Preview

How to set it up and use it

- If you haven't already, you'll need
  - A Gatsby site running on Gatsby Cloud, with CMS Previews enabled
  - A publicly available WordPress instance with the following plugins installed:
    - `WPGraphQL >= v0.15.0`
    - `WPGatsby >= v0.6.0`
- Your Gatsby site should be configured to source data from your WordPress instance using `gatsby-source-wordpress`. This means pointing the `url` plugin option for that source plugin at your WordPress instance's `/graphql` endpoint.
- Visit the "CMS Previews" tab in Gatsby Cloud.
- Once your first Preview build has completed with all the necessary plugin versions mentioned above, copy the front-end url of your Preview instance from above the top of the list of completed preview builds.
- Go to your WPGatsby settings and paste the Preview frontend url there.
- Go back to Gatsby Cloud and visit the "Site Settings" tab. Scroll down to the webhook section and copy the Preview webhook.
- Go back to your WPGatsby settings and paste the Preview webhook url there.
- Go to a post or page you want to preview and click "preview" like you normally would in WordPress.
- If anything is misconfigured you will see an error with steps on how to proceed, otherwise you'll see a loading screen which will unveil your preview page once it's ready.

Caveats

- Gutenberg and ACF do not work together for WP Previews. Gutenberg breaks ACF preview (this is not a Gatsby or WPGatsby problem), so if you want to preview ACF, you cannot use Gutenberg.
- You must add a node id to pageContext when creating pages if you want to be able to preview that page. If you don't do this, you'll see a misconfiguration error in the preview window.

How it works internally

- The user presses "preview" from the WordPress admin screen
- There are some checks in place to determine wether or not this is a new preview or if it's a draft, regular post update or duplicate. If it is new a webhook should be sent to the Preview instance.
- Preview "save_post" calls are debounced (per-post) so that many webhooks within a 5 second interval for the same post will only send a single webhook. Different posts that are previewed at the same time will still trigger multiple preview builds (for now) but the preview loading logic will still work great in this case.
- The Preview webhook is POST'ed to with the following information:
  - A JWT token which Gatsby can use to query private preview revision data.
  - The parent database id of the revision
  - Wether or not the preview is for a new post draft (or a regular draft)
  - The type of the node being previewed
  - The id of the revision (or draft) being previewed
  - The URL of the WordPress instance which is sending the Preview
  - Wether or not revisions are disabled
  - The modified time of the node being previewed
  - `preview: true` (the source plugin uses this to tell wether or not previews are being sourced)

- WPGatsby records wether or not the preview webhook was a success (to be used in optimistically loading the preview frontend)
- If the webhook is not online, WPGatsby will record an error in the WP debug log.
- On the Gatsby side, `sourceNodes` as a refresh is invoked via the refresh webhook and the source plugin detects this and invokes `sourcePreviews` instead of sourcing nodes.
- If the Preview was sent from a WP instance other than the one which the source plugin is configured to use, a preview status of `RECEIVED_PREVIEW_DATA_FROM_WRONG_URL` is sent back to WPGatsby for processing.
- The source plugin then stores (in memory) a callback which when invoked will send the preview status back to WPGatsby.
- Separately we have 2 functions that are invoked in `onCreatePage`,
    - `onCreatepageSavePreviewNodeIdToPageDependency` which (in preview mode) stores up a map of node id's to pages created from them. In order to make this performant enough, WPGatsby Preview has a requirement that there's a node id in the pageContext of any page created in Gatsby that should be previewable. The node id in pageContext is used to create the nodeId → page map.
    - `onCreatePageRespondToPreviewStatusQuery` This function checks if the currently created page's node has a preview status callback assigned to it. If it does it invokes the callback with a `PREVIEW_SUCCESS` status type, sending the preview status back to WPGatsby. After invoking it, the callback is removed from the internal store so that it can't be called again.
- In addition, there are a couple more places these preview status callbacks can be invoked.
    - During `onPreExtractQueries` we check for any leftover callbacks which haven't been invoked earlier during `onCreatePage`. Any that haven't been invoked are not previewable. This is either because the created page didn't have a node id in pageContext, or because no page was created for the node being previewed. These leftover callbacks are invoked with a `NO_PAGE_CREATED_FOR_PREVIEWED_NODE` status and WPGatsby processes this status and displays steps on how to debug & fix.
    - In the error boundary of `runSteps` any existing callbacks are invoked with a `GATSBY_PREVIEW_PROCESS_ERROR` status. A very generic error about which step the detailed error occurred in is passed along to WPGatsby on the `context` property of the callback. WPGatsby displays this generic error and encourages the user to check their preview logs.
- The preview logic sources the WPGraphQL node being previewed using the `asPreview` api from WPGraphQL and updates the Gatsby node. If a page is created as a result of this, the above `onCreatePage` logic will run which will in turn update the preview status in WPGatsby.
- While all of this is happening, WP has already opened the preview template. If the webhook that the current preview was posted to returned a `204` or `200` status, we optimistically try to load the preview ui and so display a Gatsby branded loading indicator. If the webhook returned another status, we show an error about the Gatsby Preview instance being offline. In both cases, we do a second in-browser check to see if the Preview instance really is online or offline. The reason for this is the webhook will not always be hit when the preview window is loaded or re-loaded. So the Preview instance could come online or offline in the meantime.
- If it takes longer than 45 seconds for the preview to be loaded, a loader warning will appear below the loading animation saying something may be wrong. There is also a "cancel and troubleshoot" button below the warning which when pressed will cancel the preview client from waiting for a response, and show debugging steps.
- In the case of WPGatsby misconfiguration (No preview frontend url/webhook set, or the current post type is not set to show in graphql) a different preview template will be loaded which displays an error with steps on how to remedy.

- If the preview client receives any error status (any status besides `PREVIEW_SUCCESS`) the error will be displayed and the loading icon animation will be removed.
- If the `PREVIEW_SUCCESS` status is received (along with the path the preview can be viewed at), the iframe will be updated to point to the preview frontend url + path. Once the iframe emits it's loaded event, the loader will be removed, unveiling the preview via the Gatsby Preview site.
- In addition to the above, to make the preview logic fast, Diffing the local and remote schemas before pulling preview data has been removed. Instead, we catch any GraphQL errors when updating previews, then re-run schema diffing. If the schemas are different, we regenerate our node sourcing queries and re-fetch the preview data. This means removing a field in WPGraphQL will not break previews unless the preview was specifically querying for that field.
- Upon realizing the above, it was trivial to enable similar functionality for all `gatsby develop`. Now whenever an action is received from WPGatsby, the source plugin will diff the schemas. If they're different, it will re-run `createSchemaCustomization` which will fetch the updated schema and update the Gatsby queries. This means you don't need to re-start Preview or `gatsby develop` when updating your remote schema!