Golang HAN 2023 hands-on workshop

Dear HAN Student,

You reached the manual for the Golang hands-on workshop. In this document you will be guided to complete an exercise to write a simple TCP networking server and handle multiple TCP clients.

It can be used as a starting ground for connecting an embedded system over TCP to a custom TCP-server as a backend. There are many networking protocols like UDP/TCP/HTTP/Websockets... but modern IoT embedded systems can simply use TCP as a transport layer. And has been chosen because it is easy to understand (and it is reliable compared to UDP).

For completeness of this document the presentation slides can be found at xor-gate.org/talks/golang-han-2023

Table of contents

Needed software (Toolchain/IDE)

You don't need to install anything on your computer and can directly program and run your Go application from a web browser. Google chrome is preferred and has been tested.

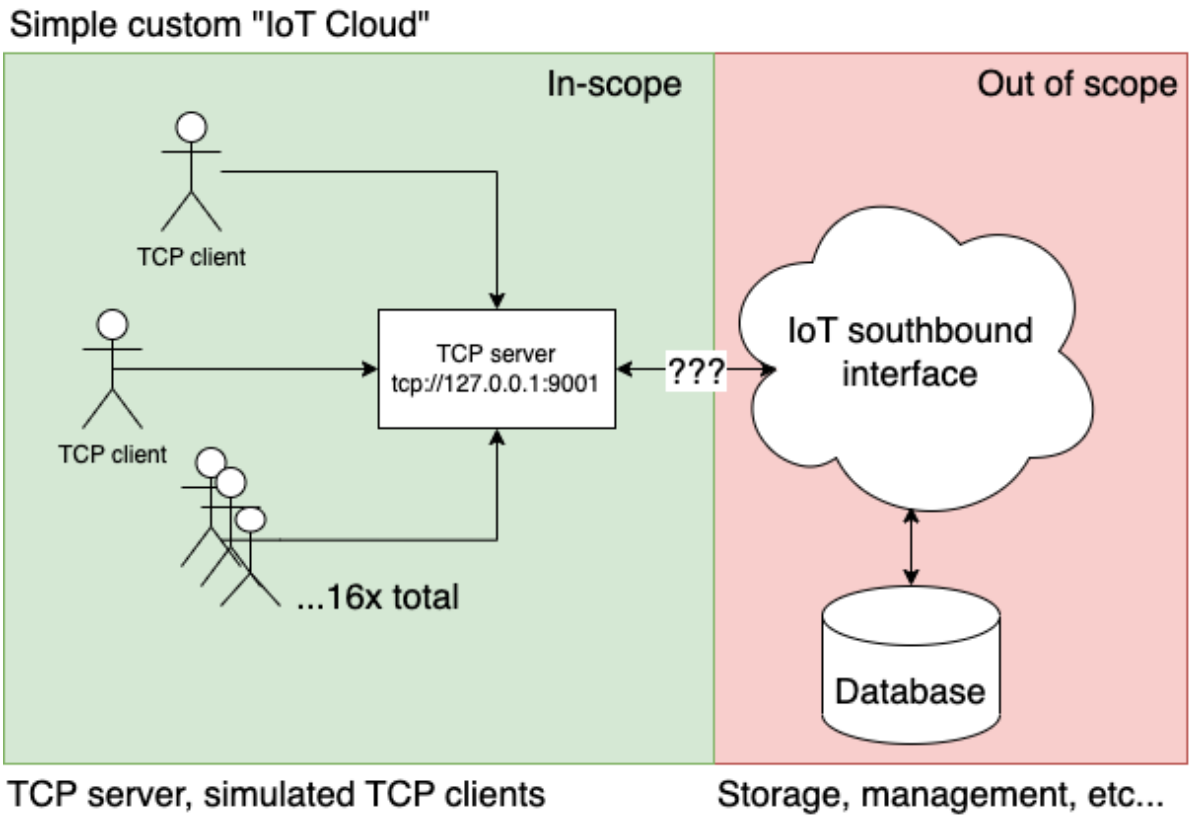
We will use the Golang playground located at goplay.tools as it allows networking using the Golang 'net' package. Normally you can better use the official playground located at go.dev/play

The exercise code can be found here: goplay.tools/snippet/5BUMGC80JGR
And here (for direct download): xor-gate.org/talks/golang-han-2023/exercise.go

The exercise (and code)

The exercise is simple so you can (hopefully) complete it within the workshop hands-on time of 30 minutes. You should work in a group of two people. This is without any prior Golang knowledge but some programming experience is required.

Below is a diagram of our custom "IoT Cloud".



Because we use concurrency (a form of multithreading) which is part of the Golang language we can run the TCP server and simulate multiple TCP clients connecting to it "at the same time".

