AI VOID FIELD GUIDE

Step-by-Step Tutorials

Focused, practical tutorials that teach you how to accomplish specific tasks step by step. Learn by doing with clear instructions and working code examples.

AI VOID

2026-05-04 20:31 IST

Contents

CHAPTER 01

Ruby S3 Directory Transfers with Transfer Manager

What you'll build: A Ruby script to efficiently upload and download local directories to and from Amazon S3 using the Transfer Manager's new directory support. **Time needed:** ~25 minutes **Prerequisites:** Ruby installed (version 2.7+ recommended), AWS Account with S3 permissions, AWS CLI configured with credentials (or environment variables), An existing Amazon S3 bucket, `aws-sdk-s3` gem installed (version 1.215+) **Version used:** 1.215+

Introduction to S3 Transfer Manager Directory Support

Working with Amazon S3 is a cornerstone of many cloud applications, especially for storing and retrieving large amounts of data. While the AWS SDK for Ruby has always provided robust ways to interact with S3, handling many files or entire directories efficiently could sometimes become complex. You'd often need to write custom logic for iterating through files, managing concurrent uploads/downloads, and ensuring multipart transfers for large objects.

This is where the AWS SDK for Ruby's `TransferManager` comes in. It's a high-level utility designed to simplify these common S3 transfer operations. With the release of `aws-sdk-s3` version 1.215+, the Transfer Manager gained powerful new capabilities: direct support for uploading and downloading entire directories.

This new feature streamlines bulk data transfers by automatically handling:

- **Multipart uploads/downloads:** For files larger than a certain threshold, it automatically splits them into parts, transfers them concurrently, and reassembles them on the destination.
- **Parallelism:** It can transfer multiple files simultaneously, significantly speeding up operations on directories with many small files.
- **Progress tracking:** Provides callbacks to monitor the real-time status of your transfers.
- **Error handling:** Offers a more resilient way to manage transfer failures.

This tutorial will guide you through using this powerful new directory support, allowing you to focus on your application logic rather than the intricacies of S3 transfer mechanics.

📌 **Key Idea:** The S3 Transfer Manager's directory support (v1.215+) simplifies and accelerates bulk data transfers to and from S3 by automating complex tasks like multipart uploads and parallel processing.

Prerequisites and Setup

Before we dive into the code, let's ensure your environment is ready. Having these prerequisites in place will make your journey smooth and error-free.

1. **Ruby Installation:** You should have Ruby installed on your system. Version 2.7 or newer is recommended for the latest features and compatibility. You can check your version by running: `bash ruby -v`
2. **AWS Account and S3 Permissions:** You'll need an active AWS account. Ensure the AWS user or role whose credentials you'll be using has the necessary permissions to perform `s3:PutObject`, `s3:GetObject`, `s3:ListBucket`, and `s3:DeleteObject` (if you plan to clean up) actions on your target S3 bucket.
3. **AWS CLI Configuration:** Your AWS credentials need to be configured. The AWS SDK for Ruby will automatically pick them up from the AWS CLI configuration file (`~/.aws/credentials`) or environment variables (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION`). If you haven't set this up, you can do so with: `bash aws configure` Follow the prompts to enter your Access Key ID, Secret Access Key, default region, and output format.
4. **Existing Amazon S3 Bucket:** You'll need an S3 bucket to upload and download directories. Make a note of its name. If you don't have one, you can create it via the AWS Management Console or the AWS CLI: `bash aws s3 mb s3://my-unique-ruby-transfer-bucket-12345` Remember, S3 bucket names must be globally unique.
5. **`aws-sdk-s3` Gem Installation:** This is crucial. The directory transfer feature was introduced in `aws-sdk-s3` version 1.215+. You must have this version or newer installed. First, let's create a `Gemfile` for our project to manage dependencies: ```ruby # Gemfile source 'https://rubygems.org'

   gem 'aws-sdk-s3', '~> 1.215' `Now, install the gem using Bundler:` bash bundle install ```

   ⚡ **Note:** Using `~> 1.215` ensures you get version 1.215 or any compatible newer patch version (e.g., 1.216, but not 2.0). If you already have `aws-sdk-s3` installed, ensure it's updated to at least 1.215.

