Go Web Dev

- Create user & delete user
- MVC design pattern - model view controller
- Install mongodb
- Connect to mongodb
- CRUD with Go & mongodb
- Hands on exercise & solution
- Hands on exercise & solution
- Hands on exercise & solution

Docker

- Introduction to Docker
- Virtual machines & containers
- Installing docker
- Docker whalesay example
- Using a Dockerfile to build an image
- Launching a container running curl
- Running a Go web app in a Docker container
- Pushing & pulling to docker hub
- Go, Docker & Amazon Web Services (AWS)

PostgreSQL

- Installing postgres
- Create database
- Create table
- Insert records
- Auto increment primary key
- Hands-on exercise
- Hands-on exercise - solution
- Relational databases

- Limit
- Sort
- Index
- Aggregation
- Users
- JSON
- Create Read Update Delete (CRUD)

More to come ...

- Google Cloud section coming in a few weeks
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx
- xx

xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xxDocker, Kubernetes, Microservices
Hello world

Getting started

Why Go for web dev

- server-side?
  - #1 Go
  - #3 Node.js
  - #3 Python
  - #4 Ruby
  - #5 PHP
- 2006 first intel dual-core processor
- 2007 Google begins development of Go
  - Rob Pike, Ken Thompson, Robert Griesemer
  - FAQ - Why are you creating a new language?
  - language features
    - take advantage of multiple cores
    - easy concurrency based upon Tony Hoare's CSP
    - compiled, static type, GC
  - goals
    - efficient compilation
    - efficient execution
    - ease of programming
      - clean syntax
      - distributed teams
- **Why go for web dev?**
  - **Go was built to do what google does**
  - **Google is rewriting Google with Go**
- 2009 open-sourced
- 2012 version 1

- **free preview this video**
- **video: lec 01 - take 02**

Prerequisites

- CLI
- github
- HTML / CSS
    - How to Create A Website: An HTML Tutorial and CSS Tutorial
- Go programming
    - Learn How To Code: Google's Go (golang) Programming Language
- **free preview this video**
- **video: lec 02 - take 01**

Resources

- Here are the resources which I think are the best:
    - **These notes:**
        - **https://goo.gl/K5GccY**
    - Go programming resources
    - The Code Used In This Course
        - https://github.com/GoesToEleven/golang-web-dev
    - Go Language Fundamentals
        - course
            - https://www.udemy.com/learn-how-to-code/
        - repo

        - https://github.com/GoesToEleven/GolangTraining
    - HTML & CSS
        - course
            - How to Create A Website: An HTML Tutorial and CSS Tutorial
        - repo
            - https://github.com/GoesToEleven/html-css-bootcamp
    - **video: lec 03 - take 01**

Language review

- variables
    - short variable declaration operator
    - using the var keyword to declare a variable
    - scope
- data structures
    - slice
    - map
    - struct
        - composite literal
- functions
    - func (receiver) identifier(parameters) (returns) { <code> }
    - methods
- composition
    - embedded types
    - interfaces
    - polymorphism
    - nice article
- hands-on exercises #1
- hands-on exercises #2
    - use https://play.golang.org/ , screenshot your solution, tweet **@Todd_McLeod #golangsnippet**

- **free preview this video**
- **file: 001_prereq**
- **video: lec 04 - take 01**

How to succeed

- Bill Gates & Warren Buffet
    - the one word they both chose as the most important contributor to their success: focus
- Priorities, commitment, focus
    - What is important to you in your life? Prioritize.
    - Can you give time everyday to that which is important? Commitment.
    - Give time to the important everyday. Focus.
- Drop by drop, the bucket gets filled
- **video: lec 05 - take 02**

Templates

Understanding templates

Templates allow us to customize information for a use. This is how we get personalized webpages. Templates are the first thing you must learn to do web programming.

- **files: 002_template**
- **video: lec 06 - take 04**

Templating with concatenation

Go encourages the developer to think like a programmer. How can you solve this problem with programming? Could we create a webpage, and merge data with it, by just working with strings?

- **files: 003_string-to-html**
- **video: lec 07 - take 01**

Understanding package text/template: parsing & executing templates

We are going to store our templates in their own files. It is customary practice to give these files a ".gohtml" extension. We will go through two steps to use our templates: (1) Parse (2) Execute. In this course, we will use ParseGlob and ExecuteTemplate. Package text/template is explained.

- steps
  - PARSE the template
  - EXECUTE the template
- performance
  - always parse your templates upon application initialization
  - do not parse your templates every time a user asks for a template
- *Template
  - container that holds the parsed templates
- Quick Note
  - Atom will require tweaking of keymap.cson to allow some plugins like Emmet to work on tab expansion
    - 'atom-text-editor[data-grammar="text html gohtml"]:not([mini])':
      'tab': 'emmet:expand-abbreviation-with-tab
    - Will allow Emmet plugin to tab expand on files with the gohtml extension

- **files: 003_string-to-html - 004_parse_execute**
- **video: lec 08 - take 04**

Passing data into templates

When we execute our template, we can pass data into our template.

- **files: 005_data**
- **video: lec 09 - take 01**

Variables in templates

We can assign values to variables in templates.
ASSIGN
{{$wisdom := .}}
USE
{{$wisdom}}

- **files: 006_variable**
- **video: lec 10 - take 02**

Passing composite data structures into templates

This lecture provides you with examples of passing various data types to templates.

- **files: 007_data-structures**
- **video: lec 11 - take 01**

Functions in templates

During execution functions are found in two function maps: first in the template, then in the global function map. By default, no functions are defined in the template but the Funcs method can be used to add them. Predefined global functions are defined in text/template.

- **files: 008_func/01**
- **video: lec 12 - take 15**

Pipelines in templates

Pipelines allow us to take the value which is output from one process or function, and pass it as the input to the next function. Also covered in this video: working with package time and formatting type Time in a template.

- **files: 008_func/02 - 008_func/04**
- **video: lec 12_02 - take 3**

Predefined global functions in templates

During execution functions are found in two function maps: first in the template, then in the global function map. By default, no functions are defined in the template but the Funcs method can be used to add them. Predefined global functions are defined in text/template.

- **files: 009_predefined-global-functions**
- **video: lec 13 - take 02**

Nesting templates - modularizing your code

When parsing a template, another template may be defined and associated with the template being parsed. Template definitions must appear at the top level of the template, much like global variables in a Go program. The syntax of such definitions is to surround each template declaration with a "define" and "end" action. Comments in templates are also covered.

- **files: 010_nested-templates**
- **video: lec 14 - take 01**

Passing data into templates & composition

In this lecture, we will pass data structures into templates. We will build our data structures using composition. FYI, from wikipedia though modified: Composition is the principle that types should achieve polymorphic behavior and code reuse by their composition (by embedding other types). An implementation of composition typically begins with the creation of various interfaces representing the behaviors that the system must exhibit. The use of interfaces allows this technique to support the Polymorphic behavior that is so valuable. Types implementing the identified interfaces are built and added as needed. Thus, system behaviors are realized without inheritance.

- **files: 011_composition-and-methods**
- **video: lec 15 - take 02**

Using methods in templates

In this lecture, we will pass data structures into templates. We will build our data structures using composition. FYI, from wikipedia though modified: Composition is the principle that types should achieve polymorphic behavior and code reuse by their composition (by embedding other types). An implementation of composition typically begins with the creation of various interfaces representing the behaviors that the system must exhibit. The use of interfaces allows this technique to support the Polymorphic behavior that is so valuable. Types implementing the identified interfaces are built and added as needed. Thus, system behaviors are realized without inheritance.

- **files: 011_composition-and-methods**
- **video: lec 15_02 - take 03**

Hands-on exercises

Here are several hands-on exercises, with solutions, to help you learn how to pass data to templates.

- **files: 012_hands-on**
- **video: lec 16 - take 01**

Using package html/template, character escaping, & cross-site scripting

Package html/template has all of the functionality of package text/template, plus additional functionality specific to HTML pages. In particular, package html/template has context aware escaping so that dangers like cross-site scripting are avoided.

