log('Saved!');
});
```

Node.js system

Il modulo File System dispone di metodi per la creazione di nuovi file:
fs.appendFile() fs.open() fs.writeFile()

Il fs.appendFile() aggiunge il contenuto specificato a un file.
Se il file non esiste, il file verrà creato:

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'ciao a tutti!', function (err) {
  if (err) throw err;
 console.log('Saved!');
});
```

Node.js system

Il fs.open() accetta un "flag" come secondo argomento, se il flag è "w" per "scrittura", il file specificato viene aperto per la scrittura. Se il file non esiste, viene creato un file vuoto:

```
var fs = require('fs');

fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
 console.log('Saved!');
});
```

Node.js system

Il fs.writeFile() sostituisce il file e il contenuto specificati se esiste.
Se il file non esiste, verrà creato un nuovo file, contenente il contenuto specificato:

```
var fs = require('fs');

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Node.js system

Il modulo File System dispone di metodi per l'aggiornamento dei file:
fs.appendFile() fs.writeFile()
Il fs.appendFile()metodo aggiunge il contenuto specificato

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {
  if (err) throw err;
 console.log('Updated!');
});
```

Node.js system

Per eliminare un file con il modulo File System, utilizzare il fs.unlink() elimina il file specificato:

```
var fs = require('fs');

fs.unlink('mynewfile1.txt', function (err) {
  if (err) throw err;
 console.log('File deleted!');
});
```

Node.js system

Per rinominare un file con il modulo File System, utilizzare il fs.rename() che rinomina il file specificato:

```
var fs = require('fs');

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
 console.log('File Renamed!');
});
```

Node.js NPM

NPM è un gestore di pacchetti per pacchetti Node.js o moduli se lo desideri.

www.npmjs.com ospita migliaia di pacchetti gratuiti da scaricare e utilizzare.

Il programma NPM viene installato sul tuo computer quando installi Node.js

NPM è già pronto per essere eseguito sul tuo computer!

Node.js NPM

Che cos'è un pacchetto?
Un pacchetto in Node.js contiene tutti i file necessari per un modulo.
I moduli sono librerie JavaScript che puoi includere nel tuo progetto.

Come scaricare un pacchetto è molto semplice.
Apri l'interfaccia della riga di comando e chiedi a NPM di scaricare il pacchetto che desideri.

```
C:\Users\Your Name>npm install upper-case
```

Node.js NPM

Che cos'è un pacchetto? Un pacchetto in Node.js contiene tutti i file necessari per un modulo. I moduli sono librerie JavaScript che puoi includere nel tuo progetto. Come scaricare un pacchetto è molto semplice. Apri l'interfaccia della riga di comando e chiedi a NPM di scaricare il pacchetto che desideri.

C:\Users\Your Name>npm install upper-case

NPM crea una cartella denominata "node_modules", in cui verrà inserito il pacchetto. Tutti i pacchetti che installerai in futuro verranno inseriti in questa cartella.

C:\Users\My Name\node_modules\upper-case

Node.js NPM

Una volta installato, il pacchetto è pronto per l'uso.Includi il pacchetto "maiuscolo" nello stesso modo in cui includi qualsiasi altro modulo:

```
var http = require('http');
var uc = require('upper-case');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc.upperCase("sono gigante!"));
  res.end();
}).listen(8080);
```

Node.js event

Ogni azione su un computer è un evento. Come quando viene stabilita una connessione o viene aperto un file.

Gli oggetti in Node.js possono attivare eventi, come l'oggetto readStream attiva eventi durante l'apertura e la chiusura di un file:

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
 console.log('The file is open');
});
```

Node.js event

Node.js ha un modulo integrato, chiamato "Eventi", in cui puoi creare, attivare e ascoltare i tuoi eventi. Per includere il modulo integrato Eventi utilizzare il require() metodo. Inoltre, tutte le proprietà ei metodi dell'evento sono un'istanza di un oggetto EventEmitter.

Per poter accedere a queste proprietà e metodi, crea un oggetto EventEmitter:

```
var events = require('events');
var eventEmitter = new events.EventEmitter()
```

Node.js event

Possiamo assegnare gestori di eventi ai tuoi eventi con l'oggetto EventEmitter. Nell'esempio seguente abbiamo creato una funzione che verrà eseguita quando viene attivato un evento "scream". Per attivare un evento, utilizzare il emit()

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

var myEventHandler = function () {
    console.log('I hear a scream!');
}

eventEmitter.on('scream', myEventHandler);
eventEmitter.emit('scream');
```

Fonti-pedia

Elenco delle Fonti:

Esempi semplificati by W3school

Documentazione sulle architteture by RedHat

Definizioni tecniche by Wikipedia

Elementi ingegneristici by Engineering

Documentazione ufficiale Node.js

Documentazione ufficiale Express

Tutti gli script sono su una repository con fork abilitato su (MasterMikk.github)