Sending Mails with SMTP

This lesson explains how Go code can be used to send an email using an SMTP server.

WE'LL COVER THE FOLLOWING

Overview of SMTP in Go

The package `net/smtp` implements the *Simple Mail Transfer Protocol* for sending mail. It contains a `Client` type that represents a client connection to an SMTP server:

- `Dial` returns a new `Client` connected to an SMTP server.
- Set `Mail` (=from) and `Rcpt` (= to)
- `Data` returns a writer that can be used to write the data, here with `buf.WriteTo(wc)`.

Explanation

```
package main
import (
"bytes"
"log"
"net/smtp"
)

func main() {
    // Connect to the remote SMTP server.
    client, err := smtp.Dial("smtp.gmail.com:587")
    if err != nil {
        log.Fatal(err)
    }
    // Set the sender and recipient
```

```
    client.Mail("sender email address")
    client.Rcpt("receiver email address")
    // Send the email body.
    wc, err := client.Data()
    if err != nil {
        log.Fatal(err)
    }
    defer wc.Close()
    buf := bytes.NewBufferString("This is the email body.")
    if _, err = buf.WriteTo(wc); err != nil {
        log.Fatal(err)
    }
}
```

SMTP Server

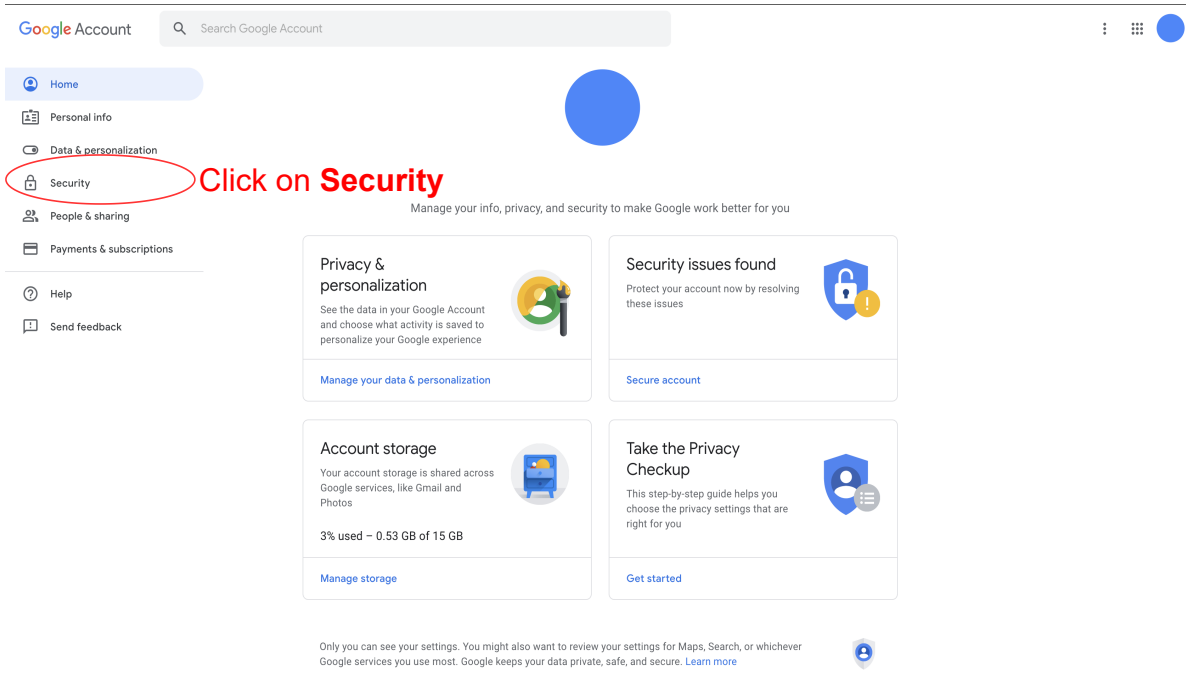
In the code above, we need the package `net/smtp`, which is imported at **line 5**. First, we need to connect to an active remote SMTP server, which is done with the `Dial` method at **line 10**, creating a client instance. Error-handling (from **line 11** to **line 13**) exits the program on error.

We set up the *sender* and *receiver* email address at **line 15** and **line 16**, respectively. **Line 18** constructs the `Data` method on `client`, which makes a client writer `wc` with similar error-handling from **line 19** to **line 21**. At **line 22**, we make sure that the writer `wc` will be closed. Then, at **line 23**, we make a buffered string and write it to `wc` at **line 24**. This if-statement is combined with error-handling, logging an eventual error and exiting the program at **line 25**.

