Micro services for Node js :

Microservice link : <https://blog.risingstack.com/why-you-should-start-using-microservices/>

* Excellent package management system.
* Minimal ceremony involved in publishing packages.(package.json , npm publish)
* End-to-end JavaScript stack can significantly accelerate development
* Excellent performance (v8) which also contributes to scalability
* Designed from the beginning for end-to-end asynchronous i/o to promote highly scalable services
* Encourages lightweight , narrowly focused packages.

microservices architecture focuses on a responsive-actor programming style

**Pros**

* Microservice architecture gives developers the freedom to independently develop and deploy services
* A microservice can be developed by a fairly small team
* Code for different services can be written in different languages (though many practitioners discourage it)
* Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)
* Easy to understand and modify for developers, thus can help a new team member become productive quickly
* The developers can make use of the latest technologies
* The code is organized around business capabilities
* Starts the web container more quickly, so the deployment is also faster
* When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application
* Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system)
* Easy to scale and integrate with third-party services
* No long-term commitment to technology stack

**Cons**

* Due to distributed deployment, testing can become complicated and tedious
* Increasing number of services can result in information barriers
* The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing
* Being a distributed system, it can result in duplication of effort
* When number of services increases, integration and managing whole products can become complicated
* In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system
* Developers have to put additional effort into implementing the mechanism of communication between the services
* Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams
* The architecture usually results in increased memory consumption
* Partitioning the application into microservices is very much an art