- **files: 013_xss**
- **video: lec 17 - take 06**

Creating your own server

Understanding servers

Before we get started building our own server, there are several important things to know:

- synonymous terms in web programming
    - router
    - request router
    - multiplexer
    - mux
    - servemux
    - server
    - http router
    - http request router
    - http multiplexer
    - http mux
    - http servemux
    - http server
- client / server architecture
    - request / response pattern
        - like in restaurants
- OSI model
- HTTP runs on top of TCP
- HTTP is a protocol - rules of communication
    - *HyperText Transfer **Protocol***
- IETF sets recommendations for HTTP
    - Request For Comment - RFC 7230 is for HTTP1.1

- **files: 014_understanding-servers**
- **video: lec 18 - take 02**

TCP server - write to connection

We can create our own tcp server using the net package from the standard library. There are three main steps: (1) Listen (2) Accept (3) Write or Read to the connection. We will use telnet to call into the TCP server we created. Telnet provides bidirectional interactive text-oriented communication using a virtual terminal connection over the Transmission Control Protocol (TCP).

- **files: 015_understanding-TCP-servers / 01_write**
- **video: lec 19 - take 01**

TCP server - read from connection using bufio.Scanner

We will now modify our TCP server to handle multiple connections. We will do this by using goroutines. We will also modify our TCP server to read from the connection. We will then contact our TCP server on port 8080 using our web browser. This will allow us to see the text sent from the browser to the TCP server and how this text adheres to HTTP (RFC 7230).

- **files: 015_understanding-TCP-servers / 02_read**
- **video: lec 20 - take 02**

TCP server - read from & write to connection

Now we are going to read and write from/to our connection.

- **files: 015_understanding-TCP-servers / 03_read-write & 04_read-write-setDeadline**
- **video: lec 21 - take 06**

TCP server - code a client

We can use the net package to create a client which dials into our TCP server.

- **files: 015_understanding-TCP-servers / 05_dial-read & 06_dial-write**
- **video: lec 22 - take 05**

TCP server - rot13 & in-memory database

Here are two sample TCP server apps.

- **files: 015_understanding-TCP-servers / 07_tcp-apps**
- **video: lec 23 - take 03**

TCP server - HTTP request / response foundation hands-on exercise

Here is how we build the foundation of a TCP server to handle HTTP requests and responses. This video also introduces a hands-on exercise.

- **files: 016_building-a-tcp-server-for-http / 01**
- **video: lec 24 - take 01**

TCP server - HTTP method & URI retrieval hands-on exercise

This is the solution to the hands-on exercise of retrieving the URI and displaying it. The next hands-on exercise is also introduced.

- **files: 016_building-a-tcp-server-for-http / 03_solution**
- **video: lec 25 - take 01**

TCP server - HTTP multiplexer

This is the solution to the hands-on exercise of creating a TCP server that handles requests & responses adhering to HTTP. The server will respond with different code according the both the method and the URI.

- **files: 016_building-a-tcp-server-for-http / 05_solution**
- **video: lec 26 - take 02**

Understanding net/http package

net/http package - an overview

The best entry point to understanding the net/http package is covered. It is essential to know the standards of HTTP. The first thing you should know about the net/http package is the Handler interface. ListenAndServe takes a value which implements the handler interface.

- **files: 017_understanding-net-http-package**
- **video: lec 28 - take 01**
  (lec 027 was accidentally skipped in numbering)

Understanding & using ListenAndServe

ListenAndServe is built using what we have just learned: from the net package, Listen & Accept. ListenAndServe takes an address, the port on which you want to listen, and a Handler. It is imperative that you solidly know type Handler.

- **files: 017_understanding-net-http-package / 01_Handler & 02_listenAndServe**
- **video: lec 29 - take 02**

Foundation of net/http: Handler, ListenAndServe, Request, ResponseWriter

The foundation of the net/http package is type Handler and ListenAndServe. ListenAndServe takes a value of type Handler. Type Handler has two parameters of type ResponseWriter and a pointer to a Request. Understanding the relationship of these parts is essential to building web apps with Go.

- **files: 017_understanding-net-http-package / 01_Handler & 02_listenAndServe**
- **video: lec 30 - take 01**

Retrieving form values - exploring *http.Request

When a user submits data to a server, that data is attached to the request. Remember, the HTTP specification (RFC 7230) says that a request will have three parts: (1) request line, (2) headers, (3) body (also known as payload). We can retrieve values submitted by the user by working with the Request type. The type Handler has a pointer to a Request (*http.Request) as one of the parameters required by the ServeHTTP method. The Request type is a struct with Form & PostForm fields that allow us to access data submitted by a user. We can also use methods attached to the Request type to access data: FormValue & FormFile.

- **files: 017_understanding-net-http-package / 03_Request / 01_ParseForm_Form**
- **video: lec 30_02 - take 01**

Retrieving other request values - exploring *http.Request

There are other request values which we can retrieve such as the method and the URL.

- **files: 017_understanding-net-http-package / 03_Request / 02_Method - 05_Host_ContentLength**
- **video: lec 31 - take 01**

Exploring http.ResponseWriter - writing headers to the response

This video will continue to reinforce your understanding of the net/http package. We will learn how to read documentation and write headers to our response.

- **files: 017_understanding-net-http-package / 03_Request / 04_ResponseWriter**
- **video: lec 32 - take 01**

Review

Reviewing: type Handler, ListenAndServe, *Request, ResponseWriter.

- **video: lec 33 - take 03**

Understanding routing

Understanding ServeMux

ServeMux is an HTTP request multiplexer. A ServeMux matches the URL of each incoming request against a list of registered patterns and calls the handler for the pattern that most closely matches the URL.

- **ServeMux**
  - NewServeMux
    - We can create a *ServeMux by using NewServeMux.
  - default ServeMux
    - We can use the default ServeMux by passing nil to ListenAndServe.
- **Handle**
  - takes a value of type **Handler**
- **HandleFunc**
  - takes a func with this signature: **func(ResponseWriter, *Request)**
- **files: 018_understanding-net-http-ServeMux & 019_HandleFunc**
- **video: lec 34 - take 05**

Disambiguation: func(ResponseWriter, *Request) vs. HandlerFunc

The differences between **func(ResponseWriter, *Request)** and **HandlerFunc** are explained and illustrated.

- **files: 020_HandlerFunc**
- **video: lec 35 - take 01**

Third-party servemux - julien schmidt’s router

Julien Schmidt’s package httprouter "github.com/julienschmidt/httprouter" is a trie based high performance HTTP request router.

- **files: 021_third-party-serveMux**
- **video: lec 36 - take 07**

Hands-on exercises

These hands-on exercises will reinforce what you are learning.

- **files: 022_hands-on**
- **video: lec 37 - take 02**

Hands-on exercises - solutions #1

Solutions to hands on exercises in folder 022_hands-on/01

- **files: 022_hands-on / 01**
- **video: lec 38 - take 01**

Hands-on exercises - solutions #2

Solutions to hands on exercises in folder 022_hands-on/02

- **files: 022_hands-on / 02**
- **video: lec 39 - take 01**

Serving files

Serving a file with io.Copy

io.Copy allows us to copy from a reader to a writer. We can use io.Copy to read from a file and then write the file to the response.

- **files: 023_serving-files**
- **video: lec 40 - take 01**

Serving a file with http.ServeContent & http.ServeFile

ServeContent & ServeFile both allow us to serve a single file.

- **files: 023_serving-files / 02_serving-files**
- **video: lec 41 - take 01**

Serving a file with http.FileServer

The preferred method for serving files is http.FileServer.

- **files: 023_serving-files / 02_serving-files / 04_FileServer**
- **video: lec 42 - take 01**

Serving a file with http.FileServer & http.StripPrefix

The preferred method for serving files is http.FileServer. We will use StripPrefix to facilitate the serving of fles.