Exercise step 1 - Importing and using packages

Background information

Golang uses “packages” to modularize code. It is used in the form of the “import” statement on top of each go file. Go is unique because the imports are URLs to HTTP locations. Except for the standard library. Every file must declare its package name on top with “package” (not the full URL). Like so:

```
package main // the main function and its helpers are declared in this package

// Multiple imports in the same statement
import (
        “os” // import standard operating system package
        “github.com/xor-gate/debpkg” // import Debian Linux packaging package
)
```

When using public bits and pieces from packages this is automatically the prefix (but the “namespace” can be renamed when needed). For example creating a temporary directory with the os package is as easy as:

```
tmpdir := os.TempDir()
// do something with the temporary directory
```

What you must do

Figure out which packages are used throughout the exercise code and import them.
Hint: create a list of <**prefix**>.<thingies> and write them down in the multi import statement.

What you should see

The code now compiles and instead of errors:

```
Error
prog.go:12:2: malformed import path "etc...": trailing dot in path element
prog.go:10:2: package package1 is not in GOROOT (/usr/local/go-faketime/src/package1)
prog.go:11:2: package package2 is not in GOROOT (/usr/local/go-faketime/src/package2)
```

Exercise step 2 - Goroutines

Background information

We will use goroutines with the Golang keyword: go to run the TCP server and simulated TCP client functions onto multiple processors (or in a CPU processes queue). In C/C++ managing threads is hard to get right, but in Golang it is a breeze to do multiple things at the same time.

Running a lambda function in the background sleeping for 1 second and then printing message:

```
go func(msg string) {
    time.Sleep(time.Second)
    fmt.Println(msg)
}("sleeping in the background...")
```

What you must do

1. Run the server_listen_and_serve function as a goroutine
2. Run the client function as a goroutine

What you should see

Instead of error:

```
fatal error: all goroutines are asleep - deadlock!

goroutine 1 [IO wait]:
internal/poll.runtime_pollWait(0x7f03d0e95e88, 0x72)
```

At least:

```
[server] Waiting for TCP clients at 127.0.0.1:9001
```

Exercise step 3 - Sending disconnect message from server to client

Background information

As a request-response messaging example over the TCP stream the “protocol” is plain text and every message is delimited with a newline.

1. When the TCP client has successfully connected it sends a hello message with ids ID to the server
2. The server then immediately sends the disconnect message

3. The client receives messages by reading on the connection and splitting on newlines. When “DISCONNECT\n” is read it closes the connection. And signals the main function (goroutine) with a message of the disconnected client

What you must do

In the serve_single_client function write the DISCONNECT message from the server connection to the client.
Hint: use a byte slice and use the connection write function

What you should see

At first you should see a program panic (stopping) because no client disconnected (within deadline), scroll down to the bottom:

```
panic: [GAME OVER] Clients not disconnected within 5 seconds!

goroutine 1 [running]:
main.main()
        /tmp/sandbox609050987/prog.go:69 +0x271
```

The application should output some log messaging like (note the LEVEL COMPLETED):

```
2009/11/10 23:00:00 [server] Waiting for TCP clients to connect at 127.0.0.1:9001
[T+0000ms]
2009/11/10 23:00:03 [client] Started client with ID 1
2009/11/10 23:00:03 [server] TCP client connected
2009/11/10 23:00:03 [client] Started client with ID 2
2009/11/10 23:00:03 [server] Received client message: id=2

2009/11/10 23:00:03 [server] TCP client connected
2009/11/10 23:00:03 [server] Received client message: id=1

[T+3001ms]
2009/11/10 23:00:03 [client] Started client with ID 3
2009/11/10 23:00:03 [server] TCP client connected
2009/11/10 23:00:03 [server] TCP client connected
2009/11/10 23:00:03 [client] Started client with ID 4
2009/11/10 23:00:03 [server] Received client message: id=3

2009/11/10 23:00:03 [server] Received client message: id=4

[T+0002ms]
~~~ REDUCED LOG ~~~
2009/11/10 23:00:03 All clients are started in parallel!
2009/11/10 23:00:03 [client] id=2 got "DISCONNECT" msg from server
2009/11/10 23:00:03 [client] id=1 got "DISCONNECT" msg from server
2009/11/10 23:00:03 [client] id=4 got "DISCONNECT" msg from server
~~~ REDUCED LOG ~~~
2009/11/10 23:00:03 [LEVEL COMPLETED!] All 16 TCP clients disconnected
```

Going further with Golang

Hopefully you made it to this end. We only scratched the surface of Golang. When you are interested there is a lot online (for free) which can make you a master of the Go programming language. Have fun!

Official information

- A tour of Go (`go.dev/tour`)
- The Go Blog (`go.dev/blog`)
- Go talks slides (`go.dev/talks`)
- Public Go packages documentation and indexer (`pkg.go.dev`)

Other sites

- Go by example “snippets” (`gobyexample.com`)
- Yourbasic.com Golang (`yourbasic.org/golang`)
- Golang Weekly Newsletter (`golangweekly.com`)

Youtube channels

- Gopher Academy
- justforfunc: Programming in Go