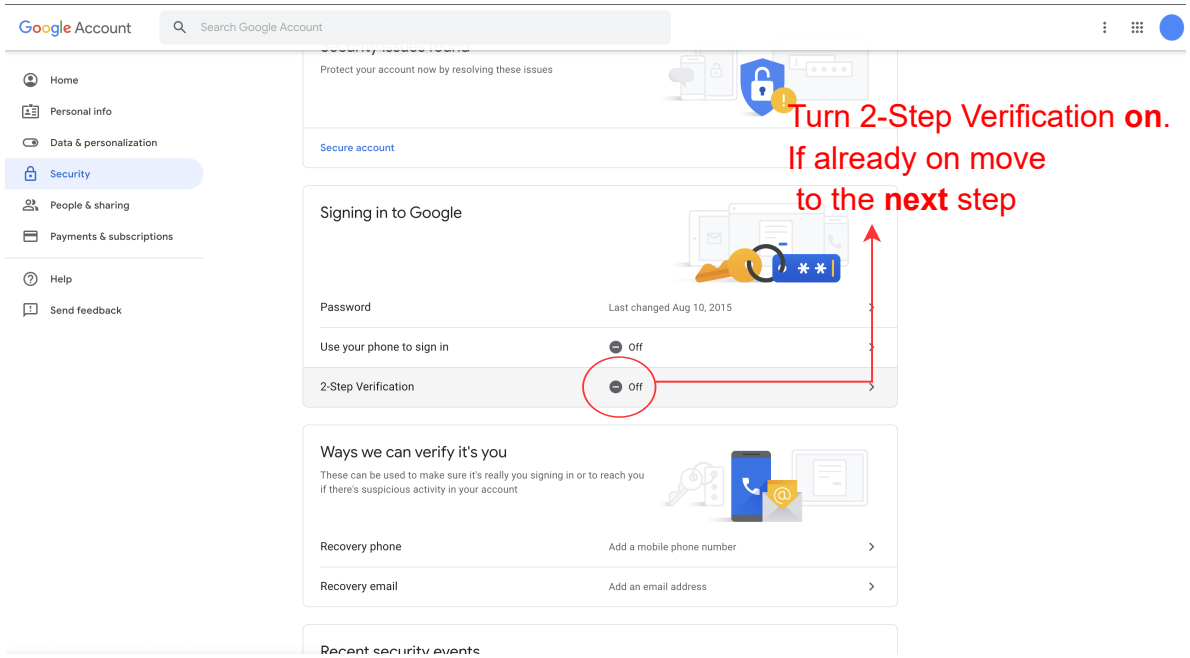
The function `SendMail` can be used if authentication is needed and when you have a number of recipients. It connects to the server at `addr`, switches to TLS (Transport Layer Security encryption and authentication protocol), authenticates with the mechanism if possible, and then sends an email from *address* `from` to *addresses* `to`, with the message *msg*:

```
func SendMail(addr string, a Auth, from string, to []string, msg []byte) e
rror
```

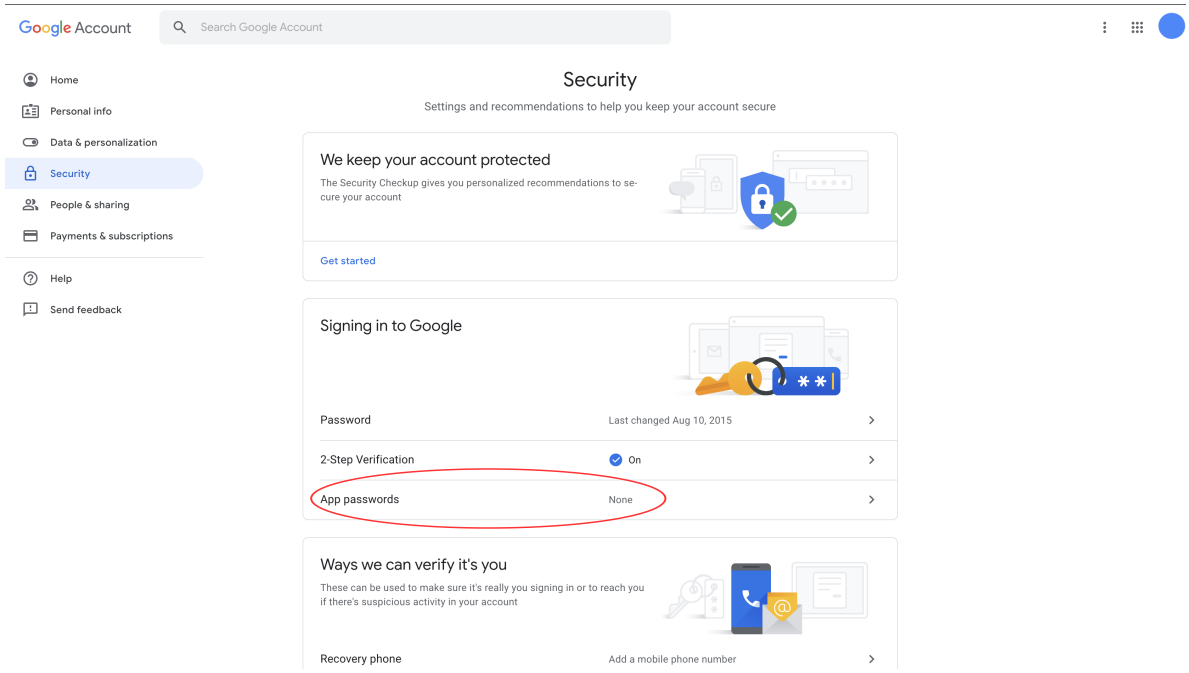
Go through the following illustrations that explain how to set a Gmail account for sending an email before running a program.



