Migrating from v3 to v4

Looking for the v3 docs?

> Have you run into something that's not covered here? Add your changes to GitHub!

Introduction

This is a reference for upgrading your site from Gatsby 3 to Gatsby 4. Version 4 introduces big performance improvements of up to 40% build time reduction and two new rendering options: Deferred Static Generation (DSG) and Server-Side Rendering (SSR). If you're curious what's new, head over to the v4.0 release notes.

Table of Contents

Handling Deprecations

Before upgrading to v4 we highly recommend upgrading `gatsby` (and all plugins) to the latest v3 version. Some changes required for Gatsby 4 could be applied incrementally to the latest v3 which should contribute to smoother upgrade experience.

> Use `npm outdated` or `yarn upgrade-interactive` for automatic upgrade to the latest v3 release.

After upgrading, run `gatsby build` and look for deprecation messages in the build log. Follow instructions to fix those deprecations.

Updating Your Dependencies

Next, you need to update your dependencies to v4.

Update Gatsby version

You need to update your `package.json` to use the `latest` version of Gatsby.

```
{
  "dependencies": {
    "gatsby": "^4.0.0"
  }
}
```

Or run

```
npm install gatsby@latest
```

Please note: If you use **npm 7** you'll want to use the `--legacy-peer-deps` option when following the instructions in this guide. For example, the above command would be:

```
npm install gatsby@latest --legacy-peer-deps
```

Update Gatsby related packages

Update your `package.json` to use the `latest` version of Gatsby related packages. You should upgrade any package name that starts with `gatsby-*`. Note, this only applies to plugins managed in the gatsbyjs/gatsby repository. If you're using community plugins, they might not be upgraded yet. Please check their repository for the current status.

Updating community plugins Using community plugins, you might see warnings like these in your terminal:

```
warning Plugin gatsby-plugin-acme is not compatible with your gatsby version 4.0.0 - It requ
```

If you are using npm 7, the warning may instead be an error:

```
npm ERR! ERESOLVE unable to resolve dependency tree
```

This is because the plugin needs to update its `peerDependencies` to include the new version of Gatsby (see section for plugin maintainers). While this might indicate that the plugin has incompatibilities, in most cases they should continue to work. When using npm 7, you can pass the `--legacy-peer-deps` to ignore the warning and install anyway. Please look for already opened issues or PRs on the plugin's repository to see the status. If you don't see any, help the maintainers by opening an issue or PR yourself! :)

Handling Breaking Changes

This section explains breaking changes that were made for Gatsby 4. Some of those changes had a deprecation message in v3. In order to successfully update, you'll need to resolve these changes.

Minimal Node.js version 14.15.0

We are dropping support for Node 12 as a new underlying dependency (`lmdb-store`) is requiring `>=14.15.0`. See the main changes in Node 14 release notes.

Check Node's releases document for version statuses.

Disallow schema-related APIs in `sourceNodes`

You can no longer use `createFieldExtension`, `createTypes` & `addThirdPartySchema` actions inside the `sourceNodes` lifecycle. Instead, move them to `createSchemaCustomization` API. Or alternatively use `createResolvers` API.

The reasoning behind this is that this way Gatsby can safely build the schema and run queries in a separate process without running sourcing.

Change arguments passed to `touchNode` action

For Gatsby v2 & v3 the `touchNode` API accepted `nodeId` as a named argument. This now has been changed in favor of passing the full `node` to the function.

```
exports.sourceNodes = ({ actions, getNodesByType }) => {
  const { touchNode } = actions

- getNodesByType("YourSourceType").forEach(node => touchNode({ nodeId: node.id }))
+ getNodesByType("YourSourceType").forEach(node => touchNode(node))
}
```

In case you only have an ID at hand (e.g. getting it from cache), you can use the `getNode()` API:

```
exports.sourceNodes = async ({ actions, getNode, getNodesByType, cache }) => {
  const { touchNode } = actions
  const myNodeId = await cache.get("some-key")

  touchNode(getNode(myNodeId)) // highlight-line
}
```

