Red Hat build of Node.js 18

Release Notes for Node.js 18

For use with Node.js 18 LTS

Last Updated: 2023-01-31

Red Hat build of Node.js 18 Release Notes for Node.js 18

For use with Node.js 18 LTS

Legal Notice

Abstract

This Release Note contains important information related to Node.js 18 LTS.

Table of Contents

PREFACE

Date of release: 2023-01-31

CHAPTER 1. REQUIRED INFRASTRUCTURE COMPONENT VERSIONS

The following infrastructure components are required when using the Red Hat build of Node.js. Except for components that are explicitly designated as supported, Red Hat does not provide support for these components.

Component name	Version
Nodeshift	2.1.1
npm 8	8.19.2
OpenShift Container Platform (OCP)[a]	3.11, 4.5
git	2.0 or later
oc command line tool	3.11 or later[b]

[a] OCP is supported by Red Hat

[b] The version of the **oc** CLI tool should correspond to the version of OCP that you are using.

CHAPTER 2. FEATURES

This section contains information about feature changes introduced in the Red Hat build of Node.js 18 release.

2.1. NEW AND CHANGED FEATURES

Node.js 18 LTS has the following new features and enhancements that the Red Hat build of Node.js supports.

For detailed changes in Node.js 18 LTS, see the upstream release notes and upstream documentation.

2.1.1. V8 JavaScript engine upgraded to v10.2

This release includes an upgrade of the V8 JavaScript engine to v10.2, which is part of Chromium 101.

The upgraded V8 JavaScript engine includes the following new features and enhancements:

- **findLast()** and **findLastIndex()** array methods
- **Intl.Locale** API improvements
- **Intl.supportedValuesOf** function
- Performance improvements for class fields and private class methods, which are now initialized as fast as ordinary property stores

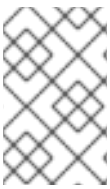
For more information about the changes that are available in the V8 JavaScript Engine, see the V8 blog.

2.1.2. Default values for HTTP timeouts

This release includes the following enhancements for HTTP timeouts:

- The **server.headersTimeout** property, which limits the amount of time that the parser waits to receive the complete HTTP headers, now has a default value of **60000** milliseconds (60 seconds).
- The **server.requestTimeout** property, which limits the amount of time that the server waits to receive the entire request from the client, now has a default value of **300000** milliseconds (5 minutes).

If these timeouts expire, the server responds with a **408** error and closes the connection without forwarding the request to the request listener.



NOTE

To protect against denial-of-service attacks in situations where a reverse proxy is not deployed in front of the server, ensure that you set these timeout values to a value other than zero.

2.1.3. Blob and BroadcastChannel APIs in the global scope

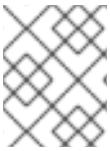
The following APIs are now fully supported and available as global objects:

- The **Blob** class encapsulates immutable, raw data that can be safely shared across multiple worker threads. **Blob** is a subclass of the **Buffer** class.
- The **BroadcastChannel** class enables asynchronous one-to-many communication with all other **BroadcastChannel** instances that are bound to the same channel name. **BroadcastChannel** extends the **EventTarget** class.

In the previous release, these APIs were Technology Preview features only.

2.2. DEPRECATED FEATURES

The following features are deprecated in the Red Hat build of Node.js 18 release.



NOTE

For more information about deprecated or removed features in this release, see the nodejs.org website.

2.2.1. Runtime deprecation of string coercion in fs methods for writing or appending files

This release includes the runtime deprecation of implicit object-to-string coercion when an object has its own **toString** property that is passed as a parameter in any of the following methods:

- **fs.write()**
- **fs.writeFile()**
- **fs.appendFile()**
- **fs.writeFileSync()**
- **fs.appendFileSync()**

As an alternative to string coercion, convert objects to primitive strings instead.

2.2.2. Option types coercion in dns.lookup and dnsPromises.lookup methods

This release removes the coercion of option types that are passed as a parameter in the **dns.lookup()** and **dnsPromises.lookup()** methods. If you use any of the following combinations of option type and value, Node.js now throws an **ERR_INVALID_ARG_TYPE** error:

- A non-nullish non-integer value for the **family** option
- A non-nullish non-number value for the **hints** option
- A non-nullish non-boolean value for the **all** option
- A non-nullish non-boolean value for the **verbatim** option

2.2.3. Runtime deprecation of multipleResolves

This release includes the runtime deprecation of the **multipleResolves** process event. For example:

```
process.on('multipleResolves', handler)
```

The **multipleResolves** event did not work with V8 promise combinators, which affected the usefulness and reliability of this event in tracking potential errors when using the **Promise** constructor. For example, the **Promise.race()** method could also trigger a **multipleResolves** event, which did not necessarily indicate an error.

2.2.4. Support for thenable objects

This release removes the ability to return **thenable** objects in stream implementation methods. The use of **thenable** objects could lead to unexpected issues if, for example, a user implemented the function in callback style but used an **async** method. This type of implementation resulted in an invalid mix of promise and callback semantics.

