The Go Programming Language

# The Go Blog

## Portable Cloud Programming with Go Cloud

24 July 2018

### Introduction

Today, the Go team at Google is releasing a new open source project, Go Cloud, a library and tools for developing on the open cloud. With this project, we aim to make Go the language of choice for developers building portable cloud applications.

This post explains why we started this project, the details of how Go Cloud works, and how to get involved.


### Previous article

Getting to Go: The Journey of Go's Garbage Collector

### Links

golang.org
Install Go
A Tour of Go
Go Documentation
Go Mailing List
Go on Google+
Go+ Community
Go on Twitter

Blog index


### Why portable cloud programming? Why now?

We estimate there are now over one million Go developers worldwide. Go powers many of the most critical cloud infrastructure projects, including Kubernetes, Istio, and Docker. Companies like Lyft, Capital One, Netflix and many more are depending on Go in production. Over the years, we've found that developers love Go for cloud development because of its efficiency, productivity, built-in concurrency, and low latency.

As part of our work to support Go's rapid growth, we have been interviewing teams who work with Go to understand how they use the language and how the Go ecosystem can improve further. One common theme with many organizations is the need for portability across cloud providers. These teams want to deploy robust applications in multi-cloud and hybrid-cloud environments, and migrate their workloads between cloud providers without significant changes to their code.

To achieve this, some teams attempt to decouple their applications from provider-specific APIs in order to produce simpler and more portable code. However the short-term pressure to ship features means teams often sacrifice longer-term efforts toward portability. As a result, most Go applications running in the cloud are tightly coupled to their initial cloud provider.

As an alternative, teams can use Go Cloud, a set of open generic cloud APIs, to write simpler and more portable cloud applications. Go Cloud also sets the foundation for an ecosystem of portable cloud libraries to be built on top of these generic APIs. Go Cloud makes it possible for teams to meet their feature development goals while also preserving the long-term flexibility for multi-cloud and hybrid-cloud architectures. Go Cloud applications can also migrate to the cloud providers that best meet their needs.

### What is Go Cloud?

We have identified common services used by cloud applications and have created generic APIs to work across cloud providers. Today, Go Cloud is launching with blob storage, MySQL database access, runtime configuration, and an HTTP server configured with request logging, tracing, and health checking. Go Cloud offers support for Google Cloud Platform (GCP) and Amazon Web

Services (AWS). We plan to work with cloud industry partners and the Go community to add support for additional cloud providers very soon.

Go Cloud aims to develop vendor-neutral generic APIs for the most-used services across cloud providers such that deploying a Go application on another cloud is simple and easy. Go Cloud also lays the foundation for other open source projects to write cloud libraries that work across providers. Community feedback, from all types of developers at all levels, will inform the priority of future APIs in Go Cloud.

## How does it work?

At the core of Go Cloud is a collection of generic APIs for portable cloud programming. Let's look at an example of using blob storage. You can use the generic type `*blob.Bucket` to copy a file from a local disk to a cloud provider. Let's start by opening an S3 bucket using the included s3blob package:

```
// setupBucket opens an AWS bucket.
func setupBucket(ctx context.Context) (*blob.Bucket, error) {
    // Obtain AWS credentials.
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-east-2"),
    })
    if err != nil {
        return nil, err
    }
    // Open a handle to s3://go-cloud-bucket.
    return s3blob.OpenBucket(ctx, sess, "go-cloud-bucket")
}
```

Once a program has a `*blob.Bucket`, it can create a `*blob.Writer`, which implements `io.Writer`. From there, the program can use the `*blob.Writer` to write data to the bucket, checking that `Close` does not report an error.

```
ctx := context.Background()
b, err := setupBucket(ctx)
if err != nil {
    log.Fatalf("Failed to open bucket: %v", err)
}
data, err := ioutil.ReadFile("gopher.png")
if err != nil {
    log.Fatalf("Failed to read file: %v", err)
}
w, err := b.NewWriter(ctx, "gopher.png", nil)
if err != nil {
    log.Fatalf("Failed to obtain writer: %v", err)
}
_, err = w.Write(data)
if err != nil {
    log.Fatalf("Failed to write to bucket: %v", err)
}
if err := w.Close(); err != nil {
    log.Fatalf("Failed to close: %v", err)
}
```

Notice how the logic of using the bucket does not refer to AWS S3. Go Cloud makes swapping out cloud storage a matter of changing the function used to open the `*blob.Bucket`. The application could instead use Google Cloud Storage by constructing a `*blob.Bucket` using `gcsblob.OpenBucket` without changing the code that copies the file:

```
// setupBucket opens a GCS bucket.
func setupBucket(ctx context.Context) (*blob.Bucket, error) {
    // Open GCS bucket.
    creds, err := gcp.DefaultCredentials(ctx)
    if err != nil {
        return nil, err
    }
    c, err := gcp.NewHTTPClient(gcp.DefaultTransport(),
gcp.CredentialsTokenSource(creds))
    if err != nil {
        return nil, err
    }
    // Open a handle to gs://go-cloud-bucket.
    return gcsblob.OpenBucket(ctx, "go-cloud-bucket", c)
}
```

While different steps are needed to access buckets on different cloud providers, the resulting type used by your application is the same: `*blob.Bucket`. This isolates application code from cloud-specific code. To increase interoperability with existing Go libraries, Go Cloud leverages established interfaces like `io.Writer`, `io.Reader`, and `*sql.DB`.

The setup code needed to access cloud services tends to follow a pattern: higher abstractions are constructed from more basic abstractions. While you could write this code by hand, Go Cloud automates this with **Wire**, a tool that generates cloud-specific setup code for you. The Wire documentation explains how to install and use the tool and the Guestbook sample shows Wire in action.

## How can I get involved and learn more?

To get started, we recommend following the tutorial and then trying to build an application yourself. If you're already using AWS or GCP, you can try migrating parts of your existing application to use Go Cloud. If you're using a different cloud provider or an on-premise service, you can extend Go Cloud to support it by implementing the driver interfaces (like `driver.Bucket`).

We appreciate any and all input you have about your experience. Go Cloud's development is conducted on GitHub. We are looking forward to contributions, including pull requests. File an issue to tell us what could be better or what future APIs the project should support. For updates and discussion about the project, join the project's mailing list.

The project requires contributors to sign the same Contributor License Agreement as that of the Go project. Read the contribution guidelines for more details. Please note, Go Cloud is covered by the Go Code of Conduct.

Thank you for taking the time to learn about Go Cloud, we are excited to work with you to make Go the language of choice for developers building portable cloud applications.

*By Eno Compton and Cassandra Salisbury*