Go workshop

Marco Molteni

2023-10-15

Getting started: before the workshop (1)

Please take the time to follow the instructions in the next slide. It will reduce frustration and will give you an editor that auto fixes problems as they appear (package imports, …) and with code completion, code navigation, intellisense, …

Warning: the majority of Go install guides on the Internet are wrong or old or misunderstand the `go.mod` way of doing things. Following the official links in the next slide is the fastest way to being productive.

Getting started: before the workshop (2)

1. Install the latest Go; do not use the one from your distribution.
2. Do not set any Go environment variable (`$GOPATH`, `$GOBIN`, …).
3. Add two directories to your `$PATH`:
   - The path to the `bin` directory where you installed Go in step 1 (example: `/usr/local/go/bin`).
   - `$HOME/go/bin`. This is where `go install` will put executables.

Getting started: before the workshop (3)

4. Verify that `$PATH` is now set correctly:
   - The shell should find the `go` executable and the version should match what you installed in step 1:

     ```
     $ which go
     /usr/local/go/bin/go
     $ go version
     ```

   - Install the `gotestsum` test runner and verify that you can find it:

     ```
     $ go install gotest.tools/gotestsum@latest
     $ which gotestsum
     $HOME/go/bin/gotestsum
     ```

Getting started: before the workshop (4)

5. Take the time to configure your editor correctly.

Then, follow these two official tutorials:

6. Tutorial: Get started with Go
7. Tutorial: Create a Go module

About this workshop

- Assumes no prior experience with Go.
- Assumes you followed the slides “Before the workshop”.
- Assumes you already know how to program, maybe with a dynamically typed language.
- Is a work in progress, so incomplete.
- Is sometimes opinionated (that is: there could be other equally good ways of doing the same things).

About you

About you

- What are your expectations from the workshop?

On learning something new

On learning something new

- Some people appreciate Go for its simplicity and accept its **defects**. Some don't. It is fine either way.
- Do not try to write in Go as if it were the language you are most familiar with.
- Instead, try to have the humility and patience to learn how to write idiomatic Go.

On what should be *easy*

On what should be *easy*

Easy for me to write or easy for others to read?

On what should be *easy*

Easy for me to write or easy for others to read?

- Readability (by your future self and your team members) is of the utmost importance.
- Being “easy to write” often clashes with readability.
- Go favours readability. Also if it can seem boring.

How?

On what should be *easy*

Easy for me to write or easy for others to read?

- Readability (by your future self and your team members) is of the utmost importance.
- Being "easy to write" often clashes with readability.
- Go favours readability. Also if it can seem boring.

How?

- NO MAGIC.
- No monkey patching.
- No dynamic "changes" to the code.

Error handling

Error handling

- Go has no exceptions, instead functions return multiple values, of which the last one, by convention, is the error.
- **Errors must be handled, immediately, *every* time**.
- This can be tedious at first, but please just *swim with the flow* and accept it.
- Your programs will be more robust and *sincere*: in the real world, errors happen all the time!
- You will also discover that testing code without exceptions is easier and explicit…

Error handling: example

Error handling: example

```go
func enjoy() error {
    flavor, err := iceCream()
    if err != nil {
        return fmt.Errorf("enjoy: %s", err)
    }
    // Use flavor.
    // ...

    // All done, all OK.
    return nil
}

func iceCream() (string, error) {
    ...
}
```

Bubbling up errors

Bubbling up errors

If we follow this approach, then the errors arrive up to the `main()` function:

```
func main() {
  if err := run(); err != nil {
    fmt.Println("error:", err)
    os.Exit(1)
  }
}

func run() error {
   ...
}
```

This is also an answer to “how do I test the main function?” (more details later).

