Debugging and Troubleshooting

Node sourcing

Missing data in Gatsby

If some fields are returning null or an empty array when you think they should be returning data, a few things could be at play. To debug follow these steps:

1. Make the same query in Gatsby and directly to WPGraphQL using a GraphQL client app like GraphQL Playground. Do not use wp-graphiql to debug because it is authenticated and Gatsby will not be authenticated when sourcing data. Use a 3rd party GraphQL client outside of WordPress.

For example if pages weren't returning titles for some reason and the following was your query in Gatsby:

```
{
  allWpPage {
    nodes {
      id
      title
    }
  }
}
```

Take that query and make the query directly to your WP instance GraphQL API like so:

```
{
  pages {
    nodes {
      id
      title
    }
  }
}
```

2. Does the returned data look the same in GraphQL playground as it did in Gatsby? If it does but data should exist there is likely a bug in WPGraphQL or in a WPGraphQL extension you're using. Be sure to double check

that these fields should actually be returning data and that that data isn't missing when you go to edit your post in WordPress.

3. If the returned data does not look the same, enable this plugin option in Gatsby:

```
{
  resolve: `gatsby-source-wordpress`,
  options: {
    debug: {
      graphql: {
        writeQueriesToDisk: true,
      },
    },
  },
},
```

4. Run `gatsby develop` or `gatsby build` again.

5. In your Gatsby project you will now have a directory at `your-app/WordPress/GraphQL/` . Within this directory you will find a directory for each node type that is sourced from WPGraphQL for your Gatsby site.

6. Find the directory of the type that returns missing data. For our example we would look in `WordPress/GraphQL/Page/`

7. Within this directory there will be 3 `.graphql` files. Open the file named `node-list-query.graphql` and copy the query contained within it.

8. Paste this query into GraphQL Playground and add these query variables:

```
{
  "first": 100,
  "after": null
}
```

9. Execute the query

10. Does the response also have missing data?

    If it does have missing data:

    this means you're experiencing a bug on the WPGraphQL side of things. Seek help in the WPGraphQL Slack or open an issue in the WPGraphQL GitHub repo, or the GitHub repo for the WPGraphQL extension that manages the fields you're having trouble with. To help them debug you should narrow down exactly which combination of fields in the generated query you copied is causing issues. Comment out fields 1 by 1 until the problem goes away to determine which combination of fields isn't working.

    Note: A common cause of this problem is that you're using ACF and you've named multiple fields with the same name but in different field groups. Identify conflicting field names and rename them.

    If it does not have missing data:

    This means it's a `gatsby-source-wordpress` bug. Open an issue in the GitHub repo.

Node Sourcing GraphQL errors

If your build is erroring on a GraphQL error returned in the response from a GraphQL request, the terminal output will tell you which type this is. Returned GraphQL errors almost always indicate a bug on the WP side of things. To replicate the error outside of gatsby, enable the `debug.graphql.writeQueriesToDisk` plugin option:

```
{
  resolve: `gatsby-source-wordpress`,
  options: {
    debug: {
      graphql: {
        writeQueriesToDisk: true,
      },
    },
  },
},
```

Now run `gatsby develop` or `gatsby build` again. When the process exits on your GraphQL error, take note of which GraphQL type the error was for, then open this file: `your-site/WordPress/GraphQL/[TypeOfErroringQuery]/node-list-query.graphql`.

Within this file will be the query made during node sourcing to fetch data from WPGraphQL into Gatsby.

You can use this query to reproduce your error message and debug your error message outside of Gatsby. If you're stuck seek help in the WPGraphQL Slack or open an issue in the WPGraphQL GitHub repo, or the GitHub repo for the WPGraphQL extension that manages the fields you're having trouble with.

WordPress 50* errors

If you're running into these errors during node sourcing, the plugin may be trying to fetch more data from your WordPress instance than your WP server can handle. Try lowering the `schema.perPage` plugin option from it's default of 100 and re-run your build process.

You can also try changing the `schema.requestConcurrency` option to limit the amount of GraphQL requests concurrent GraphQL requests made at any time when sourcing data from WPGraphQL. It's default is 15 concurrent requests.

Another reason this can happen is that one of your GraphQL queries causes an unrecoverable error on your WordPress server. See the section on debugging GraphQL errors for debugging steps.