Step 1: Open your Account Settings



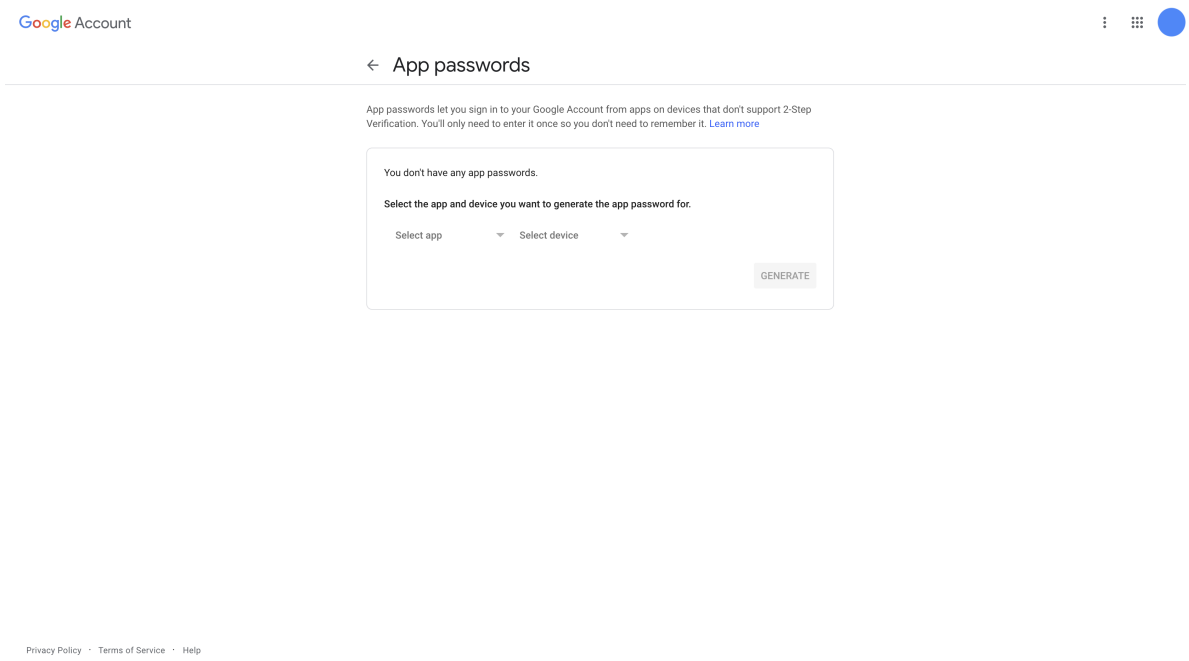
Step 2: Enable 2-Step Verification



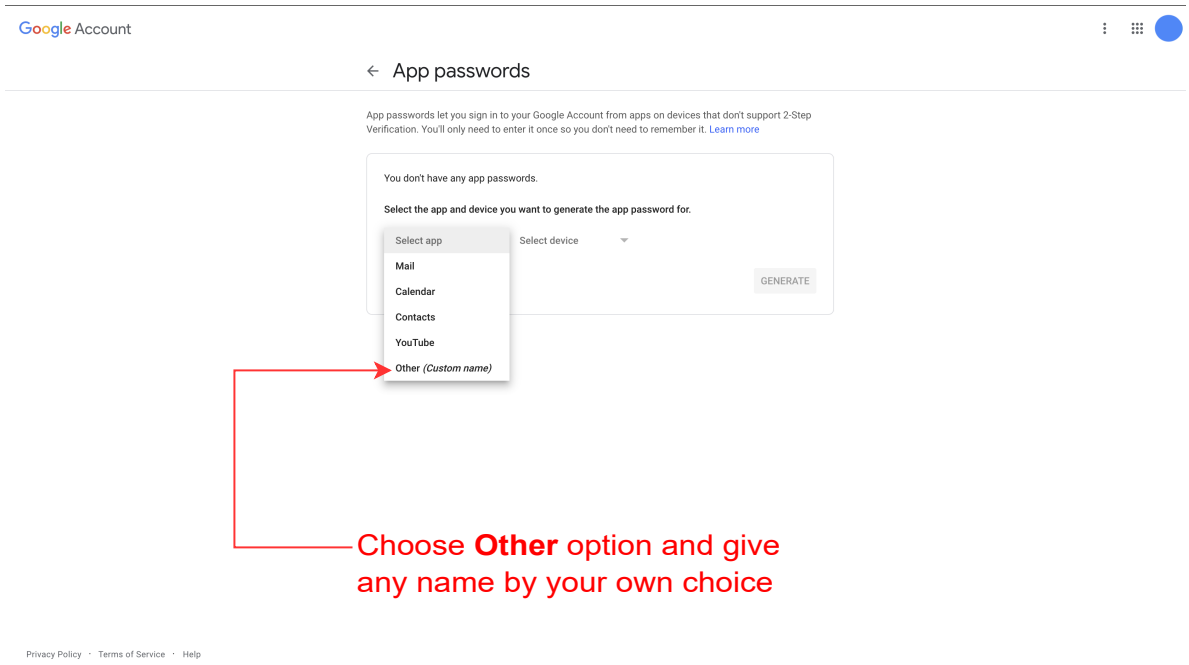
Click **App passwords** to generate a password
