Migrating repositories from GitHub Enterprise Server to GitHub Enterprise Cloud

In this article

You can migrate repositories from GitHub Enterprise Server to GitHub Enterprise Cloud, using the GitHub CLI or API.

API GitHub CLI

About repository migrations with GitHub Enterprise Importer

You can run your migration with either the GitHub CLI or the API.

The GitHub CLI simplifies the migration process and is recommended for most customers. Advanced customers with heavy customization needs can use the API to build their own integrations with GitHub Enterprise Importer.

If you choose to use the API, you'll need to write your own scripts or use an HTTP client like Insomnia. You can learn more about getting started with GitHub's APIs in "Getting started with the REST API" and "Forming calls with GraphQL."

To migrate your repositories from GitHub Enterprise Server to GitHub Enterprise Cloud with the APIs, you will:

1. Create a personal access token for both the source and destination organization
2. Fetch the `ownerId` of the destination organization on GitHub Enterprise Cloud
3. Set up a migration source via GitHub.com's GraphQL API to identify where you're migrating from
4. For each repository you want to migrate, repeat these steps.
    - Use the REST API on your GitHub Enterprise Server instance to generate migration archives for your repository
    - Upload your migration archives to a location where they can be accessed by GitHub.com
    - Start your migration using the GraphQL API for GitHub.com, passing in your archive URLs
    - Check the status of your migration via the GraphQL API
    - Validate your migration and check the error log

To see instructions for using the API, use the tool switcher at the top of the page.

To see instructions for using the GitHub CLI, use the tool switcher at the top of the page.

Prerequisites

- To ensure you understand the known support limitations of the Importer, review "About GitHub Enterprise Importer."
- We strongly recommend that you perform a trial run of your migration and complete your production migration soon after. To learn more about trial run best practices, see "Preparing to run a migration with GitHub Enterprise Importer."
- While not required, we recommend halting work during your production migration. The Importer doesn't support delta migrations, so any changes that happen during the migration will not migrate. If you choose not to halt work during your production migration, you'll need to manually migrate these changes.
- You must be either an organization owner or be granted the migrator role for both the source and destination organizations. For more information, see "Granting the migrator role for GitHub Enterprise Importer."
- If you use GitHub Enterprise Server 3.8 or higher, you need access to the Management Console.

Step 1. Create your personal access token

1. Create and record a personal access token (classic) that will authenticate for the destination organization on GitHub Enterprise Cloud, making sure that the token meets all requirements. For more information, see "Managing access for GitHub Enterprise Importer."
2. Create and record a personal access token that will authenticate for the source organization, making sure that this token also meets all of the same requirements.

Step 2: Get the `ownerId` for the destination organization

As an organization owner in GitHub Enterprise Cloud, use the `GetOrgInfo` query to return

the `ownerId`, also called the organization ID, for the organization you want to own the migrated repositories. You'll need the `ownerId` to identify your migration destination.

`GetOrgInfo` query

```
query(
  $login: String!
){
  organization (login: $login)
  {
    login
    id
    name
    databaseId
  }
}
```

Query variable	Description
`login`	Your organization name.

`GetOrgInfo` response

```
{
  "data": {
    "organization": {
      "login": "Octo",
      "id": "MDEyOk9yZ2FuaXphdGlvbjU2MTA=",
      "name": "Octo-org",
      "databaseId": 5610
    }
  }
}
```

In this example, `MDEyOk9yZ2FuaXphdGlvbjU2MTA=` is the organization ID or `ownerId`, which we'll use in the next step.

Step 3: Set up blob storage

Because many GitHub Enterprise Server instances sit behind firewalls, for GitHub Enterprise Server versions 3.8 or higher, we use blob storage as an intermediate location to store your data that GitHub can access.

You must first set up blob storage with a supported cloud provider, then configure your settings in the Management Console of your GitHub Enterprise Server instance.