Change arguments passed to `deleteNode` action

For Gatsby v2 & v3, the `deleteNode` API accepted `node` as a named argument. This now has been changed in favor of passing the full `node` to the function.

```
exports.onCreateNode = ({ actions, node }) => {
  const { deleteNode } = actions

- deleteNode({ node })
+ deleteNode(node)
}
```

Replace `@nodeInterface` with interface inheritance

For Gatsby v2 & v3, `@nodeInterface` was the recommended way to implement queryable interfaces. Now it is changed in favor of interface inheritance:

```
exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`
-   interface Foo @nodeInterface
+   interface Foo implements Node
    {
      id: ID!
    }
  `)
}
```

Use `onPluginInit` API to share context with other lifecycle APIs

Sites and in particular plugins that rely on setting values on module context to access them later in other lifecycles will need to use `onPluginInit`. This is also the case for when you use `onPreInit` or `onPreBootstrap`. The `onPluginInit` API will run in each worker as it is initialized and thus each worker then has the initial plugin state.

Here's an example of a v3 plugin fetching a GraphQL schema at the earliest stage in order to use it in later lifecycles:

```
const stateCache = {}

const initializePlugin = async (args, pluginOptions) => {
  const res = await getRemoteGraphQLSchema()
  const graphqlSdl = await generateSdl(res)
  const typeMap = await generateTypeMap(res)

  stateCache['sdl'] = graphqlSdl
  stateCache['typeMap'] = typeMap
}

// highlight-start
exports.onPreBootstrap = async (args, pluginOptions) => {
  await initializePlugin(args, pluginOptions)
}
// highlight-end

exports.createResolvers = ({ createResolvers }, pluginOptions) => {
  const typeMap = stateCache['typeMap']

  createResolvers(generateResolvers(typeMap))
```

```
}

exports.createSchemaCustomization = ({ actions }, pluginOptions) => {
  const { createTypes } = actions

  const sdl = stateCache['sdl']

  createTypes(sdl)
}
```

In order to make this work for Gatsby 4 & Parallel Query Running the logic inside `onPreBootstrap` must be moved to `onPluginInit`:

```
// Rest of initializePlugin stays the same

exports.onPluginInit = async (args, pluginOptions) => {
  await initializePlugin(args, pluginOptions)
}

// Schema APIs stay the same
```

This also applies to using the `reporter.setErrorMap` function. It now also needs to be run inside `onPluginInit` instead of in `onPreInit`.

```
const ERROR_MAP = {
  10000: {
    text: context => context.sourceMessage,
    level: "ERROR",
    category: "SYSTEM",
  },
}

// highlight-start
exports.onPluginInit = ({ reporter }) => {
  reporter.setErrorMap(ERROR_MAP)
}
// highlight-end

const getDataFromAPI = async ({ reporter }) => {
  let data
  try {
    const res = await requestAPI()
    data = res
  } catch (error) {
    reporter.panic({
      id: "10000",
      context: {
        sourceMessage: error.message,
```

```
      },
    })
  }

  return data
}
```

Remove obsolete flags

Remove the flags for `QUERY_ON_DEMAND`, `LAZY_IMAGES`, `FUNCTIONS`, `DEV_WEBPACK_CACHE` and `PRESERVE_WEBPACK_CACHE` from `gatsby-config`. Those features are a part of Gatsby core now and don't need to be enabled nor can't be disabled using those flags.

Do not create nodes in custom resolvers

The most typical scenario is when people use `createRemoteFileNode` in custom resolvers to lazily download only those files that are referenced in page queries.

It is a well-known workaround aimed for build time optimization, however it breaks a contract Gatsby establishes with plugins and prevents us from running queries in parallel and makes other use-cases harder (like using GraphQL layer in functions).