- **files: 023_serving-files / 02_serving-files / 04_FileServer**
- **video: lec 43 - take 02**

Creating a static file server with http.FileServer

There is a special case: if you have an index.html file in a directory that FileServer is serving, then the FileServer will serve that file when only the root of the directory (“/”) is asked for.

- **files: 023_serving-files / 02_serving-files / 04_FileServer**
- **video: lec 44 - take 01**

log.Fatal & http.Error

Here are two pieces of code that you will sometimes see.

- **files: 023_serving-files**
- **video: lec 45 - take 01**

Hands-on exercises

These hands-on exercises will help reinforce what you are learning.

- **files: 024_hands-on**
- **video: lec 46 - take 01**

Hands-on exercises - solutions

Here are the solutions to the hands-on exercises.

- **files: 024_hands-on**
- **video: lec 47 - take 02**

The http.NotFoundHandler

We can use the NotFoundHandler as the Handler for something that isn't found.

- **files: 025_NotFoundHandler**
- **video: lec 48 - take 03**

Deploying your site

Buying a domain - google domains

When deploying a project to Google Cloud, it is good to have your domain with Google Domains. This will make the configuration of your domain easier.

- **video: lec 50 - take 01**

Deploying to google cloud

- publishing your site
    - buying a domain
        - https://domains.google/#/
    - deploying to Google Cloud https://cloud.google.com/
        - install google appengine
            - https://cloud.google.com/appengine/docs/go/download
        - make sure python is installed VERSION 2.7.x
            - python -V
            - https://www.python.org/downloads/release/python-2712/

        - configure environment PATH variables
        - google cloud developer console
            - create a project
            - get the project ID
        - update the app.yaml file with your project ID
        - deploy to that project
            - **appcfg.py -A <YOUR_PROJECT_ID> -V v1 update .**
        - view your project
            - http://<YOUR PROJECT ID>.appspot.com/
            - example
                - http://temp-145415.appspot.com/
        - change DNS info at google domains
            - point your domain to your appengine project
- **files: 026_appengine-deploy**
- **video: lec 51 - take 07**

Creating state

State overview

HTTP does not have state built into it. You could say HTTP is stateless, though it has the tools necessary for you, as a developer, to create state. In this section, we'll learn how to create state on the web. To learn about state, check out wikipedia.

- **video: lec 52 - take 03**

Passing values through the URL

We can pass values through the URL. We can retrieve them with req.FormValue

- **files: 027_passing-data**
- **video: lec 53 - take 01**

Passing values from forms

When a form is submitted, we can pass the submitted values either through the request body payload or through the URL. If the form's method attribute is post, then the values of the form are sent to the server through the request body's payload. If the form's method attribute is get, then the values of the form are sent to the server through the URL.

- **files: files: 027_passing-data / 02_form-post - 03_form-get - 04_form**
- **video: lec 54 - take 02**

Uploading a file, reading the file, creating a file on the server

In many web programming languages, dealing with files can be a challenge. In Go, it's easy. We'll see in this video how to allow a user to upload a file. We'll also see how to read that file and, if you want, create a new file to store on the server.

- **files: 027_passing-data / 05_form-file**
- **video: lec 55 - take 06**

Enctype

"When you make a POST request, you have to encode the data that forms the body of the request in some way. HTML forms provide three methods of encoding:

- **application/x-www-form-urlencoded** (the default)
- **multipart/form-data**
- **text/plain**

Work was being done on adding application/json, but that has been abandoned. The specifics of the formats don't matter to most developers. The important points are: When you are writing client-side code, **all you need to know is use multipart/form-data when your form includes any <input type="file"> elements.**

text/plain Never use text/plain. If you are writing (or debugging) a library for parsing or generating the raw data, then you need to start worrying about the format. You might also want to know about it for interest's sake. **text/plain** is introduced by HTML 5 and is useful only for debugging — from the spec: They are not reliably interpretable by computer — and I'd argue that the others combined with tools (like the Net tab in the developer tools of most browsers) are better for that).

application/x-www-form-urlencoded is more or less the same as a query string on the end of the URL.

multipart/form-data is significantly more complicated but it allows entire files to be included in the data."
http://stackoverflow.com/questions/4526273/what-does-enctype-multipart-form-data-mean
https://www.w3.org/TR/html5/forms.html#text/plain-encoding-algorithm
https://www.ietf.org/rfc/rfc1867.txt
https://www.ietf.org/rfc/rfc2388.txt

- **files: 027_passing-data / 05_form-file**
- **video: lec 55-02 - take 06**

Redirects - overview

On the web, we have a client / server architecture. Clients make requests, and servers write responses to those clients. The request and response are both just text that must conform to the rules of HTTP. Both the request & response have a start line, headers, and a body. The request start line is called the "request line". It consists of

request-line = method SP request-target SP HTTP-version CRLF

The response start line is called the “status line”. It consists of

status-line = HTTP-version SP status-code SP reason-phrase CRLF

You can see that the status line wants a status-code and a reason-phrase. For redirects, we have several status-codes and reason-phrases from which we can choose, but primarily you should choose either

- 301 moved permanently
- 303 see other // changes method to get
- 307 temporary redirect // keeps same method

- **files: 028_redirect**
- **video: lec 56 - take 01 (- 07:06)**

Redirects - diagrams & documentation

The definitive source for knowing which status code to use for HTTP/1.1 is RFC 7231. Diagrams of 301, 303, and 307 are presented. The status code 418 & 420 are also discussed.

- **files: 028_redirect**
- **video: lec 56_02 - take 01**

Redirects - in practice

Here is a code review of redirects in action.

- **files: 028_redirect**
- **video: lec 57 - take 03 (1:55 -)**

Cookies - overview

Cookies allow us to maintain state. We can write a unique ID to a cookie. When a client makes a request to a server at a particular domain, if there is a cookie from that particular domain on the client’s machine, the browser will include that cookie in the request.

The server can then read that cookie, pull out the unique ID, and know which user is making the request. There are various methods to make this all secure, but the primary method is to use HTTPS.

- **files: 029_cookies**
- **video: lec 58 - take 03**

Cookies - writing and reading

How to write and read a cookie to and from a client's machine.

- **files: 029_cookies**
- **video: lec 59 - take 01**

Writing multiple cookies & hands-on exercise

Two examples showing how (1) you can write multiple cookies and (2) you can create a counter.

- **files: 029_cookies**
- **video: lec 60 - take 03**

Hands-on exercise solution: creating a counter with cookies

How to create a counter to track how many times a user has been to your website domain using a cookie.

- **files: 029_cookies**
- **video: lec 61 - take 02**

Deleting a cookie

To delete a cookie, set the “MaxAge” field to either zero or a negative number. You can expire a cookie by setting one of these two fields: Expires or MaxAge Expires sets the exact time when the cookie expires. Expires is Deprecated. MaxAge sets how long the cookie should live (in seconds).

- **files: 029_cookies**
- **video: lec 62 - take 01**

Creating sessions

Sessions

In computer science, a session is an interactive information interchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user. A session is set up or established at a certain point in time, and then torn down at some later point.

An HTTP exchange between a browser and a server may include an HTTP cookie which identifies state using a unique ID which can be used to look up the user.

Each transaction in HTTP creates a separate connection. Maintaining session continuity between phases requires a session ID. The session ID is embedded within the <A HREF> or <FORM> links of dynamic web pages so that it is passed back to the server. The server then uses the session ID to ensure session continuity between transactions.

A unique ID session token is a unique identifier that is generated and sent from a server to a client. The client usually stores and sends the token as an HTTP cookie and/or sends it as a parameter in GET or POST queries. The reason to use session tokens is that the client only has to handle the identifier—all session data is stored on the server (usually in a database, to which the client does not have direct access) linked to that identifier.