Node Sourcing Timeouts

If you're running into timeouts during node sourcing, it is likely that your WordPress server is limiting the number of concurrent requests to a number lower than the number of different types that this source plugin is attempting to source from your WordPress server. You can either increase the number of concurrent connections your WordPress server can handle by tuning or replacing your server, or you can lower the plugin option `schema.requestConcurrency` to a lower number than the number of concurrent GraphQL requests your WordPress server can handle.

To determine how many concurrent GraphQL requests your server can handle, enable verbose mode with the `verbose` plugin option:

```
{
  resolve: `gatsby-source-wordpress`,
  options: {
    verbose: true,
  },
},
```

Then run your build again. You will see a long list of types that the source plugin is fetching from WordPress such as Page, Post, User, etc.

Notice that each of these types is reporting on the right-side how many nodes is sourced from each.

You will see that at the bottom of the list there are some types which aren't yet returning nodes.

Count upwards from the last type that seems to be frozen to the top of the list. This is the number of concurrent GraphQL requests your server can handle.

Note that `GATSBY_CONCURRENCT_DOWNLOAD` has been retired, now `schema.requestConcurrency` plugin option is used.

Media File Download Errors

The main error that occurs while fetching media files is overwhelming the remote server due to too many concurrent requests. You can set the `schema.requestConcurrency` plugin option below it's default of `100`. You will need to experiment a bit to determine what the maximum number of concurrent requests for media files your server can handle is.

Media File Download Skipped

This might happen for several reasons, but two of the most common are that the file was excluded due to the file's size exceeding the `maxFileSizeBytes` config option or because its mime type was included in the `excludeByMimeTypes` config option.

In order to determine what media items were not downloaded on account of `maxFileSizeBytes`, start your gatsby develop server and run the following GraphQL query.

> ⚠️ *Ensure that you replace the number in the `gt` filter with the value of `maxFileSizeBytes` found in your gatsby config*

```
query TOO_LARGE_FILES {
  allWpMediaItem(filter: { fileSize: { gt: 15728640 } }) {
    nodes {
      id
      sourceUrl
      fileSize
    }
  }
}
```

If you want to investigate which images weren't downloaded due to the `excludeByMimeTypes` option, start up a gatsby develop server and run the following.

> ⚠️ *Ensure that the array of mime types passed to the `in` filter in the following query matches the value of `excludeByMimeTypes` in your gatsby config*

```
query MIME_TYPE_EXCLUDED {
  allWpMediaItem(filter: { mimeType: { in: ["image/jpeg", "video/mp4"] } }) {
    nodes {
      id
      sourceUrl
      mimeType
    }
  }
}
```

Broken Preview templates

Since a Previewed post might have a lot less data attached to it than what you're testing with during development, you might get errors in previews when that data is missing. You can debug your previews by running Gatsby in preview mode locally.

- Run Gatsby in refresh mode with `ENABLE_GATSBY_REFRESH_ENDPOINT=true gatsby develop`
- Install ngrok with `npm i -g ngrok`
- In a new terminal window run `ngrok http 8000`
- In your WP instances GatsbyJS settings, set your Preview instance URL to `https://your-ngrok-url.ngrok.io` and your preview refresh endpoint `https://your-ngrok-url.ngrok.io/__refresh`

Now when you click preview in `wp-admin` it will use your local instance of Gatsby. You can inspect the preview template to see which Gatsby path is being loaded in the preview iframe and open it directly to do further debugging.

You can proactively protect against broken Preview templates by using optional chaining to prevent trying to access properties on `undefined`.

Previews Don't Update

Check in your `wp-config.php` to determine wether or not post revisions are disabled. Look for `define( 'WP_POST_REVISIONS', FALSE );` and if you find it, remove it. This appears to be a WPGraphQL bug and we're working on fixing it. If you're interested in the status of that check https://github.com/wp-graphql/wp-graphql/issues/1408. If post revisions are enabled on your site and previews are still not working, please open a new issue.

Preview debug mode

You can enable Preview debug mode to help you debug issues with the Gatsby Preview build process. Visit the plugin options page to see how to enable this debug option or enable this option by setting the env variable `WP_GATSBY_PREVIEW_DEBUG`. This option will log addition info to the terminal output including the contents of the webhook body that was sent to the Gatsby process, a list of the Preview change events Gatsby receives, as well as the node Preview data that was sourced during the Preview.

Up Next :point_right:

- :national_park: Community and Support
- :point_left: Back to README.md