Cloud Functions in Go must provide all of their dependencies via either Go modules with a `go.mod` file, or a `vendor` directory. Your function cannot specify dependencies using both Go modules and a `vendor` directory at the same time.

The Go runtime includes a number of system packages (/functions/docs/reference/go-system-packages) in the execution environment. If your function uses a dependency that requires a system package that is not listed, you can request a package (/functions/docs/reference/go-system-packages#requesting_a_package).

The content in this document applies to both Go 1.11 and Go 1.13.

Cloud Functions support Go's experimental Modules functionality (https://golang.org/doc/go1.11#modules), which enables you to specify dependencies in a `go.mod` file at the root of your project. When you deploy your function, dependencies specified in the `go.mod` file will be fetched and built automatically.

The behavior of Go modules differs depending on whether you are developing inside or outside of `GOPATH`. To determine whether you are inside `GOPATH`:

1. Navigate to your project directory.
2. Find your `GOPATH` by running the command:

   This outputs a line similar to:

3. Find your current working directory by running:

If your working directory begins with ***YOUR_GOPATH***, you are inside `GOPATH`. In this case, generate your `go.mod` file by running the following three commands:

If your working directory does not begin with ***YOUR_GOPATH***, you are outside `GOPATH`. In this case, generate your `go.mod` file by running the commands:

In the above example, ***MODULE*** is the name of your module. For example, your module name might be `example.com/myproject` (note that the domain name is required). The `go` command automatically detects the module name when you are inside `GOPATH`.

After you've created a `go.mod` file, you can use the `go get` command to fetch dependencies and automatically add them to your project. For example:

In the above example, ***DEPENDENCY*** is a dependency that you want to add to your function. For example, the command `go get cloud.google.com/go/storage` adds the Cloud Storage client library to your function.

ng: If you have both a **`go.mod`** file and a **`vendor`** directory at the root of your project, the contents of the **`vendor`** dire ignored when your function is built in the cloud. To ensure that your **`vendor`** directory is used, you must exclude the **d** file from your project's source code prior to deployment. If you are using the **`gcloud`** command-line tool, you can en **.mod** is not uploaded by using **[`.gcloudignore`]** (/sdk/gcloud/reference/topic/gcloudignore).

Cloud Functions also allows you to include your dependencies via a [`vendor` directory] (https://golang.org/cmd/go/#hdr-Vendor_Directories). Most of the time, `vendor` directories are maintained with a dependency manager. You can use any dependency manager you like. For example, you can use Go's Modules functionality to create a `vendor` directory from your `go.mod` file.

If you have a `go.mod` file and a `vendor` directory, the `vendor` directory will be ignored when you deploy your function. You can use a [`.gcloudignore`](/sdk/gcloud/reference/topic/gcloudignore) (/sdk/gcloud/reference/topic/gcloudignore) file to avoid uploading your `go.mod` and `go.sum` files, in which case the contents of your `vendor` directory will be respected:

1. Create a `.gcloudignore` file at the root of your project directory with the following contents:

2. Create a `vendor` directory using the contents of your `go.mod` file by running the following command:

If your function's dependencies are hosted in a repository that is not publicly accessible, you must use a `vendor` directory to fetch your dependencies before deploying your function. If you plan to use a `go.mod` file, see the instructions above to avoid conflicts between the `go.mod` file and the `vendor` directory.