The recommended approach is to always create nodes in `sourceNodes`. We are going to come up with alternatives to this workaround that will work using `sourceNodes`. It is still being worked on, please post your use-cases and ideas in this discussion to help us shape this new APIs.

If you've used this with `gatsby-source-graphql`, please switch to Gatsby GraphQL Source Toolkit. Generally speaking you'll want to create your own source plugin to fully support such use cases.

You can also learn more about this in the migration guide for source plugins.

Changes to built-in types

The built-in type `SitePage` now returns the `pageContext` key as `JSON` and won't infer any other information anymore. The `SitePlugin` type now has two new keys: `pluginOptions: JSON` and `packageJson: JSON`.

Field `SitePage.context` is no longer available in GraphQL queries
Before v4 you could query specific fields of the page context object:

```
{
  allSitePage {
    nodes {
      context {
        foo
```

```
      }
    }
  }
}
```

Starting with v4, `context` field is replaced with `pageContext` of type `JSON`. It means you can't query individual fields of the context. The new query would look like this:

```
{
  allSitePage {
    nodes {
      pageContext # returns full JS object passed to `page.context` in `createPages`
    }
  }
}
```

If you still need to query individual `context` fields - you can workaround it by providing a schema for `SitePage.context` manually:

```
// Workaround for missing sitePage.context:
exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`
    type SitePage implements Node {
      context: SitePageContext
    }
    type SitePageContext {
      foo: String
    }
  `)
}
```

Removal of `gatsby-admin`

You can no longer use `gatsby-admin` (activated with environment variable `GATSBY_EXPERIMENTAL_ENABLE_ADMIN`) as we removed this functionality from `gatsby` itself. We didn't see any major usage and don't plan on developing this further in the foreseeable future.

Removal of `process.env.GATSBY_BUILD_STAGE`

This environment variable was internally used by `gatsby-preset-gatsby`. If you're using it you now must pass the `stage` as an option to the preset.

Windows-specific: No support for WSL1

With the introduction of `lmdb-store` instances running WSL1 sadly won't work anymore. You'll see errors like `Error: MDB_BAD_RSLOT: Invalid reuse of`

`reader locktable slot` or similar. This is an upstream issue that we can't fix and we recommend updating to WSL2 (Comparison of WSL1 & WSL2).

Gatsby related packages

Breaking Changes in plugins that we own and maintain.

`gatsby-plugin-feed`

- The `feeds` option is required now
- The `serialize` & `title` key inside the `feeds` option is required now. Please define your own `serialize` function if you used the default one until now.

`gatsby-plugin-sharp`

- The `sizeByPixelDensity` option was removed

`gatsby-remark-images`

- The `sizeByPixelDensity` option was removed

`gatsby-remark-images-contentful`

- The `sizeByPixelDensity` option was removed

`gatsby-transformer-json` While technically the change that was made is a bugfix, it can be a breaking change in your setup. Previously, if an item contained an `id` key it was used internally to create the node and track it. This led to cases where different files (with partially the same `id`) had missing items.

The new behavior now is that `gatsby-transformer-json` automatically transforms the `id` key to `jsonId` and uses an UUID internally for the actual `id` field on the node. This way the bug with missing items is fixed.

If you use `id` in your GraphQL queries, swap it out with `jsonId`.

`gatsby-transformer-yaml` While technically the change that was made is a bugfix, it can be a breaking change in your setup. Previously, if an item contained an `id` key it was used internally to create the node and track it. This led to cases where different files (with partially the same `id`) had missing items.

The new behavior now is that `gatsby-transformer-yaml` automatically transforms the `id` key to `yamlId` and uses an UUID internally for the actual `id` field on the node. This way the bug with missing items is fixed.

If you use `id` in your GraphQL queries, swap it out with `yamlId`.

Future Breaking Changes

This section explains deprecations that were made for Gatsby 4. These old behaviors will be removed in v5, at which point they will no longer work. For now, you can still use the old behaviors in v4, but we recommend updating to the new signatures to make future updates easier.