Warmup: an hello word program

`helloworld`: a simplistic program

- Clone the workshop repo: github.com/marco-m/go-workshop
- `cd` to the helloworld directory
- The module structure is the simplest possible.
- Have a look at the README, run the commands explained there.
- Have a look at the code, kick the tires…

`fruits`: a useful template to get started

`fruits`

- github.com/marco-m/go-workshop/fruits
- Contains a **lot** of useful techniques and conventions for writing a command-line program.
- We will have a quick overview, but it is more for you to use as a reference in future projects.

`loadmaster`: a useful command-line program

`loadmaster`: cumulative Concourse build times

Given a pipeline and a time window, calculate the total and average build time per job and display it, sorted per total time.

```
$ ./loadmaster build-time --pipeline=concourse
job                        count  average   total
dev-image                      7  2h45m17s  19h16m57s
resource-types-images         22  14m55s    5h27m59s
unit                           5  59m36s    4h58m2s
bosh-topgun-both               2  2h10m36s  4h21m11s
testflight                     4  49m27s    3h17m48s
bosh-topgun-runtime            2  1h12m36s  2h25m12s
bosh-topgun-core               2  59m18s    1h58m36s
...
```

Background: Concourse pipelines, jobs and builds

Background: Concourse pipelines, jobs and builds

- A pipeline is a directed graph, where each node is a resource or a job.
- The graph can be connected or disconnected.
- A pipeline cannot be triggered!
  - Can only trigger a node (a resource check or a job build).
- Often a pipeline is a tree, and the root of the tree is a git resource.
  - Then triggering the git resource or the first job gives the impression of triggering the pipeline.
- A **build** is one execution of a **job**.

Backgroud: Concourse builds

```
$ fly -t developers builds --count=8

id   name              status     start            end       duration  team   created by
550  p1-mast/j13/123   started    2023-10-04@..  n/a       0s+       cloud  system
549  p2-mast/j1/5      started    2023-10-04@..  n/a       0s+       devs   system
548  p2-feat-5/j7/4    started    2023-10-04@..  n/a       0s+       cloud  system
530  p1-mast/j3/119    succeeded  2023-10-04@..  2023-..   3s        devs   system
529  p3-mast/j3/123    pending    n/a          ..  n/a       n/a       devs   bob@ex.org
525  p4-feat-9/j1/23   started    2023-10-04@..  n/a       1m1s+     cloud  system
519  p9-mast/j9/13     started    2023-10-04@..  n/a       1m31s+    cloud  system
518  p1-mast/j4/3      succeeded  2023-10-04@..  2023-..   25s       cloud  system
```

Question: What are:

- **id**: ?
- **name**: ?

Backgroud: Concourse builds

```
$ fly -t developers builds --count=8

id   name              status     start            end       duration  team   created by
550  p1-mast/j13/123   started    2023-10-04@..  n/a       0s+       cloud  system
549  p2-mast/j1/5      started    2023-10-04@..  n/a       0s+       devs   system
548  p2-feat-5/j7/4    started    2023-10-04@..  n/a       0s+       cloud  system
530  p1-mast/j3/119    succeeded  2023-10-04@..  2023-..   3s        devs   system
529  p3-mast/j3/123    pending    n/a          ..  n/a       n/a       devs   bob@ex.org
525  p4-feat-9/j1/23   started    2023-10-04@..  n/a       1m1s+     cloud  system
519  p9-mast/j9/13     started    2023-10-04@..  n/a       1m31s+    cloud  system
518  p1-mast/j4/3      succeeded  2023-10-04@..  2023-..   25s       cloud  system
```

Question: What are:

- **id**: `global-build-Id`
- **name**: `pipeline / job / relative-build-Id`

What we are going to learn

What we are going to learn

- How to write a simple CLI program
- HTTP client
- JSON parsing
- Testing
- Testing HTTP clients with high fidelity
- …

Back to `loadmaster`

Back to `loadmaster`

Given a pipeline and a time window, calculate the total and average build time per job and display it, sorted per total time.

```
$ ./loadmaster build-time --pipeline=concourse
job                     count  average   total
dev-image                   7  2h45m17s  19h16m57s
resource-types-images      22  14m55s    5h27m59s
unit                        5  59m36s    4h58m2s
bosh-topgun-both            2  2h10m36s  4h21m11s
testflight                  4  49m27s    3h17m48s
bosh-topgun-runtime         2  1h12m36s  2h25m12s
bosh-topgun-core            2  59m18s    1h58m36s
...
```