Once these steps are complete, your environment is ready for action!

Preparing Your Local Directory for Transfer

To demonstrate directory transfers effectively, we need a local directory with some nested files that we can then upload to S3. Let's create a simple, representative directory structure.

First, create a new directory for your Ruby script and navigate into it:

```
mkdir ruby-s3-transfer-demo
cd ruby-s3-transfer-demo
```

Now, inside `ruby-s3-transfer-demo`, let's create a sample directory called `my_local_data` with a few files and subdirectories. This will simulate a typical project or data folder.

```
mkdir -p my_local_data/images
mkdir -p my_local_data/documents
mkdir -p my_local_data/misc

echo "<h1>Welcome to my site!</h1>" > my_local_data/index.html
echo "This is a sample report." > my_local_data/documents/report.txt
echo "Confidential data here." > my_local_data/documents/sensitive.csv
echo "A beautiful cat picture." > my_local_data/images/cat.jpg
echo "Another image file." > my_local_data/images/logo.png
echo "A temporary file." > my_local_data/misc/temp.log
```

After running these commands, your `my_local_data` directory structure should look something like this:

```
my_local_data/
├── documents/
│   ├── report.txt
│   └── sensitive.csv
├── images/
│   ├── cat.jpg
│   └── logo.png
├── misc/
│   └── temp.log
└── index.html
```

This structure provides a good mix of files at different levels, which is perfect for testing the Transfer Manager's directory capabilities.

⚡ **Real-world insight:** When you upload a local directory like `my_local_data` to S3, the S3 objects will typically have keys that preserve this hierarchy. For

example, `my_local_data/images/cat.jpg` will become an S3 object with a key like `your_prefix/images/cat.jpg`. S3 itself doesn't have "directories" in the traditional filesystem sense; it uses object keys with `/` as a delimiter to simulate them.

You've successfully prepared a local directory for our S3 transfer examples.

Uploading a Directory to Amazon S3

Now that we have our local data ready, let's write our first Ruby script to upload the entire `my_local_data` directory to your S3 bucket using the Transfer Manager.

Create a new file named `upload_directory.rb` in your `ruby-s3-transfer-demo` directory.

```ruby
# upload_directory.rb
require 'aws-sdk-s3'
require 'pathname' # For robust path handling

# --- Configuration ---
# Replace with your S3 bucket name
S3_BUCKET_NAME = 'my-unique-ruby-transfer-bucket-12345'
# The local directory to upload
LOCAL_DIRECTORY_PATH = 'my_local_data'
# The S3 prefix (folder) where the directory content will be placed
# An empty string means upload to the bucket root.
# A value like 'my-uploads/' will place it inside a folder named 'my-uploads'
S3_UPLOAD_PREFIX = 'my-uploaded-data/'

# --- Script Logic ---
puts "Initializing S3 client and Transfer Manager..."
s3_client = Aws::S3::Client.new(region: 'us-east-1') # Ensure your region is
correct
transfer_manager = Aws::S3::TransferManager.new(client: s3_client)

begin
  source_path = Pathname.new(LOCAL_DIRECTORY_PATH)
  unless source_path.directory?
    raise "Error: Local directory
'#{LOCAL_DIRECTORY_PATH}' not found or is not a directory."
  end

  puts "Starting upload of directory '#{LOCAL_DIRECTORY_PATH}' to s3://#{S3_BUC
KET_NAME}/#{S3_UPLOAD_PREFIX}"

  # The core upload call
  transfer_manager.upload_directory(
    source: LOCAL_DIRECTORY_PATH,
    bucket: S3_BUCKET_NAME,
    prefix: S3_UPLOAD_PREFIX
  )

  puts "Directory upload complete!"

rescue Aws::S3::Errors::NoSuchBucket => e
  puts "⚠️ Error: The specified bucket '#{S3_BUCKET_NAME}' does not exist.
Please check the bucket name."
  puts "Details: #{e.message}"
rescue Aws::S3::Errors::AccessDenied => e
  puts "⚠️ Error: Access denied to S3 bucket. Check your AWS credentials and
IAM permissions."
  puts "Details: #{e.message}"
rescue StandardError => e
  puts "An unexpected error occurred: #{e.message}"
end

puts "Closing Transfer Manager..."
transfer_manager.shutdown
```

