# The effective Distributed Systems developer curriculum, with Golang [PUBLIC]

Golang is a language that is very easy to learn. Building robust and safe Distributed Systems is an art that is hard to master.

This curriculum serves as an entry point for those that are new to Golang, but most importantly, **it is a gateway for any developer to boost their knowledge in distributed systems engineering**. It goes from learning the quirks of a language designed for systems engineering, to mastering tools such as tracing and profiling to debugging complex systems. The curriculum expects that the reader knows how to program, it will not focus on how single process memory allocation works, nor how routines are called.

The curriculum is laid out for someone to boost their skills over a 5 days period. These do not need to be 5 consecutive days, but we do recommend the student to invest a significant amount of time in the beginning and to repeat the program over time, both to master any details that were overlooked or to share their knowledge with future team members.

By completing this curriculum, you will be able to:

- Build a complete distributed system using your bare hands (and a laptop :D)
- Debug nasty concurrency bugs using the tooling available in the Golang ecosystem
- Jump into any bug of a Golang program with confidence

# Curriculum

## Day 0 - The Golang fundamentals - Absorb tons of Golang wisdom, be a sponge, let the force flow through you and see how things click

*[Optional Day]*

**Goals of the Day**

- Get immersed in the world of Go. Learn the lingo and the known common pitfalls
- Use crafted education materials for accelerated learning

**Tasks**

- [ ] Watch Learn Go in 12 minutes (12 mins at 1x speed)
- [ ] Get your dev environment ready! (20 mins at 1x speed)
    - Visual Studio Code is the recommended IDE https://code.visualstudio.com with https://code.visualstudio.com/docs/languages/go
    - Vim-go for Vim https://github.com/fatih/vim-go
    - lsp-mode & go-mode for Emacs
- Start the Ultimate Go Course (8.5 hours at 1x speed)
  ( each participant to create a personal account / expense after )
    - [ ] Run through Chapter 01 of Ultimate Go - Design Guidelines
    - [ ] Run through Chapter 02 of Ultimate Go - Memory & Data Semantics
    - [ ] Run through Chapter 03 of Ultimate Go - Data Structures
    - [ ] Run through Chapter 04 of Ultimate Go - Decoupling
    - [ ] Run through Chapter 05 of Ultimate Go - Composition
    - [ ] Run through Chapter 06 of Ultimate Go - Error Handling
- [ ] Read the Golang contributing guidelines. (10 mins)

***Total time - 09:12h if videos are watched at 1x speed***

- [ ] (Bonus) Read https://golang.org/doc/effective_go.html
- [ ] (Bonus) Read the Go Programming Language book
    - This book covers a lot. Keep it accessible throughout the course as it is structured to be a resource that is easy to consult.

# Day 1 - Advanced Go (distributed systems-focused) - Solve complex distributed systems shenengingams using Golang tooling

**Goals of the Day**

- Solve complex distributed systems problems
- Master the use of Go modules, channels, goroutine and synchronization primitives
- Writing effective tests
- Debugging in Go (delve)
- Profiling Go code

**Tasks**

- 6.824 (Distributed Systems) Labs (???)
    - [ ] Complete the Exercise 1 - Map Reduce
        - [ ] Submit your solution for review at https://github.com/protocol/DSystems-Curriculum/blob/master/cohort-1/group-N-map-reduce where N is your group number (see the Buddy system at the end of the doc)
        - [ ] (Bonus) Instrument your solution with Open Tracing
- Continue the Ultimate Go course (5.7 hours at 1x speed)
    - [ ] Run through Chapter 07 of Ultimate Go - Packaging
    - [ ] Run through Chapter 08 of Ultimate Go - Goroutines
    - [ ] Run through Chapter 09 of Ultimate Go - Data Races
    - [ ] Run through Chapter 10 of Ultimate Go - Channels
    - [ ] Run through Chapter 11 of Ultimate Go - Concurrency Patterns
    - [ ] Run through Chapter 12 of Ultimate Go - Testing
    - [ ] Run through Chapter 13 of Ultimate Go - Benchmarking
    - [ ] Run through Chapter 14 of Ultimate Go - Profiling & Tracing
- [ ] Attend a demo session on Jaeger: (30 mins)
    "How does Jeromy use Jaeger for debugging Filecoin", **Recording** 📼

***Total time - 6:00h (+ exercise) if videos are watched at 1x speed***

- [ ] (Bonus) Watch "Concurrency is not Parallelism" by Rob Pike

## Day 2 - Imagine and develop a distributed system in Golang

**Goals of the Day**

- Continue on the mastery path from Day 1
- Learn the way of deploy services in Go.
- Get your hands dirty, see what your new superpowers enable you to do. Imagine an develop a distributed system

**Tasks**

- [ ] Build PADI-FS (???)
    - [ ] (Bonus) Instrument your solution with Open Tracing
    - [ ] Submit your solution for review at https://github.com/protocol/DSystems-Curriculum/blob/master/cohort-1/group-N-padi-fs where N is your group number (see the Buddy system at the end of the doc)

***Total time - Full day***

- [ ] (Bonus) Run through the Ultimate Service course

### FAQ for PADIFS

- ***Q: Do I need to write a paper about the project?***
- A: No, but you need to have a one pager that explains the architecture you designed with your group given the constraints (i.e. 1~3 meta data serves, 1~n data storage nodes, etc). (suggestion, ascii art looks great in readmes!)
- ***Q: Do I have to use a specific RPC package?***
- A: No, but you do need to use one (that you get to pick), after all it is intended to be a distributed system :)
- ***Q: Do I need to implement Raft or Paxos to solve this?***
- A: No, although you can! :D. Have in mind that grad students that have solved this just learned about
    - A ton on SQL databases and master-slave replication
    - Are inspired by systems such as GFS, Spanner, Coda, Zookeeper, and others
    - Are probably having their minds explode after learning about Paxos in class

## Day 3 - Review exceptional codebases & contributing patterns, learn the tricks of the masters, apply those tricks to the mini projects.

