SSH with Go

GoSF Meetup
25 August 2016

Chris Roche
Software Engineer, Lyft

Who am I?

Core Services Team at Lyft

- Libraries

Previously, Core Platform & DevOps at VSCO

- Services
- Libraries
- Deployment/Infrastructure Tools

Why not just `ssh example.com`?

Because golang.org/x/crypto/ssh gives you:

- Cross platform code
- Testability
- Better error handling
- More capabilities
- Ergonomics:

Either:

```
$ ssh -o ProxyCommand='ssh proxy.example.com nc example.com 22' example.com
```

Or:

```
$ sshThru proxy.example.com example.com
```

Opening A Connection

```
func Connect(host string, methods ...ssh.AuthMethod) (*ssh.Client, error) {
    cfg := ssh.ClientConfig{
        User: "chris",
        Auth: methods,
    }

    return ssh.Dial("tcp", host, &cfg)
}
```

- Can also specify timeouts, host checks, & more SSH goodies
- Each **AuthMethod** is attempted in order
- Handful of types:

```
ssh.Password             // static secret
ssh.PasswordCallback     // ask the user
ssh.KeyboardInteractive  // server-provided prompts
ssh.RetryableAuthMethod  // decorator for above
ssh.PublicKeys           // key pairs
ssh.PublicKeysCallback   // SSH-Agent
```

Authentication Methods

```go
func KeyPair(keyFile string) (ssh.AuthMethod, error) {
    pem, err := ioutil.ReadFile(keyFile)
    if err != nil {
        return nil, err
    }

    key, err := ssh.ParsePrivateKey(pem)
    if err != nil {
        return nil, err
    }

    return ssh.PublicKeys(key), nil
}
```

```go
func SSHAgent() (ssh.AuthMethod, error) {
    agentSock, err := net.Dial("unix", os.Getenv("SSH_AUTH_SOCK"))
    if err != nil {
        return nil, err
    }

    return ssh.PublicKeysCallback(agent.NewClient(agentSock).Signers), nil
}
```

Auth + Connect

```
agent, err := SSHAgent()
// handle error

keyPair, err := KeyPair("/home/chris/.ssh/id_rsa")
// handle error

client, err := Connect("example.com:22", agent, keyPair)
// handle error

defer client.Close()
```

- Don't forget **client.Close()**!
- Need **crypto/x509** if keys are password-protected / PKCS8

Run Command

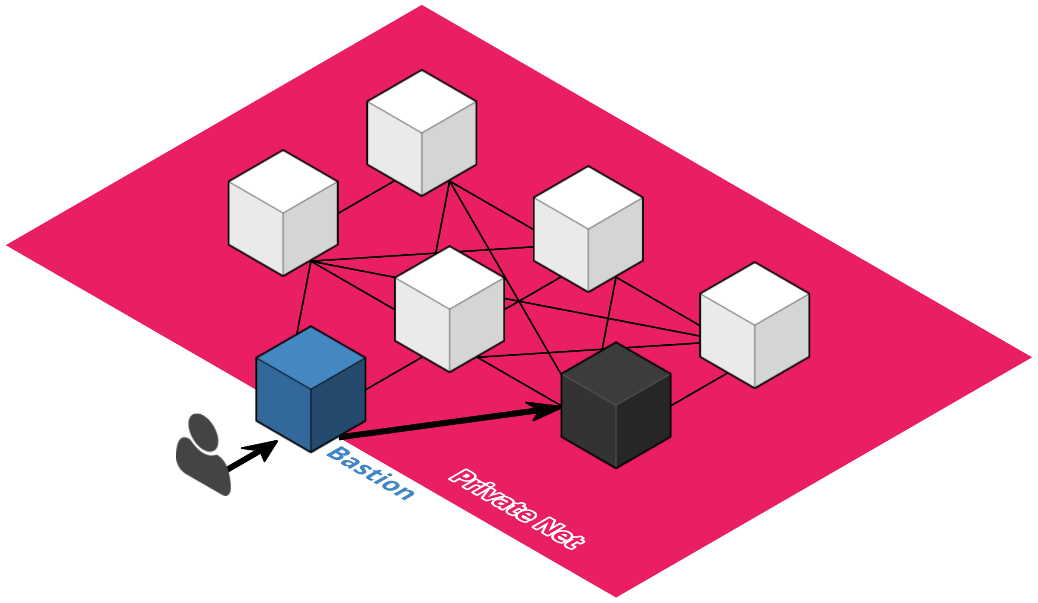
```
sess, err := client.NewSession()
// handle error
defer sess.Close()

sess.Stdout = os.Stdout
sess.Setenv("LS_COLORS", os.Getenv("LS_COLORS"))

err = sess.Run("ls -lah")
// handle error
```

