Go (https://cloud.google.com/go/) Guides

Handling sessions with Firestore

This tutorial shows how to handle sessions on App Engine (https://cloud.google.com/appengine/docs/standard/).

Many apps need session handling for authentication and user preferences. The Gorilla Web Toolkit (https://www.gorillatoolkit.org/pkg/sessions) `sessions` package comes with a file system based implementation to perform this function. However, this implementation is unsuitable for an app that can be served from multiple instances, because the session that is recorded in one instance might differ from other instances. The `gorilla/sessions` (https://www.gorillatoolkit.org/pkg/sessions) package also comes with a cookie-based implementation. But, this implementation requires encrypting cookies and storing the entire session on the client, rather than just a session ID, which may be too large for some apps.

Objectives

- Write the app.
- Run the app locally.
- Deploy the app on App Engine.

Costs

This tutorial uses the following billable components of Google Cloud:

- App Engine (https://cloud.google.com/appengine/pricing)
- Firestore (https://cloud.google.com/firestore/pricing)

To generate a cost estimate based on your projected usage, use the pricing calculator (https://cloud.google.com/products/calculator). New Google Cloud users might be eligible for a free trial (https://cloud.google.com/free-trial).

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Cleaning up (#clean-up).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Google Cloud project.

   Note: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).

4. Enable the Firestore API.

   ENABLE THE API (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=FIRESTORE

5. Install and initialize the Cloud SDK (https://cloud.google.com/sdk/docs/).

6. Update `gcloud` components:

   ```
   gcloud components update
   ```

7. Prepare your development environment (https://cloud.google.com/go/docs/setup).

Setting up the project

1. In your terminal window, clone the sample app repository to your local machine:

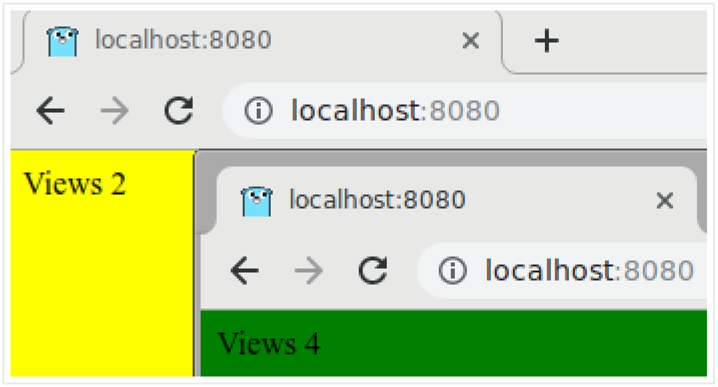
```
git clone https://github.com/GoogleCloudPlatform/golang-samples.git
```

2. Change to the directory that contains the sample code:

```
cd golang-samples/getting-started/sessions
```

Understanding the web app

This app displays greetings in different languages for every user. Returning users are always greeted in the same language.



Before the app can store preferences for a user, you need a way to store information about the current user in a session. This sample app uses Firestore to store session data.

You can use `firestoregorilla` (https://github.com/GoogleCloudPlatform/firestore-gorilla-sessions), a session store that's compatible with `gorilla/sessions` (https://www.gorillatoolkit.org/pkg/sessions).

1. The app starts by importing dependencies, defining an `app` type to hold a `sessions.Store` and an HTML template, and defining the list of greetings.

getting-started/sessions/main.go (https://github.com/GoogleCloudPlatform/golang-samples/blob/master/getting-started/sessions/main.go)

GOOGLECLOUDPLATFORM/GOLANG-SAMPLES/BLOB/MASTER/GETTING-STARTED/SESSIONS/MAIN.GO)

```
// Command sessions starts an HTTP server that uses session state.
package main

import (
        "context"
        "fmt"
        "html/template"
        "log"
        "math/rand"
        "net/http"
        "os"

        "cloud.google.com/go/firestore"
        firestoregorilla "github.com/GoogleCloudPlatform/firestore-gorilla-sess
        "github.com/gorilla/sessions"
)

// app stores a sessions.Store. Create a new app with newApp.
type app struct {
        store sessions.Store
        tmpl  *template.Template
}

// greetings are the random greetings that will be assigned to sessions.
var greetings = []string{
        "Hello World",
        "Hallo Welt",
        "Ciao Mondo",
        "Salut le Monde",
        "Hola Mundo",
}
```

2. Next, the app defines a `main` function, which creates a new `app` instance, registers the index handler, and starts the HTTP server. The `newApp` function creates the `app` instance by initializing a `sessions.Store` with the `firestoregorilla.New` function.