> **Note**: You only need to configure blob storage if you use GitHub Enterprise Server versions 3.8 or higher. If you use GitHub Enterprise Server versions 3.7 or lower, skip to "Step 4: Set up a migration source in GitHub Enterprise Cloud."
>
> Blob storage is required to migrate repositories with large Git source or metadata. If you use GitHub Enterprise Server versions 3.7 or lower, you will not be able to perform migrations where your Git source or metadata exports exceed 2GB. To perform these migrations, update to GitHub Enterprise Server versions 3.8 or higher.

Setting up blob storage with a supported cloud provider

The GitHub CLI supports the following blob storage providers:

- Amazon Web Services (AWS) S3
- Azure Blob Storage

Setting up an AWS S3 storage bucket

In AWS, set up a S3 bucket. For more information, see Creating a bucket in the AWS documentation.

You will also need an AWS access key and secret key with the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucketMultipartUploads",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::github-migration-bucket",
                "arn:aws:s3:::github-migration-bucket/*"
            ]
        }
    ]
}
```

Note: GitHub Enterprise Importer does not delete your archive from AWS after your migration is finished. To reduce storage costs, we recommend configuring auto-deletion of your archive after a period of time. For more information, see Setting lifecycle configuration on a bucket in the AWS documentation.

Setting up an Azure Blob Storage account

In Azure, create a storage account and make a note of your connection string. For more information, see Manage storage account access keys in Microsoft Docs.

Note: GitHub Enterprise Importer does not delete your archive from Azure Blob Storage after your migration is finished. To reduce storage costs, we recommend configuring auto-deletion of your archive after a period of time. For more information, see Optimize costs by automatically managing the data lifecycle in Microsoft Docs.

Configuring blob storage in the Management Console of your GitHub Enterprise Server instance

After you set up an AWS S3 storage bucket or Azure Blob Storage storage account, configure blob storage in the Management Console of your GitHub Enterprise Server instance. For more information about the Management Console, see "Administering your instance from the Management Console."

1. From an administrative account on GitHub Enterprise Server, in the upper-right corner of any page, click 🚀.
2. If you're not already on the "Site admin" page, in the upper-left corner, click **Site**

admin.

3. In the "🚀 Site admin" sidebar, click **Management Console**.
4. Log into the Management Console.
5. In the top navigation bar, click **Settings**.
6. Under **Migrations**, click **Enable GitHub Migrations**.
7. Optionally, to import storage settings you configured for GitHub Actions, select **Copy Storage settings from Actions**. For more information see, "Enabling GitHub Actions with Azure Blob storage" and "Enabling GitHub Actions with Amazon S3 storage."
8. If you do not import storage settings from GitHub Actions, select either **Azure Blob Storage** or **Amazon S3** and fill in the required details.

   > **Note**: If you use Amazon S3, you must set the "AWS Service URL" to the standard endpoint for the AWS region where your bucket is located. For example, if your bucket is located in the `eu-west-1` region, the "AWS Service URL" would be `https://s3.eu-west-1.amazonaws.com`. Your GitHub Enterprise Server instance's network must allow access to this host. Gateway endpoints are not supported, such as `bucket.vpce-0e25b8cdd720f900e-argc85vg.s3.eu-west-1.vpce.amazonaws.com`. For more information about gateway endpoints, see Gateway endpoints for Amazon S3 in the AWS documentation.

9. Click **Test storage settings**.
10. Click **Save settings**.

Step 4: Set up a migration source in GitHub Enterprise Cloud

You can set up a migration source using the `createMigrationSource` query. You'll need to supply the `ownerId`, or organization ID, gathered from the `GetOrgInfo` query.

Your migration source is your organization on GitHub Enterprise Server.

`createMigrationSource` mutation

```
mutation createMigrationSource($name: String!, $url: String!, $ownerId: ID!) {
  createMigrationSource(input: {name: $name, url: $url, ownerId: $ownerId, type:
GITHUB_ARCHIVE}) {
    migrationSource {
      id
      name
      url
      type
    }
  }
}
```

> **Note:** Make sure to use `GITHUB_ARCHIVE` for `type`.

Query variable	Description
`name`	A name for your migration source. This name is

	for your own reference, so you can use any string.
`ownerId`	The organization ID of your organization on GitHub Enterprise Cloud.
`url`	The URL for your GitHub Enterprise Server instance. This URL does not need to be accessible from GitHub Enterprise Cloud.

`createMigrationSource` response