Let's break down this script:

1. **`require 'aws-sdk-s3'` and `require 'pathname'`**: We load the necessary AWS SDK gem and `Pathname` for robust file path handling.

2. **Configuration**: We define `S3_BUCKET_NAME`, `LOCAL_DIRECTORY_PATH`, and `S3_UPLOAD_PREFIX`. Remember to replace `my-unique-ruby-transfer-bucket-12345` with your actual S3 bucket name. The `S3_UPLOAD_PREFIX` is like a folder within your S3 bucket; if you leave it as an empty string `''`, files will be uploaded directly to the bucket's root.
3. **Client Initialization**: We create an `Aws::S3::Client` instance, specifying the region. This client is then used to initialize the `Aws::S3::TransferManager`.
4. **Directory Check**: We use `Pathname.new(LOCAL_DIRECTORY_PATH).directory?` to ensure the source path actually points to an existing directory before attempting the upload.
5. **`transfer_manager.upload_directory`**: This is the core method.
    - `source`: The path to your local directory.
    - `bucket`: The name of your S3 bucket.
    - `prefix`: An optional S3 key prefix. All files from your local directory will be uploaded under this prefix. For example, `my-uploaded-data/` will result in objects like `my-uploaded-data/documents/report.txt`.
6. **Error Handling**: A `begin...rescue` block is included to catch common S3 errors like `NoSuchBucket` (if your bucket name is wrong) or `AccessDenied` (if your AWS credentials lack permissions).
7. **`transfer_manager.shutdown`**: It's important to call `shutdown` on the Transfer Manager when you're done, especially if you're using it in a long-running application, to release resources.

Now, let's run the script:

```
bundle exec ruby upload_directory.rb
```

You should see output similar to this (with your bucket name):

```
Initializing S3 client and Transfer Manager...
Starting upload of directory 'my_local_data' to s3://my-unique-ruby-transfer-bucket-12345/my-uploaded-data/
Directory upload complete!
Closing Transfer Manager...
```

To verify the upload, log into your AWS Management Console, navigate to your S3 bucket, and look for the `my-uploaded-data/` prefix. You should see all the files and their directory structure preserved.

⚠️ **Common mistake:** Forgetting to update `S3_BUCKET_NAME` to your actual bucket name. This will lead to a `NoSuchBucket` error. Also, ensure your AWS credentials have `s3:PutObject` and `s3:ListBucket` permissions for the target bucket.

You've successfully uploaded a local directory to Amazon S3 using the Transfer Manager!

Customizing Uploads: Parallelism and Filtering

The `upload_directory` method offers powerful customization options to fine-tune how your data is transferred. You can control the level of parallelism and apply filters to include or exclude specific files.