getting-started/sessions/main.go (https://github.com/GoogleCloudPlatform/golang-samples/blob/master/getting-started/sessions/main.go)

GOOGLECLOUDPLATFORM/GOLANG-SAMPLES/BLOB/MASTER/GETTING-STARTED/SESSIONS/MAIN.GO)

```
func main() {
        port := os.Getenv("PORT")
        if port == "" {
                port = "8080"
        }
        projectID := os.Getenv("GOOGLE_CLOUD_PROJECT")
        if projectID == "" {
                log.Fatal("GOOGLE_CLOUD_PROJECT must be set")
        }

        a, err := newApp(projectID)
        if err != nil {
                log.Fatalf("newApp: %v", err)
        }

        http.HandleFunc("/", a.index)

        log.Printf("Listening on port %s", port)
        if err := http.ListenAndServe(":"+port, nil); err != nil {
                log.Fatal(err)
        }
}

// newApp creates a new app.
func newApp(projectID string) (*app, error) {
        ctx := context.Background()
        client, err := firestore.NewClient(ctx, projectID)
        if err != nil {
                log.Fatalf("firestore.NewClient: %v", err)
        }
        store, err := firestoregorilla.New(ctx, client)
        if err != nil {
                log.Fatalf("firestoregorilla.New: %v", err)
        }

        tmpl, err := template.New("Index").Parse(`<body>{{.views}} {{if eq .vie
        if err != nil {
                return nil, fmt.Errorf("template.New: %v", err)
        }

        return &app{
                store: store,
                tmpl:  tmpl,
        }, nil
}
```

3. The index handler gets the user's session, creating one if needed. New sessions are assigned a random language and a view count of 0. Then, the view count is increased by one, the session is saved, and the HTML template writes the response.

getting-started/sessions/main.go
(https://github.com/GoogleCloudPlatform/golang-samples/blob/master/getting-started/sessions/main.go)

GOOGLECLOUDPLATFORM/GOLANG-SAMPLES/BLOB/MASTER/GETTING-STARTED/SESSIONS/MAIN.GO)

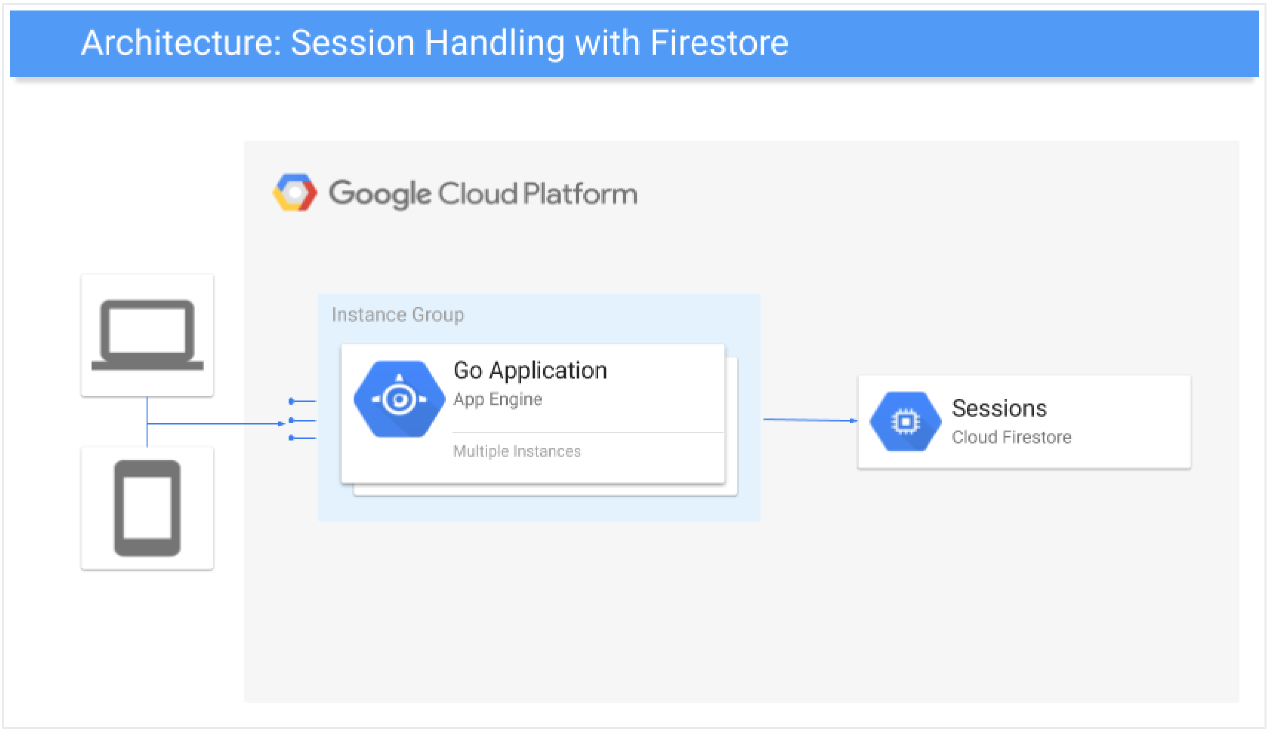
```go
// index uses sessions to assign users a random greeting and keep track of
// views.
func (a *app) index(w http.ResponseWriter, r *http.Request) {
        if r.RequestURI != "/" {
                return
        }

        // name is a non-empty identifier for this app's sessions. Set it to
        // something descriptive for your app. It is used as the Firestore
        // collection name that stores the sessions.
        name := "hello-views"
        session, err := a.store.Get(r, name)
        if err != nil {
                // Could not get the session. Log an error and continue, saving
                // session.
                log.Printf("store.Get: %v", err)
        }

        if session.IsNew {
                // firestoregorilla uses JSON, which unmarshals numbers as floa
                session.Values["views"] = float64(0)
                session.Values["greeting"] = greetings[rand.Intn(len(greetings)
        }
        session.Values["views"] = session.Values["views"].(float64) + 1
        if err := session.Save(r, w); err != nil {
                log.Printf("Save: %v", err)
                // Don't return early so the user still gets a response.
        }

        if err := a.tmpl.Execute(w, session.Values); err != nil {
                log.Printf("Execute: %v", err)
        }
}
```