```
{
  "data": {
    "createMigrationSource": {
      "migrationSource": {
        "id": "MS_kgDaACRjZTY5NGQ1OC1mNDkyLTQ2NjgtOGE1NS00MGUxYTdlZmQwNWQ",
        "name": "GHES Source",
        "url": "https://my-ghes-hostname.com",
        "type": "GITHUB_ARCHIVE"
      }
    }
  }
}
```

In this example, `MS_kgDaACRjZTY5NGQ1OC1mNDkyLTQ2NjgtOGE1NS00MGUxYTdlZmQwNWQ` is the migration source ID, which we'll use in a later step.

Step 5: Generate migration archives on your GitHub Enterprise Server instance

You will use the REST API to start two "migrations" in GitHub Enterprise Server: one to generate an archive of your repository's Git source, and one to generate an archive of your repository's metadata (such as issues and pull requests).

To generate the Git source archive, make a `POST` request to `https://HOSTNAME/api/v3/orgs/ORGANIZATION/migrations`, replacing `HOSTNAME` with the hostname of your GitHub Enterprise Server instance, and `ORGANIZATION` with your organization's login.

In the body, specify the single repository you want to migrate. Your request will look something like this:

```
POST /api/v3/orgs/acme-corp/migrations HTTP/1.1
Accept: application/vnd.github+json
Authorization: Bearer <TOKEN>
Content-Type: application/json
Host: github.acmecorp.net

{
  "repositories": ["repository_to_migrate"],
  "exclude_metadata": true
}
```

To generate your metadata archive, send a similar request to the same URL with the following body:

```
{
  "repositories": ["repository_to_migrate"],
  "exclude_git_data": true,
  "exclude_releases": false,
```

```
  "exclude_owner_projects": true
}
```

Each of these two API calls will return a JSON response including the ID of the migration you have started.

```
HTTP/1.1 201 Created

{
  "id": 123,
  // ...
}
```

For more information, see "Start an organization migration" in the REST API documentation.

Generating the archives can take a while, depending on the amount of data. You can regularly check the status of the two migrations with the "Get an organization migration status" API until the `state` of the migration changes to `exported`.

```
GET /api/v3/orgs/acme-corp/migrations/123 HTTP/1.1
Accept: application/vnd.github+json
Authorization: Bearer <TOKEN>
Host: github.acmecorp.net

HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 123,
  "state": "exported",
  // ...
}
```

For more information, see "Get an organization migration status" in the REST API documentation.

Note: If your migration moves to the `failed` state rather than the `exported` state, try starting the migration again. If the migration fails repeatedly, we recommend generating the archives using `ghe-migrator` instead of the API.

Follow the steps in "Exporting migration data from your enterprise," adding only one repository to the migration. At the end of the process, you will have a single migration archive with your Git source and metadata, and you can move to step 6 in this article.

After the `state` of a migration moves to `exported`, you can fetch the migration's URL using the "Download an organization migration archive" API.

```
GET /api/v3/orgs/acme-corp/migrations/123/archive HTTP/1.1
Accept: application/vnd.github+json
Authorization: Bearer <TOKEN>
Host: github.acmecorp.net

HTTP/1.1 302 Found
Location: https://media.github.acmecorp.net/migrations/123/archive/cca2ebe9-7403-
4ffa-9b6a-4c9e16c94410?token=AAAAABEWE7JP4H2HACKEGMTDOYRC6
```

The API will return a `302 Found` response with a `Location` header redirecting to the URL where the downloadable archive is located. Download the two files: one for the Git source, and one for the metadata.

For more information, see "Download an organization migration archive" in the REST API

documentation.

After both migrations have completed and you have downloaded the archives, you can move to the next step.

Step 6: Upload your migration archives

To import your data into GitHub Enterprise Cloud, you must pass both archives for each repository (Git source and metadata) from your machine to GitHub.com, using our GraphQL API.

If you're using GitHub Enterprise Server 3.7 or lower, you must first generate URLs for the archives that are accessible by GitHub.com. Most customers choose to upload the archives to a cloud provider's blob storage service, such as Amazon S3 or Azure Blob Storage, then generate a short-lived URL for each.

If you're using GitHub Enterprise Server 3.8 or higher, your instance uploads the archives and generates the URLs for you. The `Location` header in the previous step will return the short-lived URL.

You may need to allowlist GitHub's IP ranges. For more information, see "Managing access for GitHub Enterprise Importer."

Step 7: Start your repository migration

When you start a migration, a single repository and its accompanying data migrates into a brand new GitHub repository that you identify.

If you want to move multiple repositories at once from the same source organization, you can queue multiple migrations. You can run up to 5 repository migrations at the same time.

`startRepositoryMigration` mutation

