Gophercloud: an OpenStack SDK for Go

build passing coverage 80%

Gophercloud is an OpenStack Go SDK.

Useful links

- Reference documentation
- Effective Go

How to install

Before installing, you need to ensure that your GOPATH environment variable is pointing to an appropriate directory where you want to install Gophercloud:

```
mkdir $HOME/go
export GOPATH=$HOME/go
```

To protect yourself against changes in your dependencies, we highly recommend choosing a dependency management solution for your projects, such as godep. Once this is set up, you can install Gophercloud as a dependency like so:

```
go get github.com/gophercloud/gophercloud

# Edit your code to import relevant packages from
"github.com/gophercloud/gophercloud"

godep save ./...
```

This will install all the source files you need into a `Godeps/_workspace` directory, which is referenceable from your own source files when you use the `godep go` command.

Getting started

Credentials

Because you'll be hitting an API, you will need to retrieve your OpenStack credentials and either store them as environment variables or in your local Go files. The first method is recommended because it decouples credential information from source code, allowing you to push the latter to your version control system without any security risk.

You will need to retrieve the following:

- username
- password
- a valid Keystone identity URL

For users that have the OpenStack dashboard installed, there's a shortcut. If you visit the `project/access_and_security` path in Horizon and click on the "Download OpenStack RC File" button at the top right hand corner, you will download a bash file that exports all of your access details to environment variables. To execute the file, run `source admin-openrc.sh` and you will be prompted for your password.

Authentication

Once you have access to your credentials, you can begin plugging them into Gophercloud. The next step is authentication, and this is handled by a base "Provider" struct. To get one, you can either pass in your credentials explicitly, or tell Gophercloud to use environment variables:

```
import (
  "github.com/gophercloud/gophercloud"
  "github.com/gophercloud/gophercloud/openstack"
  "github.com/gophercloud/gophercloud/openstack/utils"
)

// Option 1: Pass in the values yourself
opts := gophercloud.AuthOptions{
  IdentityEndpoint: "https://openstack.example.com:5000/v2.0",
  Username: "{username}",
  Password: "{password}",
}

// Option 2: Use a utility function to retrieve all your environment variables
opts, err := openstack.AuthOptionsFromEnv()
```

Once you have the `opts` variable, you can pass it in and get back a `ProviderClient` struct:

```
provider, err := openstack.AuthenticatedClient(opts)
```

The `ProviderClient` is the top-level client that all of your OpenStack services derive from. The provider contains all of the authentication details that allow your Go code to access the API - such as the base URL and token ID.

Provision a server

Once we have a base Provider, we inject it as a dependency into each OpenStack service. In order to work with the Compute API, we need a Compute service client; which can be created like so:

```
client, err := openstack.NewComputeV2(provider, gophercloud.EndpointOpts{
  Region: os.Getenv("OS_REGION_NAME"),
})
```

We then use this `client` for any Compute API operation we want. In our case, we want to provision a new server - so we invoke the `Create` method and pass in the flavor ID (hardware specification) and image ID (operating system) we're interested in:

```
import "github.com/gophercloud/gophercloud/openstack/compute/v2/servers"

server, err := servers.Create(client, servers.CreateOpts{
  Name:      "My new server!",
  FlavorRef: "flavor_id",
    ImageRef:  "image_id",
}).Extract()
```

The above code sample creates a new server with the parameters, and embodies the new resource in the `server` variable (a `servers.Server` struct).

Advanced Usage

Have a look at the FAQ for some tips on customizing the way Gophercloud works.

Backwards-Compatibility Guarantees

None. Vendor it and write tests covering the parts you use.

Contributing

See the contributing guide.

Help and feedback

If you're struggling with something or have spotted a potential bug, feel free to submit an issue to our bug tracker.

Thank You

We'd like to extend special thanks and appreciation to the following:

OpenLab

OpenLab is providing a full CI environment to test each PR and merge for a variety of OpenStack releases.

VEXXHOST

VEXXHOST is providing their services to assist with the development and testing of Gophercloud.