Let's modify our `upload_directory.rb` script to demonstrate these features. We'll add options to upload only image files and limit the concurrent uploads.

```ruby
# upload_directory_custom.rb (New file or modify previous)
require 'aws-sdk-s3'
require 'pathname'

# --- Configuration ---
S3_BUCKET_NAME = 'my-unique-ruby-transfer-bucket-12345'
LOCAL_DIRECTORY_PATH = 'my_local_data'
S3_UPLOAD_PREFIX = 'my-custom-uploaded-data/' # Use a new prefix for this
example

# --- Script Logic ---
puts "Initializing S3 client and Transfer Manager..."
s3_client = Aws::S3::Client.new(region: 'us-east-1')
transfer_manager = Aws::S3::TransferManager.new(client: s3_client)

begin
  source_path = Pathname.new(LOCAL_DIRECTORY_PATH)
  unless source_path.directory?
    raise "Error: Local directory
'#{LOCAL_DIRECTORY_PATH}' not found or is not a directory."
  end

  puts "Starting CUSTOM upload of directory '#{LOCAL_DIRECTORY_PATH}' to s3://#
{S3_BUCKET_NAME}/#{S3_UPLOAD_PREFIX}"
  puts "  - Only uploading image files (*.jpg, *.png)"
  puts "  - Using a maximum of 2 concurrent uploads."

  # The core upload call with custom options
  transfer_manager.upload_directory(
    source: LOCAL_DIRECTORY_PATH,
    bucket: S3_BUCKET_NAME,
    prefix: S3_UPLOAD_PREFIX,
    max_concurrent_puts: 2, # Limit concurrent uploads to 2
    include: [          # Only include files matching these patterns
      '**/*.jpg',
      '**/*.png'
    ],
    exclude: [          # Exclude files matching these patterns (applied after
include)
      '**/temp.log'     # Example: exclude specific log files
    ]
  )

  puts "Custom directory upload complete!"

rescue Aws::S3::Errors::NoSuchBucket => e
  puts "⚠️ Error: The specified bucket '#{S3_BUCKET_NAME}' does not exist.
Please check the bucket name."
  puts "Details: #{e.message}"
rescue Aws::S3::Errors::AccessDenied => e
  puts "⚠️ Error: Access denied to S3 bucket. Check your AWS credentials and
IAM permissions."
  puts "Details: #{e.message}"
rescue StandardError => e
  puts "An unexpected error occurred: #{e.message}"
end

puts "Closing Transfer Manager..."
transfer_manager.shutdown
```

Here's what we've added and changed:

1. `max_concurrent_puts`: This option controls how many files the Transfer Manager will attempt to upload simultaneously. By default, it uses a sensible number based on your system, but you can explicitly set it. Lowering it can reduce system resource usage, while increasing it can speed up transfers over high-bandwidth connections, up to a point.

- **Value:** An integer representing the maximum number of concurrent `PutObject` operations. 2. **`include` and `exclude`**: These options allow you to filter which files are included or excluded from the transfer. They accept an array of glob-style patterns (e.g., `*.txt`, `images/*.jpg`, `**/*.log`).
- **`include`**: Only files matching any of these patterns will be considered for upload.
- **`exclude`**: Files matching any of these patterns will be excluded, even if they were previously included by an `include` pattern. `exclude` patterns take precedence over `include` patterns.
    - In our example, `**/*.jpg` and `**/*.png` will upload only the image files, and `**/temp.log` is explicitly excluded (though it wouldn't have been included by the `include` patterns anyway, this demonstrates its usage).

Now, run this modified script:

```
bundle exec ruby upload_directory_custom.rb
```

You'll see the completion message. To verify, check your S3 bucket under the `my-custom-uploaded-data/` prefix. You should only find `cat.jpg` and `logo.png` within `my-custom-uploaded-data/images/`. The `index.html`, `report.txt`, `sensitive.csv`, and `temp.log` files should be absent.

🔥 **Optimization / Pro tip:** The optimal value for `max_concurrent_puts` depends on your network bandwidth, the number and size of files, and your local machine's resources. For many small files, increasing concurrency can help. For a few very large files, the multipart upload mechanism already handles concurrency within a single file transfer, so `max_concurrent_puts` might have less impact. Experiment to find the sweet spot for your specific use case.

You've learned how to customize your directory uploads using parallelism and file filtering.

Downloading a Directory from Amazon S3

Just as easily as you can upload a directory, the Transfer Manager allows you to download an entire S3 prefix (which acts like a directory) to your local filesystem. Let's create a new script to download the `my-uploaded-data/` prefix we uploaded earlier.

