Scaling of Node JS application

Scalability

Reasons to scale the applications are:

- The workload
- to increase their availability and
- tolerance to failure.

Ways of Scaling

Horizontally Scaling:

Duplicate your application instance, enabling it to "cope with" a larger number of incoming connections. You can horizontally scale your Node.js app either across different machines or on a single multi-core machine.

Vertical Scaling:

If your scalability architecture needs involve nothing more than injecting more power or adding more memory with no particular changes are required on the code, then vertical scaling will work best.

One best way to increase the efficiency of the application is to spawn one process for each core of machine.

Its not good to spawn a number of processes bigger than the number of cores.

Native Cluster Module:

Node's cluster module scale up the application on a single machine.

It makes setting up child processes sharing server ports conveniently easy.

One of the process is known as "master" which is responsible to spawn the other child processes called as "workers". The connections are distributed using the round-robin strategy to all the workers, which has the service on the same port.

Ways of Scaling

PM2 Cluster mode:

If PM2 is used as process manager, which has the magic cluster features that helps you to scale your process across all the cores. The PM2 will act as the master process, and spawn N processes of application as workers with round-robin methods.

Network load balancing for multiple machines:

Scaling across multiple machines is similar with that of scaling on multiple cores, there will be multiple machines which will have one or more processes running and the balancer will redirect the traffic to different machine.

Scaling Strategies

1. Decomposition

For practically you'll be "juggling" with multiple microservices (although their size is of no significant importance, actually).

Or multiple applications, with different codebases (and in many cases, each one of them has its own UI and dedicated database).

And it's by services and functionalities that you'll be decomposing/scaling your Node.js app.

2. Splitting

"horizontal partitioning" or "sharding", if you prefer. This strategy involves splitting your app into multiple instances, each one responsible for a single, specific part of your app's data!

Word of caution: data partitioning calls for a lookup before you carry out each operation; this way you'll identify the right instance of the application to be used.

3. Cloning

easiest strategy

Just clone your Node.js back-end application, multiple times, and assign a specific part of the workload to each cloned instance!

Moreover, Node's cluster module makes cloning on a single server ideally easy to implement!

Useful Reference Links

- https://www.optasy.com/blog/how-scale-your-nodejs-app-best-strategies-and-built-tools-scalability