The following diagram illustrates how Firestore handles sessions for the App Engine app.



Note: Firestore is a persistent, distributed, transactional database. Often, it's more appropriate to choose a different storage solution for sessions such as Memcache (https://redislabs.com/lp/memcached-cloud/) or Redis (https://redislabs.com), whose designs might result in faster operation in this use case.

Deleting sessions

`firestoregorilla` (https://github.com/GoogleCloudPlatform/firestore-gorilla-sessions) doesn't delete old or expired sessions. You can delete session data (https://cloud.google.com/firestore/docs/using-console#delete_data) in the Google Cloud Console (https://console.cloud.google.com/firestore/) or implement an automated deletion strategy. If you use storage solutions for sessions such as Memcache or Redis, expired sessions are automatically deleted.

Running locally

1. In your terminal window, build the `sessions` binary:

```
go build
```

2. Start the HTTP server:

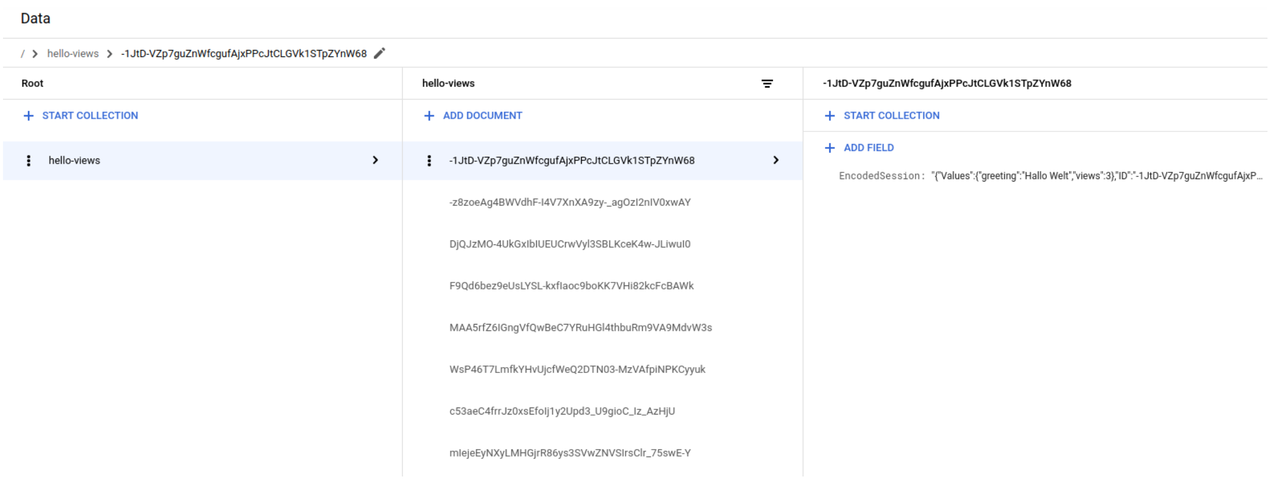
```
./sessions
```

3. View the app in your web browser:

CLOUD SHELL LOCAL MACHINE

In the Cloud Shell toolbar, click **Web preview** and select **Preview on port 8080**.

