
Gophercloud: an OpenStack SDK for Go

coverage 79%

Reference documentation

Gophercloud is a Go SDK for OpenStack.

Join us on kubernetes slack, on #gophercloud. Visit slack.k8s.io for an invitation.

Note This branch contains the current stable branch of Gophercloud: [v2](#). The legacy stable version can be found in the [v1](#) branch.

How to install

Reference a Gophercloud package in your code:

```
1 import "github.com/gophercloud/gophercloud/v2"
```

Then update your `go.mod`:

```
1 go mod tidy
```

Getting started

Credentials

Because you'll be hitting an API, you will need to retrieve your OpenStack credentials and either store them in a `clouds.yaml` file, 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:

- A valid Keystone identity URL
- Credentials. These can be a username/password combo, a set of Application Credentials, a pre-generated token, or any other supported authentication mechanism.

For users who have the OpenStack dashboard installed, there's a shortcut. If you visit the [project/api_access](#) path in Horizon and click on the "Download OpenStack RC File" button at the top right hand corner, you can download either a `clouds.yaml` file or an `openrc` bash file that exports

all of your access details to environment variables. To use the `clouds.yaml` file, place it at `~/config/openstack/clouds.yaml`. To use the `openrc` file, run `source openrc` and you will be prompted for your password.

Gophercloud authentication

Gophercloud authentication is organized into two layered abstractions: * `ProviderClient` holds the authentication token and can be used to build a `ServiceClient`. * `ServiceClient` specializes against one specific OpenStack module and can directly be used to make API calls.

A provider client is a top-level client that all of your OpenStack service clients 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.

One single Provider client can be used to build as many Service clients as needed.

With `clouds.yaml`

```
1 package main
2
3 import (
4     "context"
5
6     "github.com/gophercloud/gophercloud/v2/openstack"
7     "github.com/gophercloud/gophercloud/v2/openstack/config"
8     "github.com/gophercloud/gophercloud/v2/openstack/config/clouds"
9 )
10
11 func main() {
12     ctx := context.Background()
13
14     // Fetch coordinates from a `cloud.yaml` in the current directory,
15     // or
16     // in the well-known config directories (different for each
17     // operating
18     // system).
19     authOptions, endpointOptions, tlsConfig, err := clouds.Parse()
20     if err != nil {
21         panic(err)
22     }
23
24     // Call Keystone to get an authentication token, and use it to
25     // construct a ProviderClient. All functions hitting the OpenStack
26     // API
27     // accept a `context.Context` to enable tracing and cancellation.
28     providerClient, err := config.NewProviderClient(ctx, authOptions,
29         config.WithTLSConfig(tlsConfig))
30     if err != nil {
```

```

27         panic(err)
28     }
29
30     // Use the ProviderClient and the endpoint options fetched from
31     // `clouds.yaml` to build a service client: a compute client in
32     // this case. Note that the constructor does not accept a `context.
33     // Context`:
34     // no further call to the OpenStack API is needed at this stage.
35     computeClient, err := openstack.NewComputeV2(providerClient,
36         endpointOptions)
37     if err != nil {
38         panic(err)
39     }
40     // use the computeClient

```

With environment variables (`openrc`)

Gophercloud can parse the environment variables set by running `source openrc`:

```

1  package main
2
3  import (
4      "context"
5      "os"
6
7      "github.com/gophercloud/gophercloud/v2"
8      "github.com/gophercloud/gophercloud/v2/openstack"
9  )
10
11  func main() {
12      ctx := context.Background()
13
14      opts, err := openstack.AuthOptionsFromEnv()
15      if err != nil {
16          panic(err)
17      }
18
19      providerClient, err := openstack.AuthenticatedClient(ctx, opts)
20      if err != nil {
21          panic(err)
22      }
23
24      computeClient, err := openstack.NewComputeV2(providerClient,
25          gophercloud.EndpointOpts{
26              Region: os.Getenv("OS_REGION_NAME"),
27          })
28      if err != nil {
29          panic(err)
30      }

```

```
29     }
30
31     // use the computeClient
32 }
```

Manually

You can also generate a “Provider” by passing in your credentials explicitly:

```
1 package main
2
3 import (
4     "context"
5
6     "github.com/gophercloud/gophercloud/v2"
7     "github.com/gophercloud/gophercloud/v2/openstack"
8 )
9
10 func main() {
11     ctx := context.Background()
12
13     providerClient, err := openstack.AuthenticatedClient(ctx,
14         gophercloud.AuthOptions{
15         IdentityEndpoint: "https://openstack.example.com:5000/v2.0",
16         Username:         "username",
17         Password:          "password",
18     })
19     if err != nil {
20         panic(err)
21     }
22
23     computeClient, err := openstack.NewComputeV2(providerClient,
24         gophercloud.EndpointOpts{
25         Region: "RegionName",
26     })
27     if err != nil {
28         panic(err)
29     }
30
31     // use the computeClient
32 }
```

Provision a server

We can use the Compute service client generated above for any Compute API operation we want. In our case, we want to provision a new server. To do this, we invoke the [Create](#) method and pass in the flavor ID (hardware specification) and image ID (operating system) we’re interested in:

```
1 import "github.com/gophercloud/gophercloud/v2/openstack/compute/v2/
   servers"
2
3 func main() {
4     // [...]
5
6     server, err := servers.Create(context.TODO(), computeClient,
7         servers.CreateOpts{
8         Name:      "My new server!",
9         FlavorRef: "flavor_id",
10        ImageRef:  "image_id",
11    }).Extract()
12    // [...]
```

The above code sample creates a new server with the parameters, and returns a `servers.Server`.

Advanced Usage

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

Backwards-Compatibility Guarantees

Gophercloud versioning follows semver.

Before `v1.0.0`, there were no guarantees. Starting with `v1`, there will be no breaking changes within a major release.

See the Release instructions.

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.