```
mutation startRepositoryMigration (
  $sourceId: ID!,
  $ownerId: ID!,
  $repositoryName: String!,
  $continueOnError: Boolean!,
  $accessToken: String!,
  $githubPat: String!,
  $gitArchiveUrl: String!,
  $metadataArchiveUrl: String!,
  $sourceRepositoryUrl: URI!,
  $targetRepoVisibility: String!
){
  startRepositoryMigration( input: {
    sourceId: $sourceId,
    ownerId: $ownerId,
    repositoryName: $repositoryName,
    continueOnError: $continueOnError,
    accessToken: $accessToken,
    githubPat: $githubPat,
    targetRepoVisibility: $targetRepoVisibility
    gitArchiveUrl: $gitArchiveUrl,
    metadataArchiveUrl: $metadataArchiveUrl,
    sourceRepositoryUrl: $sourceRepositoryUrl,
  }) {
    repositoryMigration {
      id
      migrationSource {
        id
```

```
        name
        type
      }
      sourceUrl
    }
  }
}
```

Query variable	Description
`sourceId`	Your migration source `id` returned from the `createMigrationSource` mutation.
`ownerId`	The organization ID of your organization on GitHub Enterprise Cloud.
`repositoryName`	A custom unique repository name not currently used by any of your repositories owned by the organization on GitHub Enterprise Cloud. An error-logging issue will be created in this repository when your migration is complete or has stopped.
`continueOnError`	Migration setting that allows the migration to continue when encountering errors that don't cause the migration to fail. Must be `true` or `false`. We highly recommend setting `continueOnError` to `true` so that your migration will continue unless the Importer can't move Git source or the Importer has lost connection and cannot reconnect to complete the migration.
`githubPat`	The personal access token for your destination organization on GitHub Enterprise Cloud. For personal access token requirements, see "Managing access for GitHub Enterprise Importer."
`accessToken`	The personal access token for your source.
`targetRepoVisibility`	The visibility of the new repository. Must be `private`, `public`, or `internal`. If not set, your repository is migrated as private.
`gitArchiveUrl`	A GitHub Enterprise Cloud-accessible URL for your Git source archive.
`metadataArchiveUrl`	A GitHub Enterprise Cloud-accessible URL for your metadata archive.
`sourceRepositoryUrl`	The URL for your repository on your GitHub Enterprise Server instance. This is required, but GitHub Enterprise Cloud will not communicate directly with your GitHub Enterprise Server instance.

In the next step, you'll use the migration ID returned from the `startRepositoryMigration` mutation to check the migration status.

Step 8: Check the status of your migration

To detect any migration failures and ensure your migration is working, you can check

your migration status using the `getMigration` query. You can also check the status of multiple migrations with `getMigrations`.

The `getMigration` query will return with a status to let you know if the migration is `queued`, `in progress`, `failed`, or `completed`. If your migration failed, the Importer will provide a reason for the failure.

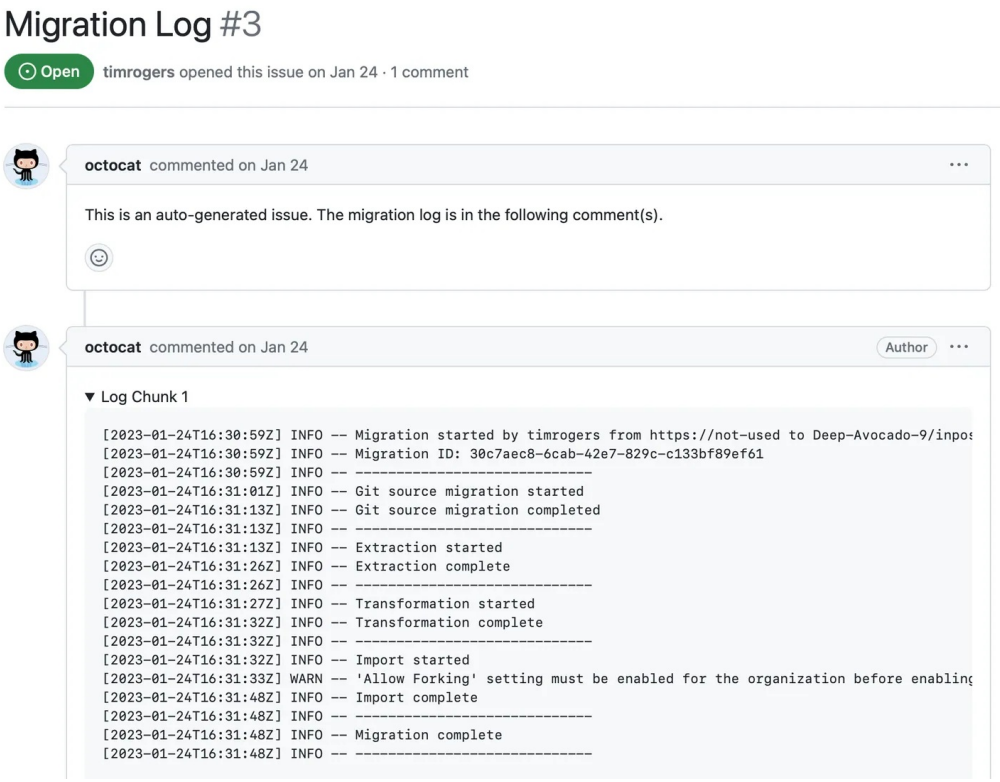
`getMigration` query