Create a new file named `download_directory.rb`.

```ruby
# download_directory.rb
require 'aws-sdk-s3'
require 'pathname'

# --- Configuration ---
S3_BUCKET_NAME = 'my-unique-ruby-transfer-bucket-12345'
# The S3 prefix (folder) to download from
S3_DOWNLOAD_PREFIX = 'my-uploaded-data/'
# The local directory where the S3 content will be downloaded
LOCAL_DESTINATION_PATH = 'my_downloaded_data'

# --- Script Logic ---
puts "Initializing S3 client and Transfer Manager..."
s3_client = Aws::S3::Client.new(region: 'us-east-1')
transfer_manager = Aws::S3::TransferManager.new(client: s3_client)

begin
  destination_path = Pathname.new(LOCAL_DESTINATION_PATH)
  # Ensure the destination directory exists, Transfer Manager will create it,
  # but it's good practice to be aware or pre-create if needed for specific
permissions.
  # If it exists, files will be downloaded into it. If not, it will be created.

  puts "Starting download of S3 prefix 's3://#{S3_BUCKET_NAME}/#{S3_DOWNLOAD_PR
EFIX}' to local directory '#{LOCAL_DESTINATION_PATH}'"

  # The core download call
  transfer_manager.download_directory(
    bucket: S3_BUCKET_NAME,
    prefix: S3_DOWNLOAD_PREFIX,
    destination: LOCAL_DESTINATION_PATH
  )

  puts "Directory download complete!"

rescue Aws::S3::Errors::NoSuchBucket => e
  puts "⚠️ Error: The specified bucket '#{S3_BUCKET_NAME}' does not exist.
Please check the bucket name."
  puts "Details: #{e.message}"
rescue Aws::S3::Errors::NoSuchKey => e
  puts "⚠️ Error: The specified S3 prefix '#{S3_DOWNLOAD_PREFIX}' does not
exist or contains no objects."
  puts "Details: #{e.message}"
rescue Aws::S3::Errors::AccessDenied => e
  puts "⚠️ Error: Access denied to S3 bucket. Check your AWS credentials and
IAM permissions."
  puts "Details: #{e.message}"
rescue StandardError => e
  puts "An unexpected error occurred: #{e.message}"
end

puts "Closing Transfer Manager..."
transfer_manager.shutdown
```

Let's examine the key parts of this download script:

1. **Configuration**: We define `S3_BUCKET_NAME`, `S3_DOWNLOAD_PREFIX` (which corresponds to the prefix we uploaded), and `LOCAL_DESTINATION_PATH` where the files will be saved.
2. `transfer_manager.download_directory`: This is the method for downloading.
    - `bucket`: The name of your S3 bucket.
    - `prefix`: The S3 key prefix (like a folder) you want to download. All objects with keys starting with this prefix will be downloaded.
    - `destination`: The local path where the downloaded files will be stored. The directory structure under the S3 prefix will be recreated locally.
3. **Error Handling**: Similar to uploads, we include error handling for `NoSuchBucket`, `NoSuchKey` (if the prefix doesn't exist or is empty), and `AccessDenied`.

Before running, ensure your `my_downloaded_data` directory does not exist or is empty, so you can clearly see the new files.

```
rm -rf my_downloaded_data # Be careful with rm -rf! Only run if you're sure.
```

Now, run the script:

```
bundle exec ruby download_directory.rb
```

You should see output indicating the download is complete.

```
Initializing S3 client and Transfer Manager...
Starting download of S3 prefix 's3://my-unique-ruby-transfer-bucket-12345/my-
uploaded-data/' to local directory 'my_downloaded_data'
Directory download complete!
Closing Transfer Manager...
```

To verify, check your local filesystem. You should find a new directory `my_downloaded_data` containing the structure and files you originally uploaded:

```
my_downloaded_data/
├── documents/
│   ├── report.txt
│   └── sensitive.csv
├── images/
│   ├── cat.jpg
│   └── logo.png
└── index.html
```

🧠 **Important:** The `download_directory` method will automatically create the `destination` directory if it doesn't exist. If it does exist, new files will be added, and existing files will **not** be overwritten by default unless you explicitly specify `overwrite: true`.

