Red Hat

Red Hat build of Node.js 20

Release Notes for Node.js 20

For use with Node.js 20 LTS

Last Updated: 2024-04-02

Red Hat build of Node.js 20 Release Notes for Node.js 20

For use with Node.js 20 LTS

Abstract

This Release Note contains important information related to Node.js 20 LTS.

Table of Contents

PREFACE

Date of release: 2024-03-25

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 10	10.1.0
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 in the Red Hat build of Node.js 20 release.

2.1. NEW AND CHANGED FEATURES

Node.js 20 LTS has the following new features and enhancements that the Red Hat build of Node.js supports.

For detailed changes in Node.js 20 LTS, see the upstream community release notes and documentation.

2.1.1. V8 JavaScript engine upgraded to v11.3

This release includes an upgrade of the V8 JavaScript engine to v11.3, which is part of Chromium 113.

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

- String.prototype.isWellFormed and String.prototype.toWellFormed methods
- **RegExp** object **v** flag with set notation and properties of strings
- Resizable **ArrayBuffer** and growable **SharedArrayBuffer** objects
- WebAssembly tail calls

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

2.1.2. Full support for test runner module

Red Hat build of Node.js 20 provides the test runner module as a stable feature that you can use in production environments. In earlier releases, the test runner module was available as an experimental feature only.

The test runner module is not intended to replace full-featured test frameworks such as Jest or Mocha. The test runner module offers a quick and easy way to write and run a test suite without needing to install additional dependencies.

In Red Hat build of Node.js 20, the test runner module includes the following types of enhancements:

- Command-line test runner that you can invoke by using the **node --test** flag
- Configurable and custom test reporters by using the **--test-reporter** flag
- Experimental test coverage by using the **--experimental-test-coverage** flag
- Mocking capabilities

For more information, see the Node.js Test Runner documentation.

2.1.3. Custom ESM loader hooks run on dedicated thread

In Red Hat build of Node.js 20, ECMAScipt modules (ESM) hooks that you supply through a loader (for example, **--experimental-loader=myhook.mjs**) run on a dedicated thread, which is separate from the main application thread. By providing a separate scope for loaders, this enhancement protects the application code from potential contamination.

2.1.4. Synchronous import.meta.resolve() function

In Red Hat build of Node.js 20, the **import.meta.resolve()** function returns values synchronously. However, depending on your preferences, you can still define custom loader **resolve** hooks as either synchronous or asychronous functions. Even if asynchronous **resolve** hooks are loaded, the **import.meta.resolve()** function returns values synchronously for application code.

2.1.5. Web Crypto API use of WebIDL converters

In Red Hat build of Node.js 20, the Web Crypto API coerces and validates function arguments based on their WebIDL definitions, similar to other implementations of the Web Crypto API. This enhancement helps to improve interoperability between Node.js and browser implementations of the Web Crypto API.

2.1.6. Property import.meta.resolve no longer relies on CLI flag

From Red Hat build of Node.js 20 onward, the **import.meta.resolve(specifier)** property no longer relies on the use of the **--experimental-import-meta-resolve** command-line interface (CLI) flag by default. You can now use the **import.meta.resolve(specifier)** property to get an absolute string to which the specifier string resolves, which is similar to the **require.resolve** functionality in CommonJS modules. This enhancement helps to align Node.js with browsers and other server-side runtimes.

For more information, see the Node.js import.meta.resolve documentation.

2.1.7. Method module.register for module customization hooks

Red Hat build of Node.js 20 provides a new **module.register(specifier[, parentURL][, options])** method that is part of the **node:module** API. You can use this new method to specify a file that exports module customization hooks, pass data to the hooks, and establish communication channels with these hooks. This enhancement supersedes the behavior in previous releases, which required the use of an **--experimental-loader** flag to specify the file that exported the hooks.

To ensure that customization hooks are registered before any application code runs, consider using the **--import** flag when running applications that use the register functionality.