```
query (
  $id: ID!
){
  node( id: $id ) {
    ... on Migration {
      id
      sourceUrl
      migrationSource {
        name
      }
      state
      failureReason
    }
  }
}
```

Query variable	Description
`id`	The `id` of your migration that the `startRepositoryMigration` mutation returned.

Step 9: Validate your migration and check the error log

To finish your migration, we recommend that you check the "Migration Log" issue. This issue is created on GitHub in the destination repository.



Finally, we recommend that you review your migrated repositories for a soundness

check.

Step 1: Install the GEI extension of the GitHub CLI

If this is your first migration, you'll need to install the GEI extension of the GitHub CLI. For more information about the GitHub CLI, see "About GitHub CLI."

1. Install the GitHub CLI. For installation instructions for GitHub CLI, see the GitHub CLI repository.

   > **Note:** You need version 2.4.0 or newer of GitHub CLI. You can check the version you have installed with the `gh --version` command.

2. Install the GEI extension.

   Shell

   ```
   gh extension install github/gh-gei
   ```

Any time you need help with the GEI extension, you can use the `--help` flag with a command. For example, `gh gei --help` will list all the available commands, and `gh gei migrate-repo --help` will list all the options available for the `migrate-repo` command.

Step 2: Update the GEI extension of the GitHub CLI

The GEI extension is updated weekly. To make sure you're using the latest version, update the extension.

```
gh extension upgrade github/gh-gei
```

Step 3: Set environment variables

Before you can use the GEI extension to migrate to GitHub Enterprise Cloud, you must create personal access tokens that can access the source and destination organizations, then set the personal access tokens as environment variables.

1. Create and record a personal access token (classic) that will authenticate for the destination organization on GitHub Enterprise Cloud, making sure that the token meets all requirements. For more information, see "Managing access for GitHub Enterprise Importer."
2. Create and record a personal access token that will authenticate for the source organization, making sure that this token also meets all of the same requirements.
3. Set environment variables for the personal access tokens, replacing TOKEN in the commands below with the personal access tokens you recorded above. Use `GH_PAT` for the destination organization and `GH_SOURCE_PAT` for the source organization.
   - If you're using Terminal, use the `export` command.

     Shell