You've successfully downloaded an S3 directory to your local machine using the Transfer Manager.

Customizing Downloads: Overwriting and Filtering

Similar to uploads, `download_directory` offers options to control how files are handled locally and which files are included in the download. We'll focus on `overwrite` behavior and filtering.

Let's modify `download_directory.rb` (or create a new `download_directory_custom.rb`) to demonstrate these options. We'll download only document files and allow overwriting.

```ruby
# download_directory_custom.rb
require 'aws-sdk-s3'
require 'pathname'

# --- Configuration ---
S3_BUCKET_NAME = 'my-unique-ruby-transfer-bucket-12345'
S3_DOWNLOAD_PREFIX = 'my-uploaded-data/' # Using the same prefix as before
LOCAL_DESTINATION_PATH = 'my_custom_downloaded_data'
# New destination to avoid conflicts

# --- Script Logic ---
puts "Initializing S3 client and Transfer Manager..."
s3_client = Aws::S3::Client.new(region: 'us-east-1')
transfer_manager = Aws::S3::TransferManager.new(client: s3_client)

begin
  destination_path = Pathname.new(LOCAL_DESTINATION_PATH)
  # Ensure the destination directory exists and is clean for this example
  FileUtils.rm_rf(destination_path) if destination_path.exist?
  FileUtils.mkdir_p(destination_path)

  puts "Starting CUSTOM download of S3 prefix 's3://#{S3_BUCKET_NAME}/#{S3_DOWN
LOAD_PREFIX}' to local directory '#{LOCAL_DESTINATION_PATH}'"
  puts "  - Only downloading document files (*.txt, *.csv)"
  puts "  - Overwriting existing local files."

  # The core download call with custom options
  transfer_manager.download_directory(
    bucket: S3_BUCKET_NAME,
    prefix: S3_DOWNLOAD_PREFIX,
    destination: LOCAL_DESTINATION_PATH,
    overwrite: true, # Allow overwriting existing local files
    include: [       # Only include objects matching these patterns
      '**/*.txt',
      '**/*.csv'
    ],
    exclude: [       # Exclude objects matching these patterns (applied after
include)
      '**/temp.log'  # Example: exclude specific log files (if they were in S3)
    ]
  )

  puts "Custom directory download complete!"

rescue Aws::S3::Errors::NoSuchBucket => e
  puts "⚠️ Error: The specified bucket '#{S3_BUCKET_NAME}' does not exist.
Please check the bucket name."
  puts "Details: #{e.message}"
rescue Aws::S3::Errors::NoSuchKey => e
  puts "⚠️ Error: The specified S3 prefix '#{S3_DOWNLOAD_PREFIX}' does not
exist or contains no objects."
  puts "Details: #{e.message}"
rescue Aws::S3::Errors::AccessDenied => e
  puts "⚠️ Error: Access denied to S3 bucket. Check your AWS credentials and
IAM permissions."
  puts "Details: #{e.message}"
rescue StandardError => e
  puts "An unexpected error occurred: #{e.message}"
end
```

```
puts "Closing Transfer Manager..."
transfer_manager.shutdown
```

Here's what these new options do:

1. **`overwrite: true`**: By default, `download_directory` will not overwrite existing files in the `destination` directory. If a file with the same name already exists locally, it will be skipped. Setting `overwrite: true` forces the Transfer Manager to replace any existing local files with the downloaded S3 objects if their keys match.
2. **`include` and `exclude`**: These work identically to the upload filtering options. They take an array of glob-style patterns and apply them to the S3 object keys.
    - In this example, `**/*.txt` and `**/*.csv` will ensure only the document files are downloaded.

Before running, ensure you delete the `my_custom_downloaded_data` directory if it exists, or let the script's `FileUtils.rm_rf` handle it.

```
bundle exec ruby download_directory_custom.rb
```

After execution, check your `my_custom_downloaded_data` directory. You should only find `report.txt` and `sensitive.csv` within `my_custom_downloaded_data/documents/`. The `index.html` and image files should not be present.