- **video: lec 63 - take 03**

Universally unique identifier - UUID

A universally unique identifier (UUID) is an identifier standard used in software construction. A UUID is simply a 128-bit value. The meaning of each bit is defined by any of several variants. For human-readable display, many systems use a canonical format using hexadecimal text with inserted hyphen characters. For example: 123e4567-e89b-12d3-a456-426655440000 The intent of UUIDs is to enable distributed systems to uniquely identify information without significant central coordination. In this context the word unique should be taken to mean **"practically unique"** rather than "guaranteed unique". Since the identifiers have a finite size, it is possible for two differing items to share the same identifier. This is a form of hash collision. The identifier size and generation process need to be selected so as to make this sufficiently improbable in practice. Anyone can create a UUID and use it to identify something with reasonable confidence that the same identifier will never be unintentionally created by anyone to identify something else. Information labeled with UUIDs can therefore be later combined into a single database without needing to resolve identifier (ID) conflicts. Adoption of UUIDs is widespread with many computing platforms providing support for generating UUIDs and for parsing/generating their textual representation. Only after generating 1 billion UUIDs every second for the next 100 years would the probability of creating just one duplicate would be about 50%.

- **files: 030_sessions / 01_uuid**
- **video: lec 64 - take 01**

Your first session

You can [see this live-coded here: https://www.youtube.com/watch?v=sknrOJqBpd0](https://www.youtube.com/watch?v=sknrOJqBpd0).

- **files: 030_sessions / 02_session**
- **video: lec 65 - take 08**

Sign-up

Creating a user sign-up page. Processing the form submission. Storing information in our maps.

- **files: 030_sessions / 03_signup**

- **video: lec 66 - take 01**

Encrypt password with bcrypt

bcrypt is a password hashing function designed by Niels Provos and David Mazières. Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power. The bcrypt function is the default password hash algorithm for OpenBSD and other systems including some Linux distributions such as SUSE Linux. We will use bcrypt to encrypt user passwords before storing them.

- **files: 030_sessions / 04_bcrypt**
- **video: lec 67 - take 02**

Login

We will create a form that allows a user to login. We will then check the login credentials to see if the user successfully authenticates.

- **files: 030_sessions / 05_login**
- **video: lec 68 - take 01**

Logout

We will create the functionality to allow a user to logout. This will end the user's session

- **files: 030_sessions / 06_logout**
- **video: lec 69 - take 01**

Permissions

We will add the ability for different users to have different permissions. These different permissions will allow some users to access some areas, while others can't. For instance, if someone had "admin" rights, then they could access the "admin" areas.

- **files: 030_sessions / 07_permissions**
- **video: lec 70 - take 01**

Expire session

We will create the ability for a user's session to expire after a certain period of time. We will need to clear the user's cookie, and remove the entry in the session's map which stores that user's session. In addition, we will want to clear out our session's map on some interval. We will use the "MaxAge" field of the cookie to set the length of time, in seconds, that a cookie lasts.

- **files: 030_sessions / 08_expire-session**
- **video: lec 71 - take 8**

Amazon Web Services

Overview

We will look at some of the various parts of Amazon Web Services (AWS) including: EC2 (Elastic Compute Cloud), S3 (Simple Storage Service), RDS, DynamoDB, ElastiCache, and Route 53. Cloud computing is covered, as is IAAS, PAAS, SAAS.

- **video: lec 72 - take 04b**

Creating a virtual server instance on AWS EC2

We will create a virtual machine running linux on Amazon’s elastic cloud computing.

- **files: 031_AWS/01_hello-world**
- **video: lec 73 - take 20b**

Hello world on AWS

Uploading code to EC2 involves a few steps. First we will build our binary to run on the correct architecture and operating system. Then we will use secure copy to copy our binary to the remote server. After that, we will use secure shell to log into our remote server and run our code.

- **files: 031_AWS/01_hello-world**
- **video: lec 74 - take 01b**

Persisting an application

To have an application continue running after our terminal session ends, we must complete a few steps. Specifically, we will be using systemd. systemd is an init system used in Linux distributions to bootstrap the user space and manage all processes. One of systemd's main goals is to unify basic Linux configurations and service behaviors across all distributions. As of 2015, most Linux distributions have adopted systemd as their default init system.

- **files: 031_AWS/01_hello-world**
- **video: lec 75 - take 05b**

Hands-on Exercise

For this hands-on exercise, deploy the code in "030_sessions/08_expire-session" and get it running on AWS.

- **files: 031_AWS/02_hands-on**

- **video: lec 76 - take 03**

Hands-on Solution

For this hands-on exercise, deploy the code in "030_sessions/08_expire-session" and get it running on AWS.

- **files: 031_AWS/02_hands-on**
- **video: lec 77 - take 06**

Terminating AWS services

When we are finished working with machines, we need to terminate them. Failure to do this might result in billing.

- **video: lec 78 - take 01**

Relational Databases

Overview

Relational databases are made up of tables which hold “like” data. For instance, we might have a “customers” table, a “videos” table, and a “rentals” table which shows which customer rented which video. This video presents you with the fundamentals of a relational database.

- **video: lec 79 - take 01**

Installing MySQL - Locally

Installing MySQL is straightforward. This video will show you the process.

- **video: lec 80 - take 02**

Installing MySQL - AWS

Installing MySQL on AWS is also straightforward. This video will show you the process.

- **video: lec 81 - take 01**

Connect Workbench to MySQL on AWS

This video shows you how to connect your workbench to MySQL on AWS.

- **video: lec 82 - take 04**

Go & SQL - Setup

This video shows you how to set up Go and SQL.

- **video: lec 83 - take 01**

Go & SQL - In Practice

This video shows you how to use Go and SQL.

- **video: lec 84 - take 01**

Scaling on AWS

Overview of load balancers

Load balancers distribute work between different machines.

- **video: lec 85 - take 05**

Create EC2 security groups

A security group acts as a virtual firewall that controls the traffic for one or more instances. When you launch an instance, you associate one or more security groups with the instance. You add rules to each security group that allow traffic to or from its associated instances.

- **files: 033_aws-scaling / 02_load-balancer**
- **video: lec 86 - take 02**

Create an ELB load balancer

Load balancing improves the distribution of workloads across multiple computing resources. Elastic Load Balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances. It enables you to achieve fault tolerance in your applications, seamlessly providing the required amount of load balancing capacity needed to route application traffic. Elastic Load Balancing offers two types of load balancers that both feature high availability, automatic scaling, and robust security. These include the Classic Load Balancer that routes traffic based on either application or network level information, and the Application Load Balancer that routes traffic based on advanced application level information that includes the content of the request. The Classic Load Balancer is ideal for simple load balancing of traffic across multiple EC2 instances, while the Application Load Balancer is ideal for applications needing advanced routing capabilities, microservices, and container-based architectures. Application Load Balancer offers ability to route traffic to multiple services or load balance across multiple ports on the same EC2 instance.

- **files: 033_aws-scaling / 02_load-balancer**

- **video: lec 87 - take 02**

Implementing the load balancer

We will create an image of our web server. We will then create a new instance of a web server from our image. We will make sure both of our web servers are added to the target group being used by the load balancer. We will then test that the load balancer is working correctly.

- **files: 033_aws-scaling / 03_ami**
- **video: lec 88 - take 01**

Connecting to your MySQL server using MySQL workbench

Now that we have our mysql server behind a firewall, we have to connect to it in a different fashion.

- **video: lec 88-a - take 01**

Hands-on exercise

For this hands on exercise, create two web servers that are accessing your database. Show that the web servers are each pulling data. Show that the load balancer is distributing work between the two web servers.

- **video: lec 89 - take 01**

Hands-on solution

Here is my solution to the hands on exercise.

- **video: lec 90 - take 01**

Autoscaling & CloudFront

Here is how to autoscale your application as demand increases. You can also serve your content from the server closest to the consumer using cloudfront.

- **video: lec 91 - take 01**

Photo Blog

Starting files

We are now going to embark on a project. This project will be to build a photoblog. The photoblog should allow users to upload photos. It should also display all of the photos for a user. The starting files for this project are introduced.

- **file: 034_photo-blog / 01_starting**
- **video: lec 92 - take 01**

User data

We will store user data in a cookie. Our first step will be to get a cookie working. We will want to store a UUID in our cookie which will serve as the session ID when we need it. We can also store additional strings in the cookie value. We will separate the different strings we store in the cookie value by the pipe “|” delimiter.