`loadmaster`: first iteration

```
./bin/loadmaster -h
This program calculates statistics for Concourse
Usage: loadmaster [--server SERVER] [--team TEAM] [--timeout TIMEOUT] <command>
[<args>]

Options:
  --server SERVER        Concourse server URL [default: https://ci.concourse-ci.org]
  --team TEAM            Concourse team [default: main]
  --timeout TIMEOUT      timeout for network operations (eg: 1h32m7s) [default: 5s]
  --version              display version and exit
  --help, -h             display this help and exit

Commands:
  build-time             calculate the cumulative build time taken by a pipeline

For more information visit FIXME https://example.org/...
```

`loadmaster`: getting started

- The exercise does not require authorization (no need to `fly login`).
- In the github.com/marco-m/go-workshop repo:
  - `loadmaster-skel`: the skeleton to use.
  - (`loadmaster`: solution to the exercise)
  - The tests are already there, you can use them as a guide.
- In the github.com/concourse/concourse repo:
  - HTTP server routes: concourse/atc/routes.go
  - Looking at `fly` to understand which endpoint to use: concourse/tree/master/fly/commands

`loadmaster`: let’s do it!

- Better to work in pairs.
- Start from loadmaster-skel, use the various links in the previous slides and try to write it.
- Feel free to ask if you have any doubt.

`loadmaster` going further 1

- Proposed additional flags:

```
build-time --pipeline=NAME --day=DATE            (cumulative over all jobs)
build-time --pipeline=NAME --day=DATE --per-job  (with details per job)
build-time --pipeline=NAME --day=DATE --job=NAME (only that job)

build-time --pipeline=NAME --from=DATETIME --to=DATETIME
```

- Understand response paging and how to navigate through them.

`loadmaster` going further 2

- Add flag to sort per average build time
- Add a column frequency, that shows (average) builds per day (requires to have solved pagination)
- Add a feature to report the top K most expensive jobs among ALL the pipelines of a Concourse team!, as usual, given a time window
- Add human time windows, for example `last-day`, `last-week`, `last-N-hours=N` …

`loadmaster` going further 3

The time in human form is confusing to compare, because it can be:

```
123h32m7s
42s
```

while is should be something like:

```
123h 32m 07s
  0h 00m 42s
```

with minutes and seconds are always 2 digits, while the hours are the minimum digits for the value.

`loadmaster` going further 4

- Focusing on a given branch (eg `banana-master`) is not really representative of the Concourse resource consumption of a project.
- Instead, we should report cumulative time per project.
- This can be done by considering the pipeline **prefix**: `banana-master`, `banana-staging`, `banana-feat-1` and so on are all part of the **banana** project.

Sources for learning Go and references

Online

More or less in suggested reading order:

- A tour of Go.
- Go by Example.
- Effective Go.
- Go Tutorials.
- Documentation overview.
- The Go blog.
- Your Basic: Go. Very good!
- Bitfield Consulting: Go. Verbose but good, start from the oldest post.

Books

- Let’s Go. Build a web app in Go, step by step, server-side rendering with some JavaScript. Very good.
- Let’s Go further. Build a web API in Go. Very good.

Videos

- All talks from Liz Rice, she is super. For example:
  - Containers from scratch.
  - Debuggers from scratch.
  - Beginner guide to eBPF programming with Go.

Free Trainings

- Exercism, the Go track. Good.
- Learn Go with tests. TDD! very good.
- Gophercises. Did not take it but author is good, so I assume course is good.

Paying Trainings

- Ardan Labs. Very good. I followed the videos (the cheapest option).
- CodeCrafters. Very good. Build your own Redis, Git, Docker, SQLite from scratch.