If already generated then use the generated password.
If you don't remember it make a new one.

Step 3: Generate an App Password

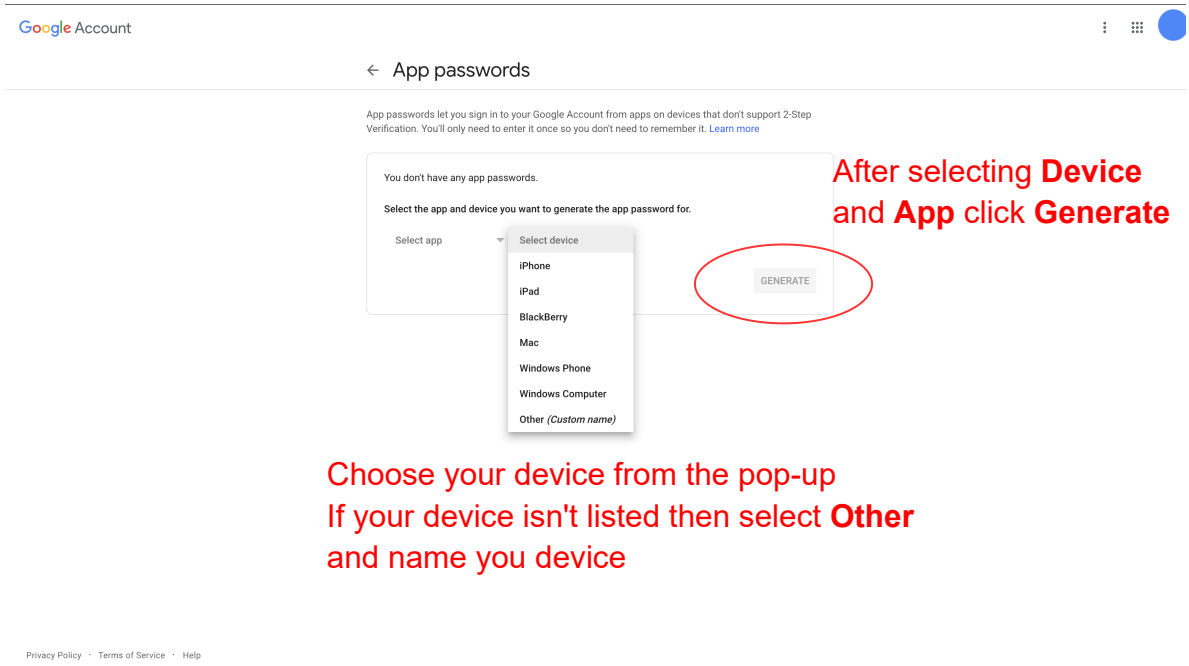
After the verification, the following page will open