- **file: 034_photo-blog / 02_cookie**
- **video: lec 93 - take 01**

Storing Multiple Values

Several string values are stored, and retrieved, from the cookie.

- **file: 034_photo-blog / 03_store-values**

- **video: lec 94 - take 02**

Uploading pictures

Uploading a picture is a little involved. First, of course, we will need a form. The “enctype” on this form must also be set to “multipart/form-data”. When the form is submitted, we will use “req.FormFile” to retrieve the image. We will then create a file name. From the file header we will get the file extension. We will then use sha1 to create a hash of the file.

- **file: 034_photo-blog / 04_upload-pictures**
- **video: lec 95 - take 02**

Displaying pictures

We will use “http.FileServer” along with “http.StripPrefix” and “http.Dir” to get our images to serve.

- **file: 034_photo-blog / 05_display-pictures**
- **video: lec 96 - take 01**

Web Dev Toolkit

Hash message authentication code (HMAC)

In cryptography, a keyed-hash message authentication code (HMAC) is a specific type of message authentication code (MAC) involving a cryptographic hash function (hence the 'H') in combination with a secret cryptographic key. As with any MAC, it may be used to simultaneously verify both the data integrity and the authentication of a message. Any cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA1 accordingly. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

- **files: 035_hmac**
- **video: lec 97 - take 02**

Base64 encoding

Base encoding of data is used in many situations to store or transfer data in environments that, perhaps for legacy reasons, are restricted to US-ASCII data. Base encoding can also be used in new applications that do not have legacy restrictions, simply because it makes it possible to manipulate objects with text editors. In the past, different applications have had different requirements and thus sometimes implemented base encodings in slightly different ways. Today, protocol specifications sometimes use base encodings in general, and "base64" in particular, without a precise description or reference. Base64 is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format. https://tools.ietf.org/html/rfc4648

- **files: 036_base64**
- **video: lec 98 - take 02**

Web storage

Web storage offers two different storage areas—local storage and session storage—which differ in scope and lifetime. Data placed in local storage is per origin (the combination of protocol, hostname, and port number as defined in the same-origin policy) (the data is available to all scripts loaded from pages from the same origin that previously stored the data) and persists after the browser is closed. Session storage is per-origin-per-window-or-tab and is limited to the lifetime of the window. Session storage is intended to allow separate instances of the same web application to run in different windows without interfering with each other, a use case that's not well supported by cookies.

- **files: 037_web-storage**
- **video: lec 99 - take 03**

Context

Package context defines the Context type, which carries deadlines, cancellation signals, and other request-scoped values across API boundaries and between processes. Incoming requests to a server should create a Context, and outgoing calls to servers should accept a Context. The chain of function calls between them must propagate the Context, optionally replacing it with a derived Context created using WithCancel, WithDeadline, WithTimeout, or WithValue. When a Context is canceled, all Contexts derived from it are also canceled. The WithCancel, WithDeadline, and WithTimeout functions take a Context (the parent) and return a derived Context (the child) and a CancelFunc. Calling the CancelFunc cancels the child and its children, removes the parent's reference to the child,

and stops any associated timers. Failing to call the CancelFunc leaks the child and its children until the parent is canceled or the timer fires. The go vet tool checks that CancelFuncs are used on all control-flow paths. Programs that use Contexts should follow these rules to keep interfaces consistent across packages and enable static analysis tools to check context propagation: Do not store Contexts inside a struct type; instead, pass a Context explicitly to each function that needs it. The Context should be the first parameter, typically named ctx: func DoSomething(ctx context.Context, arg Arg) error { // ... use ctx ... } Do not pass a nil Context, even if a function permits it. Pass context.TODO if you are unsure about which Context to use. Use context Values only for request-scoped data that transits processes and APIs, not for passing optional parameters to functions. The same Context may be passed to functions running in different goroutines; Contexts are safe for simultaneous use by multiple goroutines.

- **files: 038_context**
- **video: lec 100 - take 03 part 01 & part 02**

TLS & HTTPS

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), both frequently referred to as "SSL", are cryptographic protocols that provide communications security over a computer network. Several versions of the protocols find widespread use in applications such as web browsing, email, Internet faxing, instant messaging, and voice-over-IP (VoIP). Websites use TLS to secure all communications between their servers and web browsers. HTTPS (also called HTTP over TLS, HTTP over SSL, and HTTP Secure) is a protocol for secure communication over a computer network which is widely used on the Internet. HTTPS consists of communication over Hypertext Transfer Protocol (HTTP) within a connection encrypted by Transport Layer Security or its predecessor, Secure Sockets Layer. The main motivation for HTTPS is authentication of the visited website and protection of the privacy and integrity of the exchanged data.

- **files: 039_https**
- **video: lec 101 - take 03**

JSON - JavaScript Object Notation

In computing, JSON (JavaScript Object Notation) is an open-standard format that uses human-readable text to transmit data objects consisting of **attribute–value pairs.**

It is **the most common data format used for asynchronous browser/server communication**, largely replacing XML.

JSON is a language-independent data format.

It derives from JavaScript, but as of 2016 many programming languages include code to generate and parse JSON-format data.

The official Internet media type for JSON is **application/json**.

JSON filenames use the extension .json.

Douglas Crockford originally specified the JSON format in the early 2000s; two competing standards, RFC 7159 and ECMA-404, defined it in 2013. The ECMA standard describes only the allowed syntax, whereas the RFC covers some security and interoperability considerations. JSON grew out of a need for stateful, real-time server-to-browser communication without using browser plugins such as Flash or Java applets, which were the dominant methods in the early 2000s. Douglas Crockford was the first to specify and popularize the JSON format.

The JSON.org Web site was launched in 2002. In December 2005, Yahoo! began offering some of its Web services in JSON. Google started offering JSON feeds for its GData web protocol in December 2006.

- **files: 040_json**
- **video: lec 102 - take 01**

Go & JSON - Marshal & Encode

Now we will see how to use Go with JSON. The most important thing to understand is that you can marshal ***OR*** encode Go code to JSON. Regardless of whether or not you use “marshal” or “encode”, your Go data structures will be turned into JSON. So what’s the difference? Marshal is for turning Go data structures into JSON and then assigning the JSON to a variable. Encode is used to turn Go data structures into JSON and then send it over the wire. Both “marshal” and “encode” have their counterparts: “unmarshal” and “decode”. You can learn about Go & JSON at https://godoc.org/encoding/json - Package json implements encoding and decoding of JSON as defined in RFC 4627. The mapping between JSON and Go values is described in the documentation for the Marshal and Unmarshal functions. You can also read about Go & JSON at this Go official blogpost: https://blog.golang.org/json-and-go

- **files: 040_json / 01**
- **video: lec 103 - take 03**

Unmarshal JSON with Go

Unmarshal parses the JSON-encoded data and stores the result in the value pointed to by v. We use unmarshal and decode to take data from JSON and put it into a Go data structure.

- **files: 040_json / 02_object - 07_unmarshal**
- **video: lec 104 - take 07**

Unmarshal JSON with Go using Tags

Unmarshal parses the JSON-encoded data and stores the result in the value pointed to by v. We use unmarshal and decode to take data from JSON and put it into a Go data structure.

- **files: 040_json / 08_unmarshal_tags - 16_hands-on**
- **video: lec 105 - take 06**

Hands-on exercise solution

Here is the solution to the hands-on exercise.

- **files: 040_json / 17_solution**
- **video: lec 106 - take 02**

AJAX introduction

Ajax (also AJAX) is short for asynchronous JavaScript and XML.

Ajax is a set of web development techniques using many web technologies on the client-side to create asynchronous Web applications.

With Ajax, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page.

By decoupling the data interchange layer from the presentation layer, Ajax allows for web pages, and by extension web applications, to change content dynamically without the need to reload the entire page.