```
export GH_PAT="TOKEN"
export GH_SOURCE_PAT="TOKEN"
```

- If you're using PowerShell, use the `$env` command.

Shell

```
$env:GH_PAT="TOKEN"
$env:GH_SOURCE_PAT="TOKEN"
```

Step 4: Set up blob storage

Because many GitHub Enterprise Server instances sit behind firewalls, we use blob storage as an intermediate location to store your data that GitHub can access.

First, you must set up blob storage with a supported cloud provider. Then, you must configure your credentials for the storage provider in the Management Console or GitHub CLI.

Setting up blob storage with a supported cloud provider

The GitHub CLI supports the following blob storage providers:

- Amazon Web Services (AWS) S3
- Azure Blob Storage

Setting up an AWS S3 storage bucket

In AWS, set up a S3 bucket. For more information, see Creating a bucket in the AWS documentation.

You will also need an AWS access key and secret key with the following permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucketMultipartUploads",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::github-migration-bucket",
                "arn:aws:s3:::github-migration-bucket/*"
            ]
        }
    ]
}
```

Note: GitHub Enterprise Importer does not delete your archive from AWS after your migration is

finished. To reduce storage costs, we recommend configuring auto-deletion of your archive after a period of time. For more information, see Setting lifecycle configuration on a bucket in the AWS documentation.

Setting up an Azure Blob Storage storage account

In Azure, create a storage account and make a note of your connection string. For more information, see Manage storage account access keys in Microsoft Docs.

> **Note:** GitHub Enterprise Importer does not delete your archive from Azure Blob Storage after your migration is finished. To reduce storage costs, we recommend configuring auto-deletion of your archive after a period of time. For more information, see Optimize costs by automatically managing the data lifecycle in Microsoft Docs.

Configuring your blob storage credentials

After you set up blob storage with a supported cloud provider, you must configure your credentials for the storage provider in GitHub:

- If you use GitHub Enterprise Server 3.8 or higher, configure your credentials in the Management Console.
- If you use GitHub Enterprise Server 3.7 or lower, configure the credentials in the GitHub CLI.

Configuring blob storage in the Management Console of your GitHub Enterprise Server instance

> **Note:** You only need to configure blob storage in the Management Console if you use GitHub Enterprise Server 3.8 or higher. If you use 3.7 or lower, configure your credentials in the GitHub CLI instead.

After you set up an AWS S3 storage bucket or Azure Blob Storage storage account, configure blob storage in the Management Console of your GitHub Enterprise Server instance. For more information about the Management Console, see "Administering your instance from the Management Console."

1. From an administrative account on GitHub Enterprise Server, in the upper-right corner of any page, click 🚀.
2. If you're not already on the "Site admin" page, in the upper-left corner, click **Site admin**.
3. In the "🚀 Site admin" sidebar, click **Management Console**.
4. Log into the Management Console.
5. In the top navigation bar, click **Settings**.
6. Under **Migrations**, click **Enable GitHub Migrations**.
7. Optionally, to import storage settings you configured for GitHub Actions, select **Copy Storage settings from Actions**. For more information see, "Enabling GitHub Actions with Azure Blob storage" and "Enabling GitHub Actions with Amazon S3 storage."
8. If you do not import storage settings from GitHub Actions, select either **Azure Blob Storage** or **Amazon S3** and fill in the required details.

> **Note**: If you use Amazon S3, you must set the "AWS Service URL" to the standard endpoint for the AWS region where your bucket is located. For example, if your bucket is located in the `eu-west-1` region, the "AWS Service URL" would be `https://s3.eu-west-1.amazonaws.com`. Your GitHub Enterprise Server instance's network must allow access to this host. Gateway endpoints are not supported, such as `bucket.vpce-0e25b8cdd720f900e-argc85vg.s3.eu-west-1.vpce.amazonaws.com`. For more information about gateway endpoints, see Gateway endpoints for Amazon S3 in the AWS documentation.