**Goals of the Day**

- Get good at recognizing great and idiomatic Golang code
- Contribute to go-ipfs + go-libp2p

**Tasks**

- [ ] Read "Guidelines for PR Review"
- [ ] Read https://dave.cheney.net/2014/10/17/functional-options-for-friendly-apis
- [ ] Read https://github.com/golang/go/wiki/CodeReviewComments
- [ ] Read https://github.com/golang/go/wiki/SliceTricks
- [ ] Watch 7 common mistakes in Go and when to avoid them by Steve Francia
- [ ] Watch Hacking with Andrew and Brad: an HTTP/2 client
- [ ] Review & refactor using your new knowledge, one or more of the go-ipfs & go-libp2p repos (increasing code coverage & adding docs is also great). You can find the list of packages at
    - https://github.com/ipfs/go-ipfs#packages
    - https://github.com/libp2p/go-libp2p#packages
      In addition, Steven also left some notes of some of the top ones that could definitely see some love
        - https://github.com/ipfs/go-mfs. Things to look into:
            - "Opening" a file for writing takes an exclusive lock. This is not what users expect.
            - Moving, renaming, copying, etc. a file while it's open will lead to inconsistent results.
            - Calls the list of open files a "cache" but it's definitely not a cache.
            - Corollary, there's no way to safely cleanup this "cache" while MFS is being used.
            - Very few tests.
            - No docs, examples, etc.
        - https://github.com/ipfs/go-ipfs-posinfo
            - Should not be a separate repo. This is an example of a lazy hack that was never cleaned up.
        - https://github.com/ipfs/go-ipfs-files
            - Terrible constructors.
            - No documentation.
            - No examples.

***Total time - Full day***

- [ ] (bonus) Review the following codebases:
    - https://github.com/etcd-io/bbolt

    - Standard library - net/, io/ioutil/
    - Not-quite-standard library but often used - x/oauth2/google, x/net/proxy
      For each of the codebases reviewed, pay close attention to and share (on slack threads) your observations on:
        - Interface design (function signatures)
        - Use of types and interfaces
        - Use of channels and synchronization
        - Structure of functions and error handling
- [ ] (bonus) Review and refactor the code you created on Day 1 & Day 2

# Day 4 - Build a distributed application using go-ipfs/go-libp2p

**Goals of the Day**

- Eat our own dogfood and see how it tastes

**Tasks**

- [ ] Build with your buddy one (or more) of the following apps/tools using nothing but Golang!
    - App suggestions
        - *Build a crawler for the IPFS Network.*
            - With your app/tool, can you answer:
                - ❓ How many nodes are present in the IPFS/libp2p Network?
                - ❓ What is the daily churn?
            - Inspiration and resources to help you with this option:
                - Bittorrent Mainline statistics
                - FindPeers API Documentation (you might find using go-ipfs as a library useful)
        - *Build a benchmarking tool for libp2p transports (compare at least 3)*
            - With your app/tool, can you answer:
                - ❓How does each of the transports compare?
                - ❓What's the max throughput of each? What does that imply with regards to libp2p's overhead (vs. raw transport)?
                - ❓What assumptions does your measurement make that might not apply to a real-world environment?
            - Inspiration and resources to help you with this option:
                - libp2p host example (shows how to plug a transport)
                - The Lifecycle of a libp2p Connection
        - Don't like the ideas above? Surprise us ⚡!
    - Submit your app/tool for review at https://github.com/protocol/DSystems-Curriculum/blob/master/cohort-1/group-N-NAME where N is your group number (see the Buddy system at the end of the doc) and NAME being the name of the project you created.

**Materials**

- Tutorial: Using go-ipfs as a library.

# Extra material

If you are interested in continuing mastering the art of distributed systems with Golang, we recommend:

### Talks

- Understanding `nil` https://www.youtube.com/watch?v=ynoY2xz-F8s

### Courses

- https://github.com/golang/go/wiki/Training#classroom-and-in-person
- The Ultimate Docker course
- 6.824 (Distributed Systems) Labs
    - Exercise 1 - Map Reduce (done in the regular curriculum)
    - Exercise 2 - Raft
    - Exercise 3 - Fault-Tolerant Key/Value Service
    - Exercise 4 - Sharded Key/Value Service

### Books

- https://www.goodreads.com/book/show/23463279-designing-data-intensive-applications

### Articles

- In 2019, there was a large community discussion on making errors less verbose, that eventually did not happen. The proposal and discussion is perhaps a useful window into the design philosophy of the language.

### Conferences

- Consult Amazing conferences to speak at (apply Go filter)