In practice, modern implementations commonly substitute JSON for XML due to the advantages of being native to JavaScript.

JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

- **files: 041_ajax / 01**
- **video: lec 107 - take 01**

AJAX server side

Now we will use Go to program our server's response to an AJAX request. Remember, all AJAX is doing is making a request. A request includes some HTTP method and some route. For instance, a request might be "GET /user/score". To handle an AJAX request on the server, we create a func to handle for that specific request; we program what we want the server to send back.

- **files: 041_ajax / 02 - 03**
- **video: lec 108 - take 01**

Go & Mongodb

Organizing code into packages

We are first going to lay groundwork for building an application which works with Mongodb. Our initial groundwork will be to setup a server using Julien Schmidt's router. We will then create functionality to get a user. We will also use packages to organize our code.

- **files: 042_mongodb / 01 - 02**
- **video: lec 109 - take 01**

Create user & delete user

Continuing to build our application, we will now stub out functionality for creating and deleting a user. We will test this functionality with curl.

- **files: 042_mongodb / 03**
- **video: lec 110 - take 01**

MVC design pattern - model view controller

Model–view–controller (MVC) is a software design pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. Traditionally used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications and even mobile, desktop and other clients.One of the seminal insights in the early development of graphical user interfaces, MVC became one of the first approaches to describe and implement software constructs in terms of their responsibilities. Trygve Reenskaug introduced MVC into Smalltalk-76 while visiting the Xerox Palo Alto Research Center (PARC) in the 1970s.

MODEL
The central component of MVC, the model, captures the behavior of the application in terms of its problem domain, independent of the user interface. The model directly manages the data, logic, and rules of the application. ***A model stores data that is retrieved according to commands from the controller and displayed in the view.***

VIEW
A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. ***A view generates new output to the user based on changes in the model.***

CONTROLLER

The third part, the controller, accepts input and converts it to commands for the model or view. ***A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., scrolling through a document).***

Although originally developed for desktop computing, MVC has been widely adopted as an architecture for World Wide Web applications in major programming languages. Several commercial and noncommercial web frameworks have been created that enforce the pattern. These software frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server. Early web MVC frameworks took a thin client approach that placed almost the entire model, view and controller logic on the server. This is still reflected in popular frameworks such as Django, Rails and ASP.NET MVC. In this approach, the client sends either hyperlink requests or form input to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server. As client technologies have matured, frameworks such as AngularJS, EmberJS, JavaScriptMVC and Backbone have been created that allow the MVC components to execute partly on the client using Ajax.

The use of the MVC pattern in web applications exploded in popularity after the introduction of Apple's WebObjects in 1996, which was originally written in Objective-C (that borrowed heavily from Smalltalk) and helped enforce MVC principles. Later, the MVC pattern became popular with Java developers when WebObjects was ported to Java. Later frameworks for Java, such as Spring (released in October 2002), continued the strong bond between Java and MVC. The introduction of the frameworks and Django (July 2005, for Python) and Rails (December 2005, for Ruby), both of which had a strong emphasis on rapid deployment, increased MVC's popularity outside the traditional enterprise environment in which it has long been popular. MVC web frameworks now hold large market-shares relative to non-MVC web toolkits.

- **files: 042_mongodb / 04**
- **video: lec 111 - take 01**

Install mongodb

To install mongodb, go to https://www.mongodb.com/ and find the download area. Select the community server. Choose the operating system of the computer upon which you will install mongodb. Look for a link that offers instructions. Read and follow those instructions. When you have mongodb running, it will listen for connections on port 27017. You will also need to download and install a driver for mongodb using “go get gopkg.in/mgo.v2” and “go get gopkg.in/mgo.v2/bson”

- **files: 042_mongodb / 05_mongodb / 01**

- **video: lec 112 - take 03**

Connect to mongodb

Now we will connect to mongodb.

- **files: 042_mongodb / 05_mongodb / 01**
- **video: lec 113 - take 01**

CRUD with Go & mongodb

Now we will create CRUD with Go & mongodb. CRUD stands for create, read, update, delete. We will create a record, read that record, and delete that record.

- **files: 042_mongodb / 05_mongodb / 02 - 05**
- **video: lec 114 - take 02**

Hands on exercise & solution

Use a map instead of mongodb for your data storage.

- **files: 042_mongodb / 07 - 08**
- **video: lec 115 - take 01**

Hands on exercise & solution

Now persist the data which is in your map. Every time a user is created, or deleted, turn your map into JSON and write it to a file. Also, when your program opens, if there is a file with data in it, load that data into your program.

- **files: 042_mongodb / 09 - 10**
- **video: lec 116 - take 01**

Hands on exercise & solution

Refactor the code in folder “030_sessions / 08_expire-session”. This is the code we created when we studied sessions. Modify our session code in "starting-code" to use these packages: 1. controllers 1. models 1. session Refactor code as necessary to put code into appropriate packages.

- **files: 042_mongodb / 11 - 12**
- **video: lec 117 - take 02**

Docker

Introduction to Docker

Our “golang web dev” course is like a survey course. At the university, a survey course introduces you to a field of study. One field of study will have many areas which merit focus and additional courses. The primary purpose of our course is to learn Golang Web Dev. While learning this, however, we are coming across many areas which merit more courses. AWS, MySQL, and MongoDB all merit multiple courses to become proficient with them. Docker is another one of these areas. It is important for you to know about Docker. This section will introduce you to Docker. You will learn what Docker is and why it’s important. We will get a Docker container running a Go web app. We will also deploy a Docker container on AWS. However, to become a master at Docker, you should take additional courses on Docker.

- **files: 043_docker**
- **video: lec 118 - take 02**

Virtual machines & containers

Back in the day, an application would be created and run on a machine. There was a ratio of 1:1. One application, one machine. When wondering what type of a machine to purchase for their application, developers always purchased the most powerful machine possible. There was no certainty as to how much an application might be in demand. To cover their asses, they errored on the side of

too much power instead of not enough. The result of this: many machines were greatly underutilized. Many machines only ran at a small fraction of their capacity.

Virtual machines found a way to use that capacity. One physical machine could host several virtual machines. Each virtual machine appeared just like a real machine to its user. There was a downside to this, though: each virtual machine had its own operating system. This created license and maintenance costs, as well as using up resources on the physical machine. The question arose: Is there a way to divide up a physical machine, but only have one operating system on that machine?

Linux is built in such a way that you can create separate user spaces. Each user space is in a separate isolated "sandbox" area. Each user space has its own file system and processes. These user spaces are all segregated and separate from each other. Containers leverage this technology of the Linux operating system. A container is just like a virtual machine WITHOUT THE GUEST OPERATING SYSTEM. Each PHYSICAL MACHINE has ONE LINUX OPERATING SYSTEM. Containers use the Linux OS of the physical machine. To the user, each container appears just like a real machine.

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

- **files: 043_docker / 01_about-containers**
- **video: lec 119 - take 05**

Installing docker

Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries. Because your application can always be run with the environment it expects right in the build image, testing and deployment is simpler than ever, as your build will be fully portable and ready to run as designed in any environment. And because containers are lightweight and run without the extra load of a hypervisor, you can run many applications that all rely on different libraries and environments on a single kernel, each one never interfering with the other. This allows you to get more out of your hardware by shifting the "unit of scale" for your application from a virtual or physical machine, to a container instance.

- **files: 043_docker / 02_install**
- **video: lec 120 - take 01 PART 01 & PART 02**

Docker whalesay example

People all over the world create Docker images. You can find these images by browsing the Docker Hub. In this next section, you'll search for and find the whalesay image. Docker hub is a place to store, distribute, and manage Docker images. Docker hub is a registry, or repository, of images. Docker Hub contains images from individuals like you and official images from organizations like RedHat, IBM, Google, and a whole lot more.

- **files: 043_docker / 03_run-images**
- **video: lec 121 - take 05**

Using a Dockerfile to build an image

The whalesay image could be improved. It would be nice if you didn't have to think of something to say and you didn't need to type as much to get whalesay to talk. In this video, we will improve the whalesay image by building a new version that "talks on its own" and requires fewer words to run.