`nodeModel.runQuery` is deprecated

Use `nodeModel.findAll` and `nodeModel.findOne` instead. Those are almost a drop-in replacement for `runQuery`:

```
const entries = await nodeModel.runQuery({
  type: `MyType`,
  query: {
    /* ... */
  },
  firstOnly: false,
})
// is the same as:
const { entries } = await nodeModel.findAll({
  type: `MyType`,
  query: {
    /* ... */
  },
})

const node = await nodeModel.runQuery({
  type: `MyType`,
  query: {
    /* ... */
  },
  firstOnly: true,
})
// is the same as:
const node = await nodeModel.findOne({
  type: `MyType`,
  query: {
    /* ... */
  },
})
```

The two differences are:

1. `findAll` supports `limit`/`skip` arguments. `runQuery` ignores them when passed.
2. `findAll` returns an object with `{ entries: GatsbyIterable, totalCount: () => Promise<number> }` while `runQuery` returns a

plain array of nodes

```
// Assuming we have 100,000 nodes of the type `MyQuery`,
// the following returns an array with all 100,000 nodes
const entries = await nodeModel.runQuery({
  type: `MyType`,
  query: { limit: 20, skip: 10 },
})

// findAll returns 20 entries (starting from 10th)
// and allows to get total count using totalCount() if required:
const { entries, totalCount } = await nodeModel.findAll({
  type: `MyType`,
  query: { limit: 20, skip: 10 },
})
const count = await totalCount()
```

If you don't pass `limit` and `skip`, `findAll` returns all nodes in `{ entries }` iterable. Check out the source code of `GatsbyIterable` for usage.

The `GatsbyIterable` has some convenience methods similar to arrays, namely: `concat`, `map`, `filter`, `slice`, `deduplicate`, `forEach`, `mergeSorted`, `intersectSorted`, and `deduplicateSorted`. You can use these instead of first creating an array from `entries` (with `Array.from(entries)`) which should be faster for larger datasets.

Furthermore, you can directly return `entries` in GraphQL resolvers.

```
// Example: Directly return `entries`
resolve: async (source, args, context, info) => {
  const { entries } = await context.nodeModel.findAll({
    query: {
      filter: {
        frontmatter: {
          author: { eq: source.email },
          date: { gt: "2019-01-01" },
        },
      },
    },
    type: "MarkdownRemark",
  })
  return entries
}

// Example: Use .filter on the iterable
resolve: async (source, args, context, info) => {
  const { entries } = await context.nodeModel.findAll({ type: `BlogPost` })
  return entries.filter(post => post.publishedAt > Date.UTC(2018, 0, 1))
```

```
}

// Example: Convert to array to use methods not available on iterable
resolve: async (source, args, context, info) => {
  const { entries } = await context.nodeModel.findAll({
    type: "MarkdownRemark",
  })
  const posts = entries.filter(post => post.frontmatter.author === source.email)
  return Array.from(posts).length
}
```

`nodeModel.getAllNodes` is deprecated

Gatsby v4 uses persisted data store for nodes (using lmdb-store) and fetching an unbounded number of nodes won't play well with it in the long run.

We recommend using `nodeModel.findAll` instead as it at least returns an iterable and not an array.

```
// replace:
const entries = nodeModel.getAllNodes(`MyType`)

// with
const { entries } = await nodeModel.findAll({ type: `MyType` })
```

However, we highly recommend restricting the number of fetched nodes at once. So this is even better:

```
const { entries } = await nodeModel.findAll({
  type: `MyType`,
  query: { limit: 20 },
})
```

`___NODE` convention is deprecated

Gatsby was using `___NODE` suffix of node fields to magically detect relations between nodes. But starting with Gatsby v2.5 `@link` directive is a preferred method:

Before:

```
exports.sourceNodes = ({ actions }) => {
  actions.createNode({
    // ...required node fields
    author___NODE: userNode.id,
    internal: { type: `BlogPost` /*...*/ },
  })
}
```

After:

```
exports.sourceNodes = ({ actions }) => {
  actions.createNode({
    // ...required node fields
    author: userNode.id,
    internal: { type: `BlogPost` /*...*/ },
  })
}
exports.createSchemaCustomization = ({ actions }) => {
  actions.createTypes(`
    type BlogPost implements Node {
      author: User @link
    }
  `)
}
```

To find out if you're using this old syntax you can run `gatsby develop --verbose` or `gatsby build --verbose` and warnings will be shown.

