Technical Priorities

This list represents the current view of key technical priorities recognized by the project as important to ensure the ongoing success of Node.js. It is based on an understanding of the Node.js constituencies and their needs.

The initial version was created based on the work of the Next-10 team and the mini-summit on August 5th 2021.

They will be updated regularly and will be reviewed by the next-10 team and the TSC on a 6-month basis.

Modern HTTP

Base HTTP support is a key component of modern cloud-native applications and built-in support was part of what made Node.js a success in the first 10 years. The current implementation is hard to support and a common source of vulnerabilities. We must work towards an implementation which is easier to support and makes it easier to integrate the new HTTP versions (HTTP3, QUIC) and to support efficient implementations of different versions concurrently.

Suitable types for end-users

Using typings with JavaScript can allow a richer experience when using Visual Studio Code (or any other IDEs) environments, more complete documentation of APIs and the ability to identify and resolve errors earlier in the development process. These benefits are important to a large number of Node.js developers (maybe 50%). Further typing support may be important to enterprises that are considering expanding their preferred platforms to include Node.js. It is, therefore, important that the Node.js project work to ensure there are good typings available for the public Node.js APIs.

Documentation

The current documentation is great for experienced developers or people who are aware of what they are looking for. On the other hand, for beginners this documentation can be quite hard to read and finding the desired information is difficult. We must have documentation that is suitable for beginners to continue the rapid growth in use. This documentation should include more concrete examples and a learning path for newcomers.

WebAssembly

The use of WebAssembly has been growing over the last few years. To ensure Node.js continues to be part of solutions where a subset of the solution needs the performance that WebAssembly can deliver, Node.js must provide good support for running WebAssembly components along with the JavaScript that makes up the rest of the solution. This includes implementations of “host” APIs like WASI.

ESM

The CommonJS module system was one of the key components that led to the success of Node.js in its first 10 years. ESM is the standard that has been adopted as the equivalent in the broader JavaScript ecosystem and Node.js must continue to develop and improve its ESM implementation to stay relevant and ensure continued growth for the next 10 years.

Support for features from the latest ECMAScript spec

JavaScript developers are a fast moving group and need/want support for new ES JavaScript features in a timely manner. Node.js must continue to provide support for up to date ES versions to remain the runtime of choice and to ensure its continued growth for the next 10 years.

Observability

The ability to investigate and resolve problems that occur in applications running in production is crucial for organizations. Tools that allow people to observe the current and past operation of the application are needed to support that need. It is therefore important that the Node.js project work towards well understood and defined processes for observing the behavior of Node.js applications as well as ensuring there are well supported tools to implement those processes (logging, metrics and tracing). This includes support within the Node.js runtime itself (for example generating heap dumps, performance metrics, etc.) as well as support for applications on top of the runtime. In addition, it is also important to clearly document the use cases, problem determination methods and best practices for those tools.

Permissions/policies/security model

Organizations will only choose technologies that allow them to sufficiently manage risk in their production deployments. For Node.js to continue its growth in product/enterprise deployments we need to ensure that we help them manage that risk. We must have a well-documented security model so that consumers understand what threats are/are not addressed by the Node.js runtime. We also need to provide functions/features which help them limit attack surfaces even if it does not result in 100% protection as this will still help organizations manage their overall risk level.

Better multithreaded support

Today's servers support multiple threads of concurrent execution. Node.js deployments must be able to make full and efficient use of the available resources. The right answer is often to use technologies like containers to run multiple single threaded Node.js instances on the same server. However, there are important use cases where a single Node.js instance needs to make use of multiple threads to achieve a performant and efficient implementation. In addition, even when a Node.js instance only needs to consume a single thread to complete its work there can be issues. If that work is long running, blocking the event loop will interfere with other supporting work like metrics gathering and health checks. Node.js must provide good support for using multiple threads to ensure the continued growth and success of Node.js.

Single Executable Applications

Node.js often loses out to other runtimes/languages in cases where being able to package a single, executable application simplifies distribution and management of what needs to be delivered. While there are components/approaches for doing this, they need to be better documented and evangelized so that this is not seen as a barrier for using Node.js in these situations. This is important to support the expansion of where/when Node.js is used in building solutions.