9. Click **Test storage settings**.
10. Click **Save settings**.

Configuring your blob storage credentials in the GitHub CLI

> **Note:** You only need to configure your blob storage credentials in the GitHub CLI if you use GitHub Enterprise Server 3.7 or lower. If you use 3.8 or higher, configure blob storage in the Management Console instead.
>
> If you configure your blob storage credentials in the GitHub CLI, you will not be able to perform migrations where your Git source or metadata exports exceed 2GB. To perform these migrations, upgrade to GitHub Enterprise Server 3.8 or higher.

Configuring AWS S3 credentials in the GitHub CLI

When you're ready to run your migration, you will need to provide your AWS credentials to the GitHub CLI: region, access key, secret key, and session token (if required). You can pass them as arguments, or set environment variables called `AWS_REGION`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`.

You will also need to pass in the name of the S3 bucket using the `--aws-bucket-name` argument.

Configuring Azure Blob Storage account credentials in the GitHub CLI

When you're ready to run your migration, you can either pass your connection string into the GitHub CLI as an argument, or pass it in using an environment variable called `AZURE_STORAGE_CONNECTION_STRING`.

Step 5: Generate a migration script

If you want to migrate multiple repositories to GitHub Enterprise Cloud at once, use the GitHub CLI to generate a migration script. The resulting script will contain a list of migration commands, one per repository.

> **Note:** Generating a script outputs a PowerShell script. If you're using Terminal, you will need to output the script with the `.ps1` file extension and install PowerShell for either Mac or Linux to run it.

If you want to migrate a single repository, skip to the next step.

Generating a migration script

You must follow this step from a computer that can access the API for your GitHub Enterprise Server instance. If you can access your instance from the browser, then the computer has the correct access.

To generate a migration script, run the `gh gei generate-script` command.

For GitHub Enterprise Server 3.8 or later, or if you're using 3.7 or lower with Azure Blob Storage, use the following flags:

Shell

```
gh gei generate-script --github-source-org SOURCE \
  --github-target-org DESTINATION \
  --output FILENAME \
  --ghes-api-url GHES-API-URL
```

If you're using GitHub Enterprise Server 3.7 or lower with AWS S3, use the following flags:

Shell

```
gh gei generate-script --github-source-org SOURCE \
  --github-target-org DESTINATION \
  --output FILENAME \
  --ghes-api-url GHES-API-URL \
  --aws-bucket-name AWS-BUCKET-NAME
```

If your GitHub Enterprise Server instance uses a self-signed or invalid SSL certificate, use the `--no-ssl-verify` flag. With this flag, the GitHub CLI will skip verifying the SSL certificate when extracting data from your instance only. All other calls will verify SSL.

If you want the script to download the migration log for each migrated repository, add the `--download-migration-logs` flag. For more information about migration logs, see "Accessing your migration logs for GitHub Enterprise Importer."

Replace the placeholders in the command above with the following values.

Placeholder	Value
SOURCE	Name of the source organization
DESTINATION	Name of the destination organization
FILENAME	A filename for the resulting migration script If you're using Terminal, use a `.ps1` file extension as the generated script requires PowerShell to run. You can install PowerShell for Mac or Linux.
GHES-API-URL	The URL for your GitHub Enterprise Server instance's API, such as `https://myghes.com/api/v3`
AWS-BUCKET-NAME	The bucket name for your AWS S3 bucket

Reviewing the migration script

After you generate the script, review the file and, optionally, edit the script.

- If there are any repositories you don't want to migrate, delete or comment out the corresponding lines.
- If you want any repositories to have a different name in the destination organization, update the value for the corresponding `--target-repo` flag.

Note: If your repository has more than 10 GB of releases data, releases cannot be migrated. Use the `--skip-releases` flag to migrate the repository without releases.

Step 6: Migrate repositories

You can migrate multiple repositories with a migration script or a single repository with the `gh gei migrate-repo` command.

When you migrate repositories, the GEI extension of the GitHub CLI performs the following steps:

1. Connects to your GitHub Enterprise Server instance and generates two migration archives per repository, one for the Git source and one for the metadata
2. Uploads the migration archives to the blob storage provider of your choice
3. Starts your migration in GitHub Enterprise Cloud, using the URLs of the archives stored with your blob storage provider
4. Deletes the migration archive from your local machine

Migrate multiple repositories

If you're migrating from GitHub Enterprise Server 3.7 or earlier, before you run your script, you must set additional environment variables to authenticate to your blob storage provider.

