Published in Analytics Vidhya

Rene Manqueros Follow
Apr 15, 2021 · 6 min read · Listen

Running Go code from Python

Python is a great language that can be picked up easily by a new developer and be productive in a short amount time, it is clean code by design and it has a million libraries available, sadly all of these benefits come at a price called “speed”, python is slow, very slow; pypy or stackless sometimes help with performance but at times they break other libraries that are not compatible or they don’t show any improvement on our specific use case.

You can always take the path of building a library in C and importing it, but to some C might end up being a very complex language to work with or the time required to build something in C could negate the benefits of porting the code.

Fortunately there is a middle ground we can use: Go, easier than C (and WAY better if you do parallelism) but faster than Python, it also has a lot of libraries already available that will provide us with a lot of features that might not be available in C.

Go has a package called CGO (https://golang.org/cmd/cgo/) which allow us to connect Go to C and vice versa, since Python can import libraries in C, we can export certain Go functions to be used by Python as if they were a C library (kinda hacky, kinda neat).

Let’s start by creating a *library.go* file and adding the following:

```
)

//export helloWorld
func helloWorld(){
   log.Println("Hello World")
}

func main(){

}
```

CGO is added with the “C” import, that will provide us with certain utility function and it will make the compiler treat certain comments as special actions, the first one will be the “export” comment which will tell the compiler which functions will be exported to be called from Python, we then create our function as we regularly do and then just to be able to compile, we’ll add an empty main function at the end.

Pay **extra** attention to the *export* being right next to the comments, if you add a space in between it will not be considered an exported symbol and python won’t find it.

On the CLI we’ll now perform the compilation by running:

```
go build -buildmode=c-shared -o library.so library.go
```

Key things to note here will be the build mode, which we’ll specify that it’s going to be a C shared object and then the output extension will be a library (.so = shared object for me since I am using Linux), once we compile the library we’ll proceed to building the client in python, for which I’ve created an “app.py” with the following:

```
import ctypes
library = ctypes.cdll.LoadLibrary('./library.so')
hello_world = library.helloWorld
```


the library's *helloWorld* function, in this case the naming convention will hint us which one is the Go version and which one is Python's, finally on line 4 we'll just invoke the function and receive an output similar to the following:

```
>>> import ctypes
>>> library = ctypes.cdll.LoadLibrary('./library.so')
>>> hello_world = library.helloWorld
>>> hello_world()
2021/04/14 21:49:33 Hello World
```

Go's log format is used to say hello

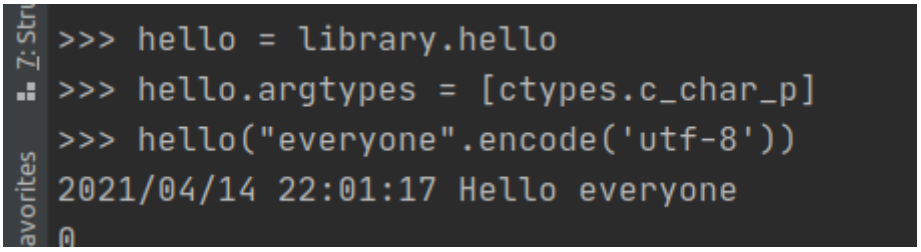
If we want to send a parameter from Python, we need to make a few adjustments, for which we'll create a new function on our library.go file:

```
//export hello
func hello(namePtr *C.char){
   name := C.GoString(namePtr)
   log.Println("Hello", name)
}
```

First thing you'll notice is the argument, which is a a pointer to a C char, this means that we'll be receiving a "string" as a parameter, because of how C handles strings (array's of characters) we'll need to make a tiny adjustment before we can use it as we expect to on Go, for that we have a handy method called C.GoString, which will receive that pointer to a C.char and return us the value of the string.

Once we have that, we'll switch over to python and create a new symbol for this function:

```
hello = library.hello
hello.argtypes = [ctypes.c_char_p]
hello("everyone".encode('utf-8'))
```

Up to this point we can do invocation from Python and send data from Python, we'll need one extra thing which is going to be sending data back from Go to Python.

For that we'll create a new Go func:

```
//export farewell
func farewell() *C.char{
   return C.CString("Bye!")
}
```

This time we'll be return a pointer to a C.char and the string will need to be converted from string to CString via the C package.

On the python side of things:

```
farewell = library.farewell
farewell.restype = ctypes.c_void_p

# this is a pointer to our string
farewell_output = farewell()

# we dereference the pointer to a byte array
farewell_bytes = ctypes.string_at(farewell_output)

# convert our byte array to a string
farewell_string = farewell_bytes.decode('utf-8')

print(farewell_output, farewell_bytes, farewell_string)
```

```
38113120 b'Bye!' Bye!
```

The first two lines are via Go and the last one via Python

We can then expand to something a bit more complex and use JSON as our way of sending data between the two worlds, on library.go we add another functon:

```
//export fromJSON
func fromJSON(documentPtr *C.char){
   documentString := C.GoString(documentPtr)
   var jsonDocument map[string]interface{}
   err := json.Unmarshal([]byte(documentString), &jsonDocument)
   if err != nil{
      log.Fatal(err)
   }
   log.Println(jsonDocument)
}
```

This one will receive a string the same way as before but we'll treat it as a JSON string and load it to a map of string-interface (could be anything really...).

Then on python:

```
import json
from_json = library.fromJSON
from_json.argtypes = [ctypes.c_char_p]
document = {
    "name": "john",
    "last_name": "smith"
}
from_json(json.dumps(document).encode('utf-8'))
```

We dump a dictionary to a JSON string and send it to our new function, the output of this will be:

```
33451872 b'Bye!' Bye!
2021/04/14 22:15:20 map[last_name:smith name:john]
```

In case something got lost in the copy/pasting of the examples, the full library.go file:

```
package main

import (
   "C"
   "encoding/json"
   "log"
)

//export helloWorld
func helloWorld(){
   log.Println("Hello World")
}

//export hello
func hello(namePtr *C.char){
   name := C.GoString(namePtr)
   log.Println("Hello", name)
}

//export farewell
func farewell() *C.char{
   return C.CString("Bye!")
}

//export fromJSON
func fromJSON(documentPtr *C.char){
   documentString := C.GoString(documentPtr)
   var jsonDocument map[string]interface{}
   err := json.Unmarshal([]byte(documentString), &jsonDocument)
   if err != nil{
      log.Fatal(err)
   }
   log.Println(jsonDocument)
}

func main(){

}
```

The app.py file:

```
hello_world = library.helloWorld
hello_world()

hello = library.hello
hello.argtypes = [ctypes.c_char_p]
hello("everyone".encode('utf-8'))

farewell = library.farewell
farewell.restype = ctypes.c_void_p

# this is a pointer to our string
farewell_output = farewell()

# we dereference the pointer to a byte array
farewell_bytes = ctypes.string_at(farewell_output)

# convert our byte array to a string
farewell_string = farewell_bytes.decode('utf-8')
print(farewell_output, farewell_bytes, farewell_string)


import json
from_json = library.fromJSON
from_json.argtypes = [ctypes.c_char_p]
document = {
    "name": "john",
    "last_name": "smith"
}
from_json(json.dumps(document).encode('utf-8'))
```

And my Makefile:

```
build:
   go build -buildmode=c-shared -o library.so library.go
```