You see one of five greetings: “Hello World”, “Hallo Welt”, "Hola mundo”, “Salut le Monde”, or “Ciao Mondo.” The language changes if you open the page in a different browser or in incognito mode. You can see and edit the session data in the Google Cloud Console (https://console.cloud.google.com/firestore/).



hello-views

-1JtD-VZp7guZnWfcgufAjxPPcJtCLGVk1STpZYnW68
-z8zoeAg4BWVdhF-I4V7XnXA9zy-_agOzI2nIV0xwAY
DjQJzMO-4UkGxIbIUEUCrwVyl3SBLKceK4w-JLiwul0
F9Qd6bez9eUsLYSL-kxflaoc9boKK7VHi82kcFcBAWk
MAA5rfZ6IGngVfQwBeC7YRuHGl4thbuRm9VA9MdvW3s
WsP46T7LmfkYHvUjcfWeQ2DTN03-MzVAfpiNPKCyyuk
c53aeC4frrJz0xsEfolj1y2Upd3_U9gioC_Iz_AzHjU
mlejeEyNXyLMHGjrR86ys3SVwZNVSlrsClr_75swE-Y

-1JtD-VZp7guZnWfcgufAjxPPcJtCLGVk1STpZYnW68

EncodedSession: "{"Values":{"greeting":"Hallo Welt","views":3},"ID":"-1JtD-VZp7guZnWfcgufAjxP...

4. To stop the HTTP server, in your terminal window, press `Control+C`.

Deploying and running on App Engine

You can use the App Engine standard environment (https://cloud.google.com/appengine/docs/standard/) to build and deploy an app that runs reliably under heavy load and with large amounts of data.

This tutorial uses the App Engine standard environment to deploy the server.

The `app.yaml` file contains the App Engine standard environment configuration:

getting-started/sessions/app.yaml (https://github.com/GoogleCloudPlatform/golang-samples/blob/master/getting-started/sessions/app.yaml)

OOGLECLOUDPLATFORM/GOLANG-SAMPLES/BLOB/MASTER/GETTING-STARTED/SESSIONS/APP.YAML)

```
runtime: go112
```

1. Deploy the app on App Engine:

   ```
   gcloud app deploy
   ```

2. View the live app at the following URL, where ***PROJECT_ID*** is your Google Cloud project ID:

   ```
   https://PROJECT_ID.appspot.com
   ```

The greeting is now delivered by a web server running on an App Engine instance.

Debugging the app

If you cannot connect to your App Engine app, check the following:

1. Check that the `gcloud` deploy commands successfully completed and didn't output any errors. If there were errors (for example, `message=Build failed`), fix them, and try deploying the App Engine app (#deploying_the_web_app) again.
2. In the Cloud Console, go to the **Logs Viewer** page.

   GO TO LOGS VIEWER PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/LOGS/VIEWER)

   a. In the **Recently selected resources** drop-down list, click **App Engine Application**, and then click **All module_id**. You see a list of requests from when you visited your app. If you don't see a list of requests, confirm you selected **All module_id** from the drop-down list. If you see error messages printed to the Cloud Console, check that your app's code matches the code in the section about writing the web app (#writing_the_web_app).

   b. Make sure that the Firestore API is enabled.

Cleaning up

Delete the project

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Cloud Console, go to the **Manage resources** page.

   GO TO THE MANAGE RESOURCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRO

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Delete the App Engine instance

1. In the Cloud Console, go to the **Versions** page for App Engine.

   GO TO THE VERSIONS PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APPENGINE/VERSIONS)

2. Select the checkbox for the non-default app version you want to delete.

Note: The only way you can delete the default version of your App Engine app is by deleting your project. However, you can stop the default version in the Cloud Console (https://console.cloud.google.com/appengine/versions). This action shuts down all instances associated with the version. You can restart these instances later if needed.

In the App Engine standard environment, you can stop the default version only if your app has manual or basic scaling.

3. Click **Delete** 🗑 to delete the app version.

What's next

- Try Cloud Functions tutorials (https://cloud.google.com/functions/docs/tutorials/).
- Learn more about App Engine (https://cloud.google.com/appengine/docs/).
- Try Cloud Run (https://cloud.google.com/run/docs/quickstarts/prebuilt-deploy), which lets you run stateless containers on a fully managed environment or in your own Google Kubernetes Engine cluster.