Microservices with Go

Golang: Installation & configuration

Installation options

Go can be installed by two ways:

- **Executable installator:** Plug & Play. Everything is configured for you. Perfect when starting and when you want to start working with the language as soon as possible.

- **Binary installation:** Custom installation by binary files. Perfect for configuring production environments and containers. You can switch between different Go versions by just changing an environment variable.

You can find all the options in Go's official site: https://golang.org/dl/

Environment configuration

Whether you use self-contained installator or binary files there are 2 environment variables you need to configure:

GOROOT: Defines where Go binary files are located. Usually points into */usr/local/go*

If you install Go using the installator, this variable it's automatically configured for you.

When working with different projects using different Go versions, the best approach is to install Go by binary files and put them in different folders like this:

- /usr/local/go/1.9
- /usr/local/go/1.10
- /usr/local/go/1.12

And then configure the GOROOT environment variable accordingly by using a function in **.bash_profile**, as follows:

Environment configuration

```
export GOHOME=/usr/local/go

useGo() {
      local goRoot="$GOHOME/$1"
      echo "$goRoot"
      if [ ! -d "$goRoot" ]; then
            echo "Go version $1 is not installed in $GOHOME"
            return
      fi
      export GOROOT="$goRoot"
      go version
}
```

Then you can just switch between different Go versions by doing:

$ useGo 1.12

go version go1.12 darwin/amd64

Environment configuration

GOPATH: The GOPATH environment variable specifies the location of your workspace. It defaults to a directory named *go* inside your home directory, so $HOME/go on Unix, $home/go on Plan 9, and %USERPROFILE%\go (usually C:\Users\YourName\go) on Windows.

If you would like to work in a different location, you will need to set GOPATH to the path to that directory. (Another common setup is to set GOPATH=$HOME.) Note that GOPATH must not be the same path as your GOROOT directory. Actually, **GOPATH should be outside GOROOT directory**.

It basically contains two directories inside:

- **src**: contains Go source files. This is where all of your projects are going to be located.
- **bin**: contains executable commands.

Under *src* you'll find the used repositories (github.com, gitlab.com, etc) followed by the username and then the repository name.

Environment configuration

Example:

Let's suppose that you have the following Go repository: *github.com/federicoleon/go-testing*

In order to download this repository in our computer we have two choices:

1. Running **go get github.com/federicoleon/go-testing** in the console: This will create the following directory structure inside GOPATH:

- src
  - github.com
    - federicoleon
      - go-testing
        - ...

Environment configuration

2. Going to $GOPATH/src/github.com/federicoleon and then cloning the repo there.

If you pay attention, these two approaches generates the same file structure in the GOPATH directory.

Every time you start working on a new project, it should be located inside GOPATH's *src* folder accordingly.

Accessing repositories via SSH

SSH configuration

If you take a look at how we work with dependencies, we work with remote repositories sources like Github, Gitlab, Bitbucket and so on... If we're working with private repositories (both personal private or third parties private ones) we need to configure SSH in order to be able to access those from our computer.

This configuration takes **4 steps**:

1. Tell our repository provider we want to use SSH instead of HTTPS for all the repos.
2. Generate a new SSH key in our computer.
3. Add the public key in our provider (Github, Gitlab, etc).
4. Test the SSH connection.

SSH configuration

1. Tell our repository provider we want to use SSH instead of HTTPS for all the repos.

```
# Github
git config --global url.ssh://git@github.com/.insteadOf https://github.com/


# Bitbucket
git config --global url.ssh://git@bitbucket.org/.insteadOf https://bitbucket.org/
```

SSH configuration

2. Generate a new SSH key in our computer.

```
# Open Terminal.

# Paste the text below, substituting in your GitHub email address.

$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

# This creates a new ssh key, using the provided email as a label.

# Generating public/private rsa key pair.
# When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

# Enter a file in which to save the key (/Users/you/.ssh/id_rsa): [Press enter]
# At the prompt, type a secure passphrase. For more information, see "Working with SSH key passphrases".

Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]
```

SSH configuration

2. Generate a new SSH key in our computer.

```
# Start the ssh-agent in the background.
$ eval "$(ssh-agent -s)"
# Agent pid 12345

# If you're using macOS Sierra 10.12.2 or later, you will need to modify your `~/.ssh/config`file to automatically load keys into the ssh-agent and store passphrases in your keychain.

Host *
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/id_rsa

# Add your SSH private key to the ssh-agent and store your passphrase in the keychain. If you created your key with a different name, or if you are adding an existing key that has a different name, replace  _id_rsa_  in the command with the name of your private key file.

$ ssh-add -K ~/.ssh/id_rsa
```

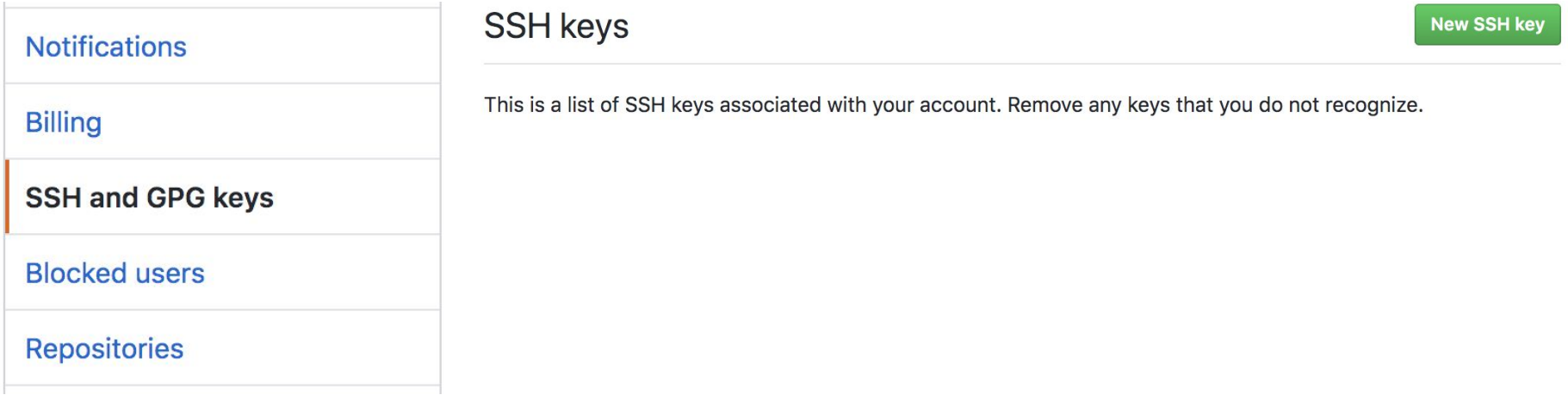
SSH configuration

3. Add the public key in our provider (Github, Gitlab, etc).

```
# Copy the content of your public id_rsa.pub When copying your key, don't add any newlines or whitespace.

$ pbcopy < ~/.ssh/id_rsa.pub

# Paste what you've copied in your (Github, Gitlab, Bitbucket, etc) SSH keys section, usually inside your profile's settings page.
```



SSH configuration

4. Test the SSH connection.

```
$ ssh -T git@github.com

# Or:

$ ssh -T git@bitbucket.com

# You should see something like following:

Hi username! You've successfully authenticated, but GitHub does not provide shell access.

# And you're done 'J'
```