Go Playground

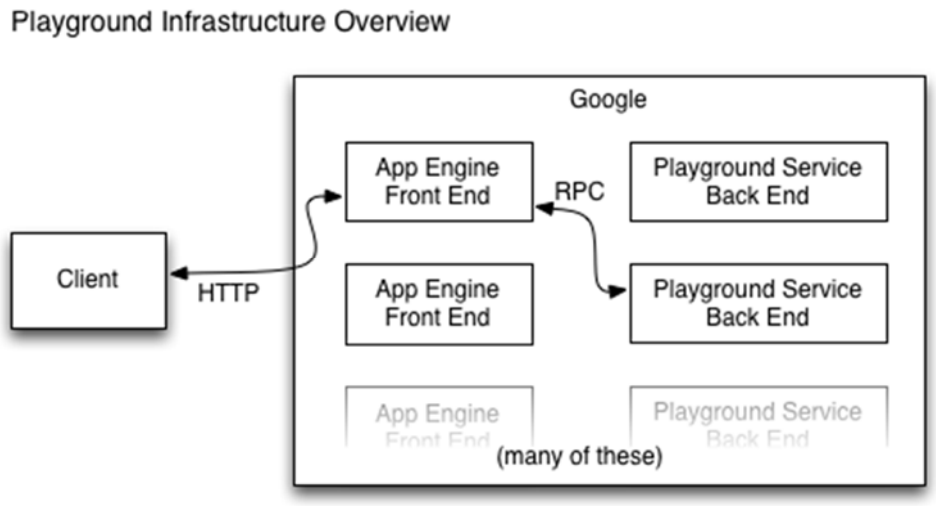
Reference:
https://blog.golang.org/introducing-go-playground
https://blog.golang.org/playground

1. About Go Playground

The Playground allows anyone with a web browser to write Go code that we immediately compile, link, and run on our servers.

The programs build and run in a sandbox with a reduced standard library; the only communication our program has to the outside world is via standard output, and there are limits to CPU and memory use.

2. Go Playground Infrastructure



The playground service has three parts:

- A back end that runs on Google's servers. It receives RPC requests, compiles the user program using the gc tool chain, executes the user program, and returns the program output (or compilation errors) as the RPC response.
  - To isolate user programs from Google's infrastructure, the back end runs them under Native Client (or "NaCl"), a technology developed by Google to permit the safe execution of x86 programs inside web browsers. The back end uses a special version of

the gc tool chain that generates NaCl executables.
    - NaCl limits the amount of CPU and RAM a program may consume, and it prevents programs from accessing the network or file system.

- A front end that runs on Google App Engine. It receives HTTP requests from the client and makes corresponding RPC requests to the back end. It also does some caching.
    - The front end serves an HTTP handler at https://golang.org/compile. The handler expects a POST request with a body field (the Go program to run) and an optional version field
    - When the front end receives a compilation request it first checks memcache to see if it has cached the results of a previous compilation of that source. If found, it returns the cached response. The cache prevents popular programs such as those on the Go home page from overloading the back ends. If there is no cached response, the front end makes an RPC request to the back end, stores the response in memcache, parses the playback events, and returns a JSON object to the client as the HTTP response

- A JavaScript client that implements the user interface and makes HTTP requests to the front end.

3. Faking the file system

Programs built with the Go's NaCl tool chain cannot access the local machine's file system. Instead, the syscall package's file-related functions (Open, Read, Write, and so on) operate on an in-memory file system that is implemented by the syscall package itself. Since package syscall is the interface between the Go code and the operating system kernel, user programs see the file system exactly the same way as they would a real one.

4. Faking the network

The playground's network stack is an in-process fake implemented by the syscall package. It permits playground projects to use the loopback interface (127.0.0.1). Requests to other hosts will fail.