- **files: 043_docker / 04_build-image**
- **video: lec 122 - take 01**

Launching a container running curl

We will review what we learned in the last video by creating a Dockerfile to build an image from which we can then run a container. In this example, however, we will launch an interactive terminal with which to interact with the application running in our container. From this terminal, we will run a "curl" command.

- **files: 043_docker / 05_curl**
- **video: lec 123 - take 01**

Running a Go web app in a Docker container

We will create a Dockerfile to build an image. This image will hold a Go web app. We will then launch a container from that image. Our container will run our Go web app.

- **files: 043_docker / 06_hello-go**

- **video: lec 124- take 02**

Pushing & pulling to docker hub

When we have an image that we like, it is a good idea to store that image. We can store images on Docker hub. Once an image is on Docker hub, we can also pull that image down to use it.

- **files: 043_docker / 07_push-pull**
- **video: lec 125 - take 01**

Go, Docker & Amazon Web Services (AWS)

We can create a Docker image running a Go web app, then deploy a container from that image to Amazon Web Services (AWS).

- **files: 043_docker / 08_aws-docker**
- **video: lec 126 - take 01**

PostgreSQL

Installing postgres

PostgreSQL, often simply Postgres, is a relational database with an emphasis on extensibility and standards compliance. As a database server, its primary function is to store data securely, and to allow for retrieval at the request of other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications (or for data warehousing) with many concurrent users. PostgreSQL is ACID-compliant and transactional. PostgreSQL has updatable views and materialized views, triggers, foreign keys; supports functions and stored procedures, and other expandability. PostgreSQL is developed by the PostgreSQL Global Development Group, a diverse group of many companies and individual contributors. It is free and open-source software, released under the terms of the PostgreSQL License, a permissive free-software license.

- **files: 044_postgresql / 01_install**
- **video: lec 127 - take 01**

Create database

We create a database at the command line in postgres with the CREATE DATABASE command. We list the databases with the \l command. We connect to a database with the \c <database name> command. We switch back to postgres database with the \c postgres command. We see the current user with the SELECT current_user; command. We see the current database with the SELECT current_database(); command. We drop (remove, delete) a database with the DROP DATABASE <database name>; command.

- **files: 044_postgresql / 02_create-database**
- **video: lec 128 - take 01**

Create table

We create a table in postgres with the CREATE TABLE statement.

- **files: 044_postgresql / 03_create-table**
- **video: lec 129 - take 01**
-

Insert records

We insert into a database with the INSERT INTO statement.

- **files: 044_postgresql / 04_insert-record**
- **video: lec 130 - take 01**

Auto increment primary key

We automatically increment a field in a table with the serial data type in postgres.

- **files: 044_postgresql / 05_auto-increment**
- **video: lec 131 - take 01**

Hands-on exercise

Delete all of your current tables. Create a new table called employees with these fields id, name, score, salary AND give score a default value of 10 AND have the id field automatically increment. Add records and then show all of the records.

- **files: 044_postgresql / 06_hands-on**
- **video: lec 132 - take 01**

Hands-on exercise - solution

This is my solution to the hands-on exercise.

- **files: 044_postgresql / 07_solution**
- **video: lec 133 - take 01**

Relational databases

A relational database is a digital database whose organization is based on the relational model of data, as proposed by E. F. Codd in 1970. The various software systems used to maintain relational databases are known as a relational database management system (RDBMS). Virtually all relational database systems use SQL (Structured Query Language) as the language for querying and maintaining the database.

- **files: 044_postgresql / 08_relational-dbs**
- **video: lec 134 - take 01**

Query - cross join

A “query” is a question we ask of the database. When we ask the question, results are returned (including zero results). We use Structured Query language to ask questions of the database. SQL is used for managing data held in a relational database

management system (RDBMS). When we do a “cross join” we are getting the Cartesian product of rows from tables in the join. In other words, it will combine each row from the first table with each row from the second table.

- **files: 044_postgresql / 09_queries_cross-join**
- **video: lec 135 - take 03**

Query - inner join

An inner join allows us to select records from two tables. When we do an inner join, we must specify ON what field the tables are connected.

- **files: 044_postgresql / 10_queries_inner-join**
- **video: lec 136 - take 04**

Query - three table inner join

We can chain together an INNER JOIN between multiple tables.

- **files: 044_postgresql / 10_queries_inner-join**
- **music: “Love Of All” by http://www.twinmusicom.org/**
- **video: lec 137 - take 04**

Query - outer joins

Outer joins will combine tables together, and also include records which are not connected to the other tables. A left outer join gives you everything in one table, and also the matching records in another table. For tables A and B a left outer join would give you all rows of the "left" table (A), even if the join-condition does not find any matching row in the "right" table (B). This means that if the ON clause matches 0 (zero) rows in B (for a given row in A), the join will still return a row in the result (for that row)—but with NULL in each column from B.

- **files: 044_postgresql / 11_queries_outer-join**
- **video: lec 138 - take 01**

Clauses

Clauses allow us to modify our queries and include commands like WHERE, AND, IN, NOT, BETWEEN, IS NOT NULL, LIKE and OR

- **files: 044_postgresql / 12_clauses**
- **video: lec 139 - take 02**

Update a record

Once we have a record in our database, we will also want to update that record. We can update a record with an UPDATE command.

- **files: 044_postgresql / 13_update**
- **video: lec 140 - take 01**

Delete a record

To delete a record from a table we use the DELETE command.

- **files: 044_postgresql / 14_delete**
- **video: lec 141 - take 01**

Users - create, grant, alter, remove

We can create users in our database. We can grant them privileges. We can alter users. And we can remove users. This video also shows you how to find, and read, Postgres documentation.

- **files: 044_postgresql / 15_users**
- **video: lec 142 - take 03**

Go & postgres

To use postgres in our Go code, we will need to use package database/sql from the standard library. We will also need a driver for our postgres database. We will then create a user, connect to our database, then test the connection.

- **files: 044_postgresql / 16_go-postgres**
- **video: lec 143 - take 02**

Select query

We will put records into our database, the use a SELECT statement from our Go program to retrieve those records.

- **files: 044_postgresql / 17_select**
- **video: lec 144 - take 01**

Web app

We will now take the code from the previous example and modify it so that the results of our query can be served back from a web server.

- **files: 044_postgresql / 18_routing**
- **video: lec 145 - take 01**

Query Row

We can use QueryRow from the database/sql package to only select one row from a database.

- **files: 044_postgresql / 19_where-clause**
- **video: lec 146 - take 04**

Insert record

We insert records using the Exec method from package sql.

- **files: 044_postgresql / 20_insert**
- **video: lec 147 - take 01**

Update record

We update records using the Exec method from package sql.

- **files: 044_postgresql / 21_update**
- **video: lec 148 - take 01**

Delete record

We update records using the Exec method from package sql.

- **files: 044_postgresql / 22_delete**
- **video: lec 149 - take 01**

Code organization

This is ONE OF THE MOST IMPORTANT VIDEOS in the entire training: it will show you how to organize your code in a go web app project.

- **files: 045-code-organization**
- **video: lec 150 - take 01**

MongoDB

NoSQL

A NoSQL (originally referring to "non SQL", "non relational" or "not only SQL") database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century, triggered by the needs of Web 2.0 companies such as Facebook, Google, and Amazon.com. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages. Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), and finer control over availability. The data structures used by NoSQL databases (e.g. **key-value, wide column, graph, or document**) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables. Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability, partition tolerance, and speed. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages (instead of SQL, for instance the lack of ability to perform ad-hoc JOINs across tables), lack of standardized interfaces, and huge previous investments in existing relational databases. Most NoSQL stores lack true ACID transactions, although a few databases, such as MarkLogic, Aerospike, FairCom c-treeACE, Google Spanner (though technically a NewSQL database), Symas LMDB, and OrientDB have made them central to their design. Instead, most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes "eventually" (typically within milliseconds) so queries for data might not return updated data immediately or might result in reading data that is not accurate, a problem known as stale reads. Additionally, some NoSQL systems may exhibit lost writes and other forms of data loss. Fortunately, some NoSQL systems provide concepts such as write-ahead logging to avoid data loss. For distributed transaction processing across multiple databases, data consistency is an even bigger challenge that is difficult for both NoSQL and relational databases. Even current relational databases "do not allow referential integrity constraints to span databases."