For example:

```
const w = new Writable({
  async final(callback) {
    await someOp();
    callback();
  },
});
```

As an alternative to **thenable** objects, use callbacks instead and avoid using the **async** function for stream implementation methods.

2.2.5. tls.parseCertString()

This release removes the **tls.parseCertString()** method, which was a parser helper to parse the certificate subject and issuer strings. The **tls.parseCertString()** method did not handle multi-value Relative Distinguished Names (RDNs) correctly, which could lead to incorrect representations and security issues.

Additionally, **_tls_common.translatePeerCertificate** no longer translates the subject and issuer properties.

NOTE

Earlier releases of Node.js recommended the use of the **querystring.parse()** method as an alternative to **tls.parseCertString()**. However, because **querystring.parse()** also does not handle all certificate subjects correctly, this release no longer recommends the use of **querystring.parse()** as an alternative.

2.3. TECHNOLOGY PREVIEW FEATURES

The following features are available as Technology Preview features in the Node.js 18 LTS release.

2.3.1. Fetch API

An experimental **fetch** API is available as a global object. The **fetch** API is based on Undici, which is an HTTP/1.1 client for Node.js, and the **node-fetch** module, which provides a Node.js implementation of the **Fetch** Web API.

For example:

```
const res = await fetch('https://nodejs.org/api/documentation.json');
if (res.ok) {
  const data = await res.json();
  console.log(data);
}
```

Because of this enhancement, the following global objects are now available for use:

- **fetch** (a browser-compatible version of the **fetch()** function)
- **FormData** (a browser-compatible version of the **FormData** interface)
- **Headers** (a browser-compatible version of the **Headers** interface)
- **Request** (a browser-compatible version of the **Request** interface)
- **Response** (a browser-compatible version of the **Response** interface)

You can disable the **fetch** API by using the **--no-experimental-fetch** command-line option.

2.3.2. Web Streams API in the global scope

The Web Streams API was introduced as a Technology Preview feature in the Red Hat build of Node.js 16 release. In this release, Node.js now exposes the Web Streams API in the global scope. This supersedes the behavior in the previous release where the Web Streams API was only accessible by using the **stream/web** core module.

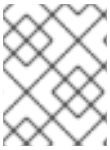
Because of this enhancement, the following APIs are now available as global objects:

ReadableStream, **ReadableStreamDefaultReader**, **ReadableStreamBYOBReader**, **ReadableStreamBYOBRequest**, **ReadableByteStreamController**, **ReadableStreamDefaultController**, **TransformStream**, **TransformStreamDefaultController**, **WritableStream**, **WritableStreamDefaultWriter**, **WritableStreamDefaultController**, **ByteLengthQueuingStrategy**, **CountQueuingStrategy**, **TextEncoderStream**, **TextDecoderStream**, **CompressionStream**, **DecompressionStream**

2.3.3. ESM Loader Hooks API supports multiple custom loaders

The ESM Loader Hooks API was introduced as a Technology Preview feature in the Red Hat build of Node.js 16 release. The ESM Loader Hooks API now supports multiple custom loaders.

To enable multiple loaders to work together, this feature uses a process called *chaining*, which is analogous to a promise chain. For example, **first-loader** calls **second-loader**, **second-loader** calls **third-loader**, and so on.



NOTE

If a custom loader does not intentionally call the next loader in the chain, the custom loader must signal a short circuit.

For more information, see the Node.js ECMAScript modules documentation.

2.3.4. Watch mode

You can now run Node.js applications in watch mode by using the **node --watch** option.

Running an application in watch mode means that the process restarts if you modify an imported file.

2.4. SUPPORTED ARCHITECTURES

Node.js builder images and RPM packages are available and supported for use with the following CPU architectures:

- AMD x86_64
- ARM64
- IBM Z (s390x) in the OpenShift environment
- IBM Power Systems (ppc64le) in the OpenShift environment

CHAPTER 3. RELEASE COMPONENTS

- Node.js 18 Builder Image for RHEL 8
- Node.js 18 Universal Base Image 8
- Node.js 18 Minimal Stand-alone Image for RHEL 8
- Node.js 18 Minimal Universal Base Image 8
- Node.js 18 Builder Image for RHEL 9
- Node.js 18 Universal Base Image 9
- Node.js 18 Minimal Stand-alone Image for RHEL 9
- Node.js 18 Minimal Universal Base Image 9

CHAPTER 4. FIXED ISSUES

This release incorporates all of the fixed issues in the community release of Node.js 18 LTS.

CHAPTER 5. KNOWN ISSUES

There are no known issues affecting this release.

CHAPTER 6. KNOWN ISSUES AFFECTING REQUIRED INFRASTRUCTURE COMPONENTS

There are no known issues affecting infrastructure components required by this release.

CHAPTER 7. ADVISORIES RELATED TO THIS RELEASE

The following advisories have been issued to document enhancements, bug fixes, and CVE fixes included in this release.

- RHSA-2022:8832
- RHSA-2022:8833
- RHBA-2022:7843
- RHBA-2022:8458