Follow this how-to guide for up-to-date guide on sourcing and defining data relations.

For Plugin Maintainers

In most cases, you won't have to do anything to be v4 compatible. The underlying changes mostly affect **source** plugins. But one thing you can do to be certain your plugin won't throw any warnings or errors is to set the proper peer dependencies.

Please also note that some of the items inside "Handling Breaking Changes" may also apply to your plugin.

`gatsby` should be included under `peerDependencies` of your plugin and it should specify the proper versions of support.

```
{
  "peerDependencies": {
-   "gatsby": "^3.0.0",
+   "gatsby": "^4.0.0",
  }
}
```

If your plugin supports both versions:

```
{
  "peerDependencies": {
-   "gatsby": "^2.32.0",
+   "gatsby": "^3.0.0 || ^4.0.0",
  }
}
```

If you defined the `engines` key you'll also need to update the minimum version:

```
{
  "engines": {
    "node": ">=14.15.0"
  }
}
```

You can also learn more about this in the migration guide for source plugins.

Don't mutate nodes outside of expected APIs

Before v4 you could do something like this, and it was working:

```
exports.sourceNodes = ({ actions }) => {
  const node = {
    /* */
  }
  actions.createNode(node)

  // somewhere else:
  node.image___NODE = `uuid-of-some-other-node`
}
```

This was never an intended feature of Gatsby and is considered an anti-pattern (see #19876 for additional information).

Starting with v4 Gatsby introduces a persisted storage for nodes and thus this pattern will no longer work because nodes are persisted after `createNode` call and all direct mutations after that will be lost.

Gatsby provides diagnostic mode to detect those direct mutations, unfortunately it has noticeable performance overhead so we don't enable it by default. See Debugging missing data for more details on it.

Gatsby provides several actions available in `sourceNodes` and `onCreateNode` APIs to use instead:

- createNode
- deleteNode
- createNodeField

You can use `createNodeField` and the `@link` directive to create the same schema shape. The `@link` directive accepts a `from` argument that you can use to place your node to the old position (as `createNodeField` places everything under a `fields` key). See the source plugin guide for more information. Checkout this PR for a real-world migration example.

`___NODE` convention

Please note that the deprecation of the `___NODE` convention especially affects source plugins and for Gatsby v5 you'll need to update your usage to keep

compatibility.

No support for circular references in data

The current state persistence mechanism supported circular references in nodes. With Gatsby 4 and LMDB this is no longer supported.

This is just a theoretical problem that might arise in v4. Most source plugins already avoid circular dependencies in data.

Bundling external files

In order for DSG & SSR to work Gatsby creates bundles with all the contents of the site, plugins, and data. When a plugin (or your own `gatsby-node.js`) requires an external file via `fs` module (e.g. `fs.readFile`) the engine won't be able to include the file. As a result you might see an error (when trying to run DSG) like `ENOENT: no such file or directory` in the CLI.

This limitation applies to these lifecycle APIs: `setFieldsOnGraphQLNodeType`, `createSchemaCustomization`, and `createResolvers`.

Instead you should move the contents to a JS/TS file and import the file as this way the bundler will be able to include the contents.

Known Issues

This section is a work in progress and will be expanded when necessary. It's a list of known issues you might run into while upgrading Gatsby to v4 and how to solve them.

If you encounter any problem, please let us know in this GitHub discussion.