- **files: 00_about-nosql**
- **video: lec 151 - take 01**

MongoDB

MongoDB (from humongous) is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents. MongoDB is developed by MongoDB Inc. and is free and open-source, published under a combination of the GNU Affero General Public License and the Apache License.

- **files: 00_about-nosql**
- **video: lec 152 - take 01**

Installing mongo

To install mongo, you will need to download the mongodb application from mongodb's website. For windows, this will be an executable file. For Mac and Linux, you will use curl to download a tar zipped filed. You will then need to set a path environment variable pointing to the bin folder where the applications mongod and mongo are located. You will also need to create a data directory "data/db/" at either the root of your C drive on windows or at the root of your computer's drive on mac and linux. Once that is all done, restart your terminal. Launch one terminal and run mongod. Launch another terminal and run mongo.

- **files: 01_install**
- **video: lec 153 - take 02**

Database

How to create a database in mongodb: we use the "use" word to use a current database or create one if the database reqeusted doesn't exist.

- **files: 044_postgresql / 02_db**
- **video: lec 154 - take 01**

Collection

A collection can be created implicitly or explicitly.

- **files: 044_postgresql / 03_collection**

- **video: lec 155 - take 01**

Document

MongoDB uses JSON-like documents. What this means is that basically you just take a bunch of JSON and store it in mongodb.

- **files: 044_postgresql / 04_document**
- **video: lec 156 - take 02**

Find (aka, query)

This video will show you how to find data in mongo (eg, do a query).

- **files: 044_postgresql / 05_query**
- **video: lec 157 - take 02**

Update

How to update a document in mongodb.

- **files: 044_postgresql / 06_update**
- **video: lec 158 - take 02**

Remove

How to remove (delete) a document in mongodb.

- **files: 044_postgresql / 07_remove**
- **video: lec 159 - take 02**

Projection

Mongodb projection is retrieving part of a document; only some of the fields.

- **files: 044_postgresql /**
- **video: lec 160 - take 01**

Limit

We can limit the number of documents returned by a find in mongo.

- **files: 044_postgresql / 09_limit**
- **video: lec 161 - take 02**

Sort

We can sort the documents returned by a find in mongo.

- **files: 044_postgresql / 10_sort**
- **video: lec 162 - take 02**

Index

This is how you add an index to a field in a document in mongodb.

- **files: 044_postgresql / 11_index**
- **video: lec 163 - take 03**

Aggregation

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function, and single purpose aggregation methods.

- **files: 044_postgresql / 12_aggregate**
- **video: lec 164 - take 05**

Users

How to add permissions to users in mongo; how to launch your database in secure mode.

- **files: 044_postgresql / 14_users**
- **video: lec 165 - take 08**

JSON

Displaying JSON from our previous postgres book example.

- **files: 044_postgresql / 15_postgres**
- **video: lec 166 - take 01**

Create Read Update Delete (CRUD)

The holy grail of web programming: connecting a database to our web app and being able to create, read, update, and delete a record. Here we achieve CRUD with Go and Mongo.

- **files: 044_postgresql / 16_go-mongo**
- **video: lec 167 - take 06**

More to come ...

Google Cloud section coming in a few weeks

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**

- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

XX

xx

- **files: 044_postgresql /**
- **video: lec xxx - take 01**

xxDocker, Kubernetes, Microservices

Google Cloud lets you build and host applications and websites, store data, and analyze data on Google's scalable infrastructure. **Google App Engine** runs on Google cloud. Google App Engine, which is often referred to as App Engine, is a platform as a service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications—as the number of requests increases for an application, App Engine automatically allocates more resources for the web application to handle the additional demand. Google App Engine is free up to a certain level of consumed resources. Fees are charged for additional storage, bandwidth,

or instance hours required by the application. It was first released as a preview version in April 2008 and came out of preview in September 2011.

You most important decision is choosing your stack and the platform upon which it is deployed.

By choosing Go, you have made the best decision possible for your server-side language, but where should your application run? Should it run on Google, AWS, or Azure? And should you use IAAS or PAAS?

Whichever cloud environment you choose, you are committing to that environment. Your code will be intertwined with that environment, and it will be hard to switch away from that provider.

You can mitigate your commitment to a cloud provider by using containers, like Docker, and container management like Kubernetes. Another interesting new trend is microservices. You should explore containers, Docker, Kubernetes, and microservices before choosing how you will build and deploy your application.

Here are a few introductory videos which are worth watching:

Docker
https://www.youtube.com/watch?v=aLipr7tTuA4

Kubernetes
https://www.youtube.com/watch?v=of45hYbkIZs

Microservices
https://www.youtube.com/watch?v=CKL3fV5UR8w

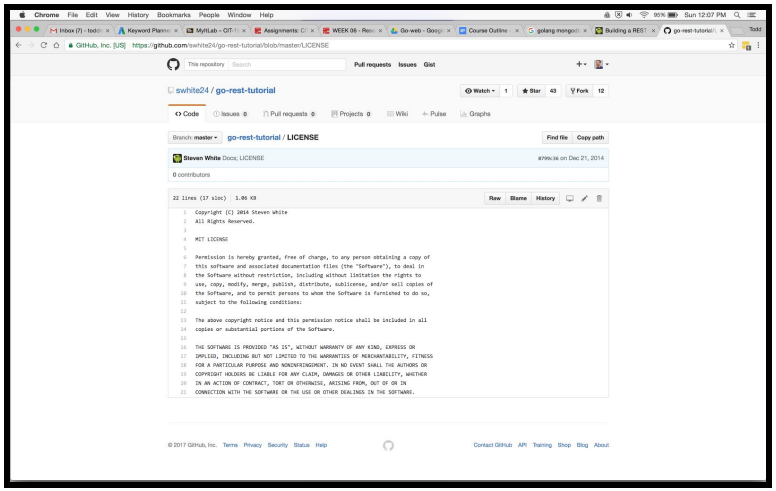
- **files: 043_google-cloud / 01**
- **video: lec xxx - take 01**

Hello world

Google Cloud lets you build and host applications and websites, store data, and analyze data on Google's scalable infrastructure. **Google App Engine** runs on Google cloud. Google App Engine, which is often referred to as App Engine, is a platform as a service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications—as the number of requests increases for an application, App Engine automatically allocates more resources for the web application to handle the additional demand. Google App Engine is free up to a certain level of consumed resources. Fees are charged for additional storage, bandwidth, or instance hours required by the application. It was first released as a preview version in April 2008 and came out of preview in September 2011.

- **files: 043_google-cloud / 01**
- **video: lec xxx - take 01**

https://stevenwhite.com/building-a-rest-service-with-golang-1/
https://stevenwhite.com/building-a-rest-service-with-golang-2/
https://stevenwhite.com/building-a-rest-service-with-golang-3/


Chrome File Edit View History Bookmarks People Window Help
Sun 12:07 PM
GitHub, Inc. [US] https://github.com/swhite24/go-rest-tutorial/blob/master/LICENSE
This repository Search
Pull requests Issues Gist
swhite24 / go-rest-tutorial
Watch 1 Star 43 Fork 12
Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs
Branch: master go-rest-tutorial / LICENSE
Find file Copy path
Steven White Docs; LICENSE
a7ffdc1a on Dec 21, 2014
0 contributors
22 lines (17 sloc) 1.06 KB
Raw Blame History
Copyright (C) 2014 Steven White
All Rights Reserved.
MIT LICENSE
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
© 2017 GitHub, Inc. Terms Privacy Security Status Help
Contact GitHub API Training Shop Blog About