Using Deferred Static Generation (DSG)

Introduction

Deferred Static Generation (DSG) is one of Gatsby's rendering options and allows you to defer non-critical page generation to user request, speeding up build times. Instead of generating *every* page at build time, you can decide to build certain pages up front and others only when a user accesses the page at run time. For large sites, with content that is infrequently visited (e.g. old blog posts or certain content types), this can dramatically reduce build times.

In this guide, you'll learn how to modify your `createPage` calls to only build your preferred pages up-front and leave the rest deferred to the first user request. You'll also learn how to use DSG with File System Route API.

For full documentation on all options, see the Reference Guide on Deferred Static Generation.

Prerequisites

Before you begin, you should already have:

- An existing Gatsby site. (Need help creating one? Follow the Quick Start.)
- Either a `gatsby-node.js` file where you're creating pages with the `createPages` API or a page template that uses the File System Route API

Directions

File System Route API

The general process for using DSG looks like this:

- Exporting an async function called `config` that returns an object with the key `defer`

```
// The rest of your page, including imports, page component & page query etc.

export async function config() {
  // Optionally use GraphQL here
```

```
      return ({ params }) => {
        return {
          defer: true,
        }
      }
    }
```

For the purpose of this guide, let's assume you have a blog powered by markdown (using `gatsby-transformer-remark`) and have a `date` & `slug` key inside the `frontmatter` of each file. You want to defer all posts that are older than 2021-10-31.

A markdown file could look something like this:

```
---
slug: my-first-post
date: 2021-10-30
---

# My First Post

Some text about something cool.
```

Your page component would look something like this:

```
import * as React from "react"
import { graphql } from "gatsby"

export default function Component(props) {
  return <pre>{JSON.stringify(props, null, 2)}</pre>
}

export const query = graphql`
  query ($id: String) {
    markdownRemark(id: { eq: $id }) {
      html
      frontmatter {
        slug
        date
      }
    }
  }
`
```

You're creating paths like `/my-first-post` as the `frontmatter__slug` is used. Then the component displays the result of the page query.

Now you need to add the async `config` function:

```
export async function config() {
```

```
  const { data } = graphql`
    {
      oldPosts: allMarkdownRemark(
        filter: { frontmatter: { date: { lt: "2021-10-31" } } }
      ) {
        nodes {
          frontmatter {
            slug
          }
        }
      }
    }
  `

  const oldPosts = new Set(data.oldPosts.nodes.map(n => n.frontmatter.slug))

  return ({ params }) => {
    return {
      defer: oldPosts.has(params.frontmatter__slug)
    }
  }
}
```

Inside the `graphql` call (which you can only use in the outer function) you can query any data you like. In this instance all posts older than 2021-10-31 are queried with their `slug`. The inner function has access to `params` that get passed through with File System Route API.

So at the end you compare the current slug (`params.frontmatter__slug`) with the set of all posts that are older than 2021-10-31. If the current slug (and thus the current post) is inside this set, the page is marked as deferred.

`gatsby-node.js`

The general process for using DSG looks like this:

- Adding `defer: true` to your `createPage` call.

  ```
  createPage({
    path: "page-path",
    component: "component-path",
    context: {},
    defer: true, // highlight-line
  })
  ```

For the purpose of this guide, let's assume you have a blog powered by MDX and have blog posts dating back years (with 1000 blog posts in total). Via your analytics tracking, you're seeing that only the latest 100 posts are regularly

visited. The rest gets occasional or no visits at all.

The `gatsby-node.js` file:

```
const blogPostTemplate = require.resolve(`./src/templates/blog-post.js`)

export.createPages = async ({ graphql, actions, reporter }) => {
  const { createPage } = actions

  const result = await graphql(`
    query {
      allMdx(sort: { fields: frontmatter___date, order: DESC }) {
        nodes {
          slug
        }
      }
    }
  `)

  if (result.errors) {
    reporter.panicOnBuild(`There was an error loading posts`, result.errors)
    return
  }

  const posts = result.data.allMdx.nodes

  posts.forEach(post => {
    createPage({
      path: post.slug,
      component: blogPostTemplate,
      context: {
        slug: post.slug,
      },
    })
  })
}
```

For this example, it's important that inside the `gatsby-node.js` the result is sorted by date and in an descending order. Depending on your use case you might need to adjust the query to sort differently or query additional information (e.g. a "category" field from the frontmatter) besides the `slug`.

This way you can use the `index` in the `forEach` to apply `defer` to all but the latest 100 posts:

```
// Rest of gatsby-node.js

const posts = result.data.allMdx.nodes
```

```
posts.forEach((post, index) => {
  createPage({
    path: post.slug,
    component: blogPostTemplate,
    context: {
      slug: post.slug,
    },
    // index is zero-based index
    defer: index + 1 > 100, // highlight-line
  })
})
```

The first 100 pages will receive `defer: false`, the other 900 pages receive `defer: true`.

Additional Resources

- API Reference guide
- Conceptual Guide