- One command or shell, one **ssh.Session**
- Similar API to **os/exec.Cmd**
- Don't forget **sess.Close()**!

Open Shell

```
sess.Stdin = os.Stdin
sess.Stdout = os.Stdout
sess.Stderr = os.Stderr

modes := ssh.TerminalModes{
    ssh.ECHO:          1,      // please print what I type
    ssh.ECHOCTL:       0,      // please don't print control chars
    ssh.TTY_OP_ISPEED: 115200, // baud in
    ssh.TTY_OP_OSPEED: 115200, // baud out
}

termFD := int(os.Stdin.Fd())

w, h, _ := terminal.GetSize(termFD)

termState, _ := terminal.MakeRaw(termFD)
defer terminal.Restore(termFD, termState)

sess.RequestPty("xterm-256color", h, w, modes)
sess.Shell()
sess.Wait()
```

Proxy Through Bastion



```
func Proxy(bastion *ssh.Client, host string, clientCfg *ssh.ClientConfig) *ssh.Client {
    netConn, _ := bastion.Dial("tcp", host)

    conn, chans, reqs, _ := ssh.NewClientConn(netConn, host, clientCfg)

    return ssh.NewClient(conn, chans, reqs)
}
```

Multiplex Commands

```go
func TailLog(name string, client *ssh.Client, lines chan<- string) {
    sess, _ := client.NewSession()
    defer sess.Close()

    out, _ := sess.StdoutPipe()

    scanner := bufio.NewScanner(out)
    scanner.Split(bufio.ScanLines)

    sess.Start("tail -f /var/log/app.log")

    for scanner.Scan() {
        lines <- fmt.Sprintf("[%s] %s", name, scanner.Text())
    }

    sess.Wait()
}
```

Multiplex Commands

```go
func MultiTail(bastion *ssh.Client, hosts []string, cfg *ssh.ClientConfig) {
    lines := make(chan string)

    for _, remote := range hosts {
        go TailLog(
            remote,
            Proxy(bastion, remote, cfg),
            lines,
        )
    }

    for l := range lines {
        log.Print(l)
    }
}
```

Tunnel

```
func Tunnel(client *ssh.Client, localHost, remoteHost string) {
    listener, _ := net.Listen("tcp", localHost)
    defer listener.Close()

    for {
        localConn, _ := listener.Accept()
        remoteConn, _ := client.Dial("tcp", remoteHost)

        go copy(localConn, remoteConn)
        go copy(remoteConn, localConn)
    }
}
```

Reverse Tunnel / Proxy

```
func ReverseTunnel(client *ssh.Client, remoteHost string) {
    listener, _ := client.Listen("tcp", remoteHost)
    defer listener.Close()

    handler := func(res http.ResponseWriter, req *http.Request) {
        fmt.Fprint(res, "Hello, GoSF!")
    }

    http.Serve(listener, http.HandlerFunc(handler))
}
```

Thank you

Chris Roche
Software Engineer, Lyft

http://rodaine.com (http://rodaine.com)
http://github.com/rodaine (http://github.com/rodaine)
@rodaine (http://twitter.com/rodaine)