⚠️ **What can go wrong:** If `overwrite: false` (the default) and you run a download multiple times, files that already exist locally will not be updated, even if the S3 version is newer. Always be explicit with `overwrite` based on your desired behavior.

You've learned how to control file overwriting and apply filters during directory downloads from S3.

Handling Progress and Potential Errors

For large directory transfers, knowing the progress is essential, and robust error handling is critical for any production-ready script. The Transfer Manager provides mechanisms for both.

Let's enhance our `upload_directory.rb` script to include a progress callback and more detailed error reporting.

```ruby
# upload_directory_with_progress_error.rb
require 'aws-sdk-s3'
require 'pathname'
require 'fileutils' # For creating local directory if needed

# --- Configuration ---
S3_BUCKET_NAME = 'my-unique-ruby-transfer-bucket-12345'
LOCAL_DIRECTORY_PATH = 'my_local_data' # Ensure this directory exists with
content
S3_UPLOAD_PREFIX = 'my-progress-data/' # Use a new prefix

# --- Script Logic ---
puts "Initializing S3 client and Transfer Manager..."
s3_client = Aws::S3::Client.new(region: 'us-east-1')
transfer_manager = Aws::S3::TransferManager.new(client: s3_client)

total_bytes_transferred = 0
total_bytes_to_transfer = 0 # This will be estimated by Transfer Manager

begin
  source_path = Pathname.new(LOCAL_DIRECTORY_PATH)
  unless source_path.directory?
    raise "Error: Local directory
'#{LOCAL_DIRECTORY_PATH}' not found or is not a directory."
  end

  puts "Starting upload of directory '#{LOCAL_DIRECTORY_PATH}' to s3://#{S3_BUC
KET_NAME}/#{S3_UPLOAD_PREFIX}"

  # The core upload call with a progress callback
  transfer_manager.upload_directory(
    source: LOCAL_DIRECTORY_PATH,
    bucket: S3_BUCKET_NAME,
    prefix: S3_UPLOAD_PREFIX,
    on_progress: proc do |bytes_transferred, total_bytes|
      total_bytes_to_transfer = total_bytes # Capture total once available
      progress_percent = (bytes_transferred.to_f / total_bytes * 100).round(2)
      print "\rProgress: #{bytes_transferred} / #{total_bytes} bytes (#{progres
s_percent}%)"
      $stdout.flush # Ensure output is immediately displayed
    end
  )

  puts "\nDirectory upload complete!" # Newline after progress bar

rescue Aws::S3::Errors::ServiceError => e
  # Catch all AWS SDK service errors
  puts "\n⚠️ AWS S3 Service Error during transfer!"
  puts "  Error Code: #{e.code}"
  puts "  Message: #{e.message}"
  puts "  Request ID: #{e.request_id}"
  puts "  HTTP Status: #{e.http_status}"
rescue StandardError => e
  # Catch any other unexpected Ruby errors
  puts "\nAn unexpected Ruby error occurred: #{e.message}"
  puts "Backtrace:\n#{e.backtrace.join("\n")}"
ensure
  # Ensure shutdown is called even if an error occurs
  puts "Closing Transfer Manager..."
```

```
  transfer_manager.shutdown
end
```

Key additions and changes:

1. **`on_progress` callback**: This option takes a `Proc` (or a block) that will be called periodically during the transfer.
    - It receives two arguments: `bytes_transferred` (the amount of data transferred so far for the entire operation) and `total_bytes` (the estimated total size of all files to be transferred).
    - We use `print "\r"` to overwrite the current line in the console, creating a dynamic progress bar effect. `$stdout.flush` ensures the output is immediately written.
2. **Enhanced Error Handling**:
    - `rescue Aws::S3::Errors::ServiceError`: This is a parent class for all AWS SDK service-specific errors. Catching this allows us to capture common S3-related issues like `NoSuchBucket`, `AccessDenied`, `NoSuchKey`, etc., and extract details like `e.code`, `e.message`, `e.request_id`, and `e.http_status`.
    - `rescue StandardError`: Catches any other unexpected Ruby errors that might occur outside of S3 service calls.
    - `ensure`: The `transfer_manager.shutdown` call is moved into an `ensure` block. This guarantees that the Transfer Manager's resources are properly released, regardless of whether the transfer completed successfully or an error occurred.