- For Azure Blob Storage, set `AZURE_STORAGE_CONNECTION_STRING` to the connection string for your Azure storage account.

  Only connection strings using storage account access keys are supported. Connection strings which use shared access signatures (SAS) are not supported. For more information about storage account access keys, see Manage storage account access keys in the Azure documentation.

- For AWS S3, set the following environment variables.
  - `AWS_ACCESS_KEY`: The access key for your bucket
  - `AWS_SECRET_KEY`: The secret key for your bucket
  - `AWS_REGION`: The AWS region where your bucket is located
  - `AWS_SESSION_TOKEN`: The session token, if you're using AWS temporary credentials (see Using temporary credentials with AWS resources in the AWS documentation)

To migrate multiple repositories, run the script you generated above. Replace FILENAME in the commands below with the filename you provided when generating the script.

- If you're using Terminal, use `./`.

```shell
./FILENAME
```

- If you're using PowerShell, use `.\`.

```shell
```

```
.\FILENAME
```

Migrate a single repository

You must follow this step from a computer that can access the API for your GitHub Enterprise Server instance. If you can access your instance from the browser, then the computer has the correct access.

To migrate a single repository, use the `gh gei migrate-repo` command.

If you're using GitHub Enterprise Server 3.8 or later, use the following flags:

Shell

```
gh gei migrate-repo --github-source-org SOURCE --source-repo CURRENT-NAME --
github-target-org DESTINATION --target-repo NEW-NAME --ghes-api-url GHES-API-URL
```

If you're migrating from GitHub Enterprise Server 3.7 or earlier and using Azure Blob Storage as your blob storage provider, use the following flags to authenticate:

Shell

```
gh gei migrate-repo --github-source-org SOURCE --source-repo CURRENT-NAME --
github-target-org DESTINATION --target-repo NEW-NAME \
    --ghes-api-url GHES-API-URL --azure-storage-connection-string
"AZURE_STORAGE_CONNECTION_STRING"
```

If you're migrating from GitHub Enterprise Server 3.7 or earlier and using Amazon S3 as your blob storage provider, use the following flags to authenticate:

Shell

```
gh gei migrate-repo --github-source-org SOURCE --source-repo CURRENT-NAME --
github-target-org DESTINATION --target-repo NEW-NAME \
    --ghes-api-url GHES-API-URL --aws-bucket-name "AWS-BUCKET-NAME"
```

If your GitHub Enterprise Server instance uses a self-signed or invalid SSL certificate, use the `--no-ssl-verify` flag. With this flag, the GitHub CLI will skip verifying the SSL certificate when extracting data from your instance only. All other calls will verify SSL.

> **Note:** If your repository has more than 10 GB of releases data, releases cannot be migrated. Use the `--skip-releases` flag to migrate the repository without releases.

Replace the placeholders in the command above with the following values.

Placeholder	Value
SOURCE	Name of the source organization
CURRENT-NAME	The name of the repository you want to migrate
DESTINATION	Name of the destination organization
NEW-NAME	The name you want the migrated repository to have
GHES-API-URL	The URL for your GitHub Enterprise Server

GHES-API-URL	The URL for your GitHub Enterprise Server instance's API, such as `https://myghes.com/api/v3`
AZURE_STORAGE_CONNECTION_STRING	The connection string for your Azure storage account. Make sure to quote the connection string, which contains characters that would likely otherwise be interpreted by shell.
AWS-BUCKET-NAME	The bucket name for your AWS S3 bucket

Step 7: Validate your migration and check the error log

When your migration is complete, we recommend reviewing your migration log. For more information, see "Accessing your migration logs for GitHub Enterprise Importer."

We recommend that you review your migrated repositories for a soundness check.

After your migration has finished, we recommend deleting the archives from your storage container. If you plan to complete additional migrations, delete the archive placed into your storage container by the ADO2GH extension. If you're done migrating, you can delete the entire container.