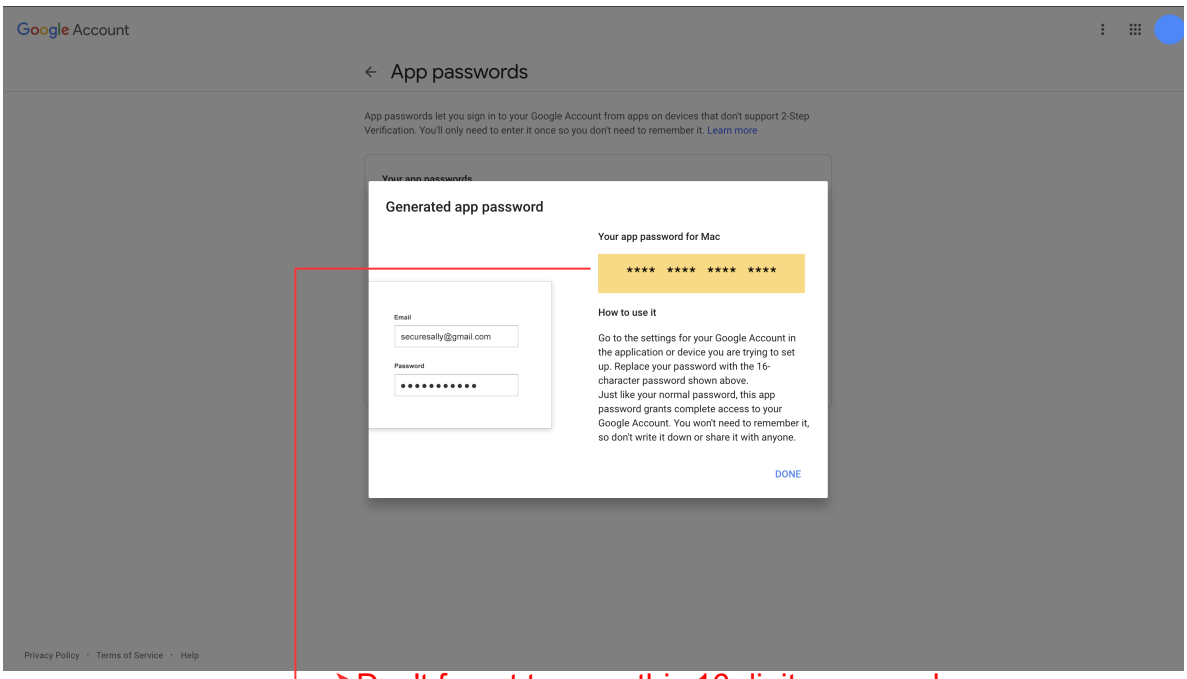
Step3: Generate App Password



Step3: Generate App Password



Step3: Generate App Password



Don't forget to copy this 16 digit password.
You can use this password in the program
to send email rather than using original password.

Step 4: Copy the Password

Look at the following program to see how it works:

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package main
import (
    "fmt"
    "net/smtp"
)
// smtpServer data to smtp server
type smtpServer struct {
 host string
 port string
}
// serverName URI to smtp server
func (s *smtpServer) serverName() string {
 return s.host + ":" + s.port
```

```
}

func main() {

    // Sender data.
    from := "sender email address"
    password := "password" // you can enter original password or password generated with App
    // Receiver email address.
    to := []string{
        "first receiver email address",
        //"second receiver email address",
    }
    // smtp server configuration.
    smtpServer := smtpServer{host: "smtp.gmail.com", port: "587"}
    // Message.
    message := []byte("Enter the message you want to send.")
    // Authentication.
    auth := smtp.PlainAuth("", from, password, smtpServer.host)
    // Sending email.
    err := smtp.SendMail("smtp.gmail.com:587", auth, from, to, message)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("Email Sent!")
}
```

Click the **RUN** button and wait for the terminal to start. Type `go run main.go` in the terminal and press **ENTER**.

Hurrah! You just sent your first email with Go! Cloud computing is so popular nowadays, and Go provides support for it. See the next lesson to see how Golang has made this possible.