Now, run this script:

```
bundle exec ruby upload_directory_with_progress_error.rb
```

You should see a live progress update as files are uploaded, followed by the completion message. If you introduce an error (e.g., by changing `S3_BUCKET_NAME` to a non-existent bucket), you'll see the detailed error output.

⚡ **Quick Note:** The `total_bytes` value provided by `on_progress` is an estimate. For very large directories or complex filtering, it might be calculated more accurately as the transfer proceeds.

You've learned how to track the progress of your transfers and implement robust error handling for a more resilient and user-friendly script.

What to Explore Next

You've successfully mastered the basics of using the AWS SDK for Ruby's Transfer Manager for S3 directory transfers. This is a powerful feature that can significantly simplify your data handling workflows. Here are some ideas to further explore and build upon what you've learned:

1. **Integrate into a Background Job System:** For real-world applications, large S3 transfers are often performed asynchronously to avoid blocking user interfaces. Integrate the Transfer Manager into a background job system like Sidekiq or Resque in a Rails application. This would allow users to initiate a directory upload/download, receive immediate feedback, and then be notified when the potentially long-running transfer completes.
2. **Implement More Sophisticated Filtering with Custom Logic:** While `include` and `exclude` patterns are powerful, you might need more complex filtering based on file size, modification date, or even custom metadata. The Transfer Manager allows you to provide a custom `filter` block that takes a `Aws::S3::TransferManager::File` object (for uploads) or a `Aws::S3::Object` object (for downloads) and returns `true` to include or `false` to exclude. This gives you ultimate control over what gets transferred.
3. **Explore Advanced Transfer Manager Options for Single Files:** The `TransferManager` also offers enhanced `upload` and `download` methods for single files, which also benefit from multipart handling and progress tracking. Experiment with these for individual large object transfers, and consider how you might combine single-file transfers with directory transfers in a hybrid strategy for highly specific use cases.

🧠 Check Your Understanding

- Why is `aws-sdk-s3` version 1.215+ a critical prerequisite for using the directory transfer feature?
- What are two benefits of using `TransferManager.upload_directory` compared to manually iterating files and calling `s3_client.put_object` for each?

⚡ Mini Task

Modify one of the upload scripts to include only files with a `.txt` extension and exclude any files found in a `temp/` subdirectory (if you were to create one).

🚀 Scenario

Your team needs to regularly backup a local application's `assets/` directory (containing images, CSS, JS) to an S3 bucket for disaster recovery. This directory can grow to contain thousands of small files. Describe how you would configure the `TransferManager.upload_directory` call, specifically considering parallelism and ensuring that temporary build files (e.g., `.map` files, `.DS_Store`) are not uploaded.

📌 TL;DR

- The AWS SDK for Ruby's `TransferManager` (v1.215+) now directly supports efficient bulk directory uploads and downloads to/from S3.
- Directory transfers automatically handle multipart uploads/downloads, parallel processing, and progress tracking.
- Customization options like `max_concurrent_puts`, `include`, `exclude`, and `overwrite` allow fine-grained control over transfer behavior.

🧠 Core Flow

1. Initialize `Aws::S3::Client` and `Aws::S3::TransferManager`.
2. Use `transfer_manager.upload_directory` with `source`, `bucket`, and `prefix` to send local folders to S3.
3. Use `transfer_manager.download_directory` with `bucket`, `prefix`, and `destination` to retrieve S3 prefixes to local folders.

🚀 Key Takeaway

The S3 Transfer Manager dramatically simplifies complex bulk data operations, abstracting away the intricacies of concurrent and multipart transfers, allowing developers to focus on application logic rather than low-level S3 API calls.