For example:

```
node --import ./file-that-calls-register.js ./app.js
```

For more information, see the Node.js module.register documentation.

2.1.8. Module customization load hook support for CommonJS modules

Red Hat build of Node.js 20 enables authors of module customization hooks to handle both ESM and CommonJS sources in the **load** hook. This enhancement is available for CommonJS modules that are referenced by using an **import** or **require** statement when the main entry point of the application is handled by the ESM loader. For example, in situations where the entry point is an ESM file or when you use the **--import** flag, this enhancement is relevant. This helps to simplify the customization of the Node.js loading process, because package authors can perform additional Node.js customization without needing to rely on deprecated APIs, such as **require.extensions**.

For more information, see the Node.js Hooks: load documentation.

2.1.9. Enhanced stream performance

Red Hat build of Node.js 20 enhances the performance of readable and writable streams. This enhancement includes improvements such as reduced memory overhead, improved **async** iterator consumption for readable streams, and improved **pipeTo** consumption for readable streams.

2.2. DEPRECATED FEATURES

The following features are deprecated in the Red Hat build of Node.js 20 release.

NOTE

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

2.2.1. Runtime deprecation of url.parse() method with invalid ports

This release includes the runtime deprecation of **url.parse()** method calls that contain invalid ports. In previous releases, the **url.parse()** method accepted URLs that contained non-numeric port values, which could result in hostname spoofing with unexpected inputs. In this release, if a URL contains a non-numeric port value, the **url.parse()** method emits a warning.

For more information, see the Node.js **url.parse** documentation.

2.3. TECHNOLOGY PREVIEW FEATURES

The following features are available as Technology Preview features in the Node.js 20 LTS release.

2.3.1. Experimental WASI feature enhancements

The experimental WebAssembly System Interface (WASI) feature includes the following enhancements in this release:

- WASI no longer requires use of the **experimental-wasi-unstable-preview1** command-line interface (CLI) flag to enable this feature. This enhancement helps to make WASI easier to use.
- WASI now requires that any new **WASI()** calls include the **version** option to request a specific WASI version. Because Node.js supports different versions of WASI and the **version** option does not have a default value, you must ensure that all new **WASI()** calls in your applications request a specific version. Otherwise, an error results.

For more information, see the Node.js WebAssembly System Interface (WASI) documentation .

2.3.2. Permission model

Red Hat build of Node.js 20 introduces an experimental Permission Model feature. This feature enables developers to restrict access to specific resources such as file system operations, child process spawning, and worker thread creation during program execution. This means that you can prevent your applications from accessing or modifying sensitive data or running potentially harmful code.

You can enable the Permission Model feature by using a **--experimental-permission** CLI flag. By enabling this feature, you automatically restrict access to all available permissions.

To manage permissions, you can use the following CLI flags:

- To manage file system permissions, use the **--allow-fs-read** and **--allow-fs-write** flags.

- To manage child process permissions, use the **--allow-child-process** flag.
- To manage worker thread permissions, use the **--allow-worker** flag.

For more information, see the Node.js Permission Model documentation .

2.3.3. Tracing channels

Red Hat build of Node.js 20 introduces the **TracingChannel** class as an experimental extension to the Diagnostics Channel feature. Tracing channels help to group diagnostic channels, which represent different points in the execution lifecycle of a single traceable action, for use in generating and consuming trace data. Tracing channels therefore help to formalize and simplify the process of producing events for tracing application flow.

For more information, see the Node.js TracingChannel documentation.

2.3.4. Mock Timers

Red Hat build of Node.js 20 introduces an experimental Mock Timers feature that enables developers to write more predictable and reliable tests for functionality that relies on timers. Timer mocking is a commonly used technique to simulate and test timer behavior without needing to wait for the specified time intervals. The Mock Timers feature includes a **MockTimers** class that enables you to simulate calls to the **setTimeout()** and **setInterval()** methods from global objects and from the **node:timers** and **node:timers/promises** API.

The Mock Timers feature provides a simple API to advance time, enable specific timers, and release all timers.

For example:

```
import assert from 'node:assert';
import { test } from 'node:test';

test('mocks setTimeout to be executed synchronously without having to actually wait for it', (context)
=> {
  const fn = context.mock.fn();
  // Optionally choose what to mock
  context.mock.timers.enable(['setTimeout']);
  const nineSecs = 9000;
  setTimeout(fn, nineSecs);

  const threeSeconds = 3000;
  context.mock.timers.tick(threeSeconds);
  context.mock.timers.tick(threeSeconds);
  context.mock.timers.tick(threeSeconds);

  assert.strictEqual(fn.mock.callCount(), 1);
```

For more information, see the Node.js Mocking: Timers documentation.

2.3.5. Built-in .env file support

Red Hat build of Node.js 20 introduces an experimental feature for configuring environment variables in a **.env** file that is passed to the Node.js application at startup. Similar to **.ini** file format, each line in a **.env** configuration file contains a key-value pair for a specific environment variable. For example:

```
PASSWORD=nodejs
```

You can initialize and run your Node.js application with predefined environment variables by using the **--env-file** flag. For example:

```
node --env-file=myconfig.env myapp.js
```

When you initialize your application as shown in the preceding example, you can access the relevant environment variables by using the **process.env** property. For example, to access the **PASSWORD=node.js** environment variable, you can use the **process.env.PASSWORD** property.

This enhancement also enables you to define the **NODE_OPTIONS** environment variable in a **.env** configuration file, which eliminates the need to include **NODE_OPTIONS** in the **package.json** 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 20 Builder Image for RHEL 8
- Node.js 20 Universal Base Image 8
- Node.js 20 Minimal Stand-alone Image for RHEL 8
- Node.js 20 Minimal Universal Base Image 8
- Node.js 20 Builder Image for RHEL 9
- Node.js 20 Universal Base Image 9
- Node.js 20 Minimal Stand-alone Image for RHEL 9
- Node.js 20 Minimal Universal Base Image 9

CHAPTER 4. FIXED ISSUES

This release incorporates all of the fixed issues in the community release of Node.js 20 LTS.

4.1. ENHANCED MEMORY MANAGEMENT IN VM MODULE APIS

Red Hat build of Node.js 20 resolves some long-standing memory leaks and use-after-free issues in the following Virtual Machine (VM) module APIs that support the **importModuleDynamically** option:

- vm.Script
- vm.compileFunction
- vm.SyntheticModule
- vm.SourceTextModule

This enhancement can enable affected users to upgrade from earlier versions of Red Hat build of Node.js.

CHAPTER 5. KNOWN ISSUES

There are no known issues affecting this release.

CHAPTER 6. KNOWN ISSUES AFFECTING REQUIRED INFRASTRUCTURE COMPONENTS

The following issues are known to affect infrastructure components required by this release.

6.1. NODE.JS PACKAGE MANAGER DOES NOT HANDLE SIGNALS CORRECTLY

Description

In Red Hat build of Node.js 20, the Node.js package manager (**npm**) is not sending signals to child processes. This issue means that Node.js applications cannot shut down correctly.

Cause

An issue in **npm** 9.6.7 and later versions stops **npm** from forwarding **SIGINT** and **SIGTERM** signals to a child process. This issue affects Node.js versions that require use of **npm** 9.6.7 or later. The Node.js community has been investigating a fix for this **npm** issue.

Workaround

No workaround is currently available. Red Hat is investigating a fix for this issue in Red Hat build of Node.js 20.

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-2023:7205
- RHEA-2023:6529
- RHEA-2023:7249
- RHEA-2023:7252
- RHBA-2023:6753
- RHBA-2023:7223
- RHBA-2023:7271
- RHBA-2023:7514
- RHBA-2023:7611
- RHBA-2023:7799