Server-Side Swift
Swift MN

Frameworks available[1]

- Perfect ⭐ 11,427
- Vapor ⭐ 9,466
- Kitura ⭐ 5,682
- Zewo ⭐ 1651

[1] Github Stars as of 2017/5/15

WHY I CHOSE VAPOR:

- I read through the docs for both
- I joined both Slack groups
- Vapor felt more interesting to me ¯_(ツ)_/¯
- Interesting article comparing the communities of Perfect and Vapor
  - https://www.sitepoint.com/server-side-swift-comparing-vapor-perfect

Getting started with Vapor

Install Vapor (macOS)

- Add Homebrew Tap

```
brew tap vapor/homebrew-tap
brew update
```

- Install Vapor

```
brew install vapor
```

- Verify Installation

```
vapor --help
```

Starting a New Project

- Create the App

```
vapor new SwiftMN
cd SwiftMN
```

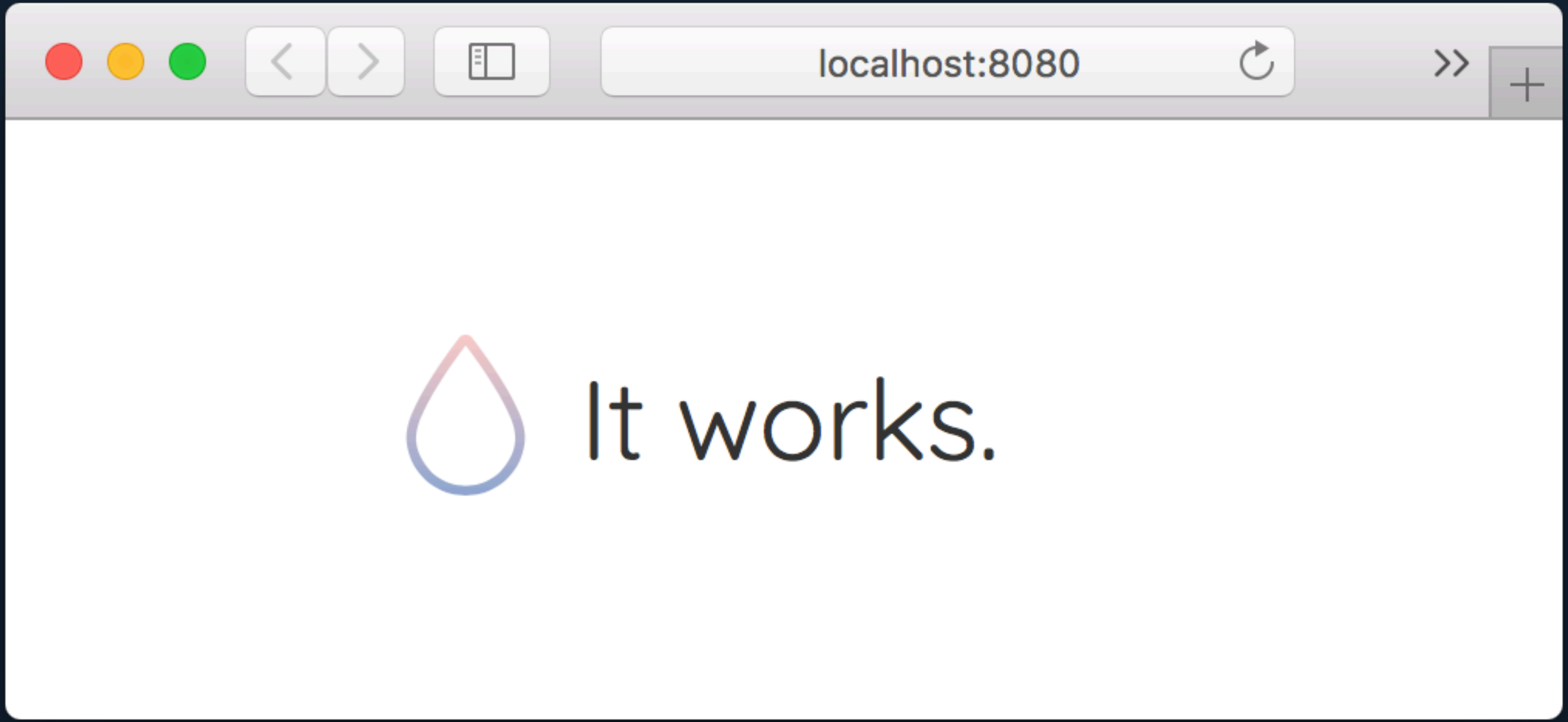
- Build the app

```
vapor build
```

- Run the app

```
.build/debug/App
```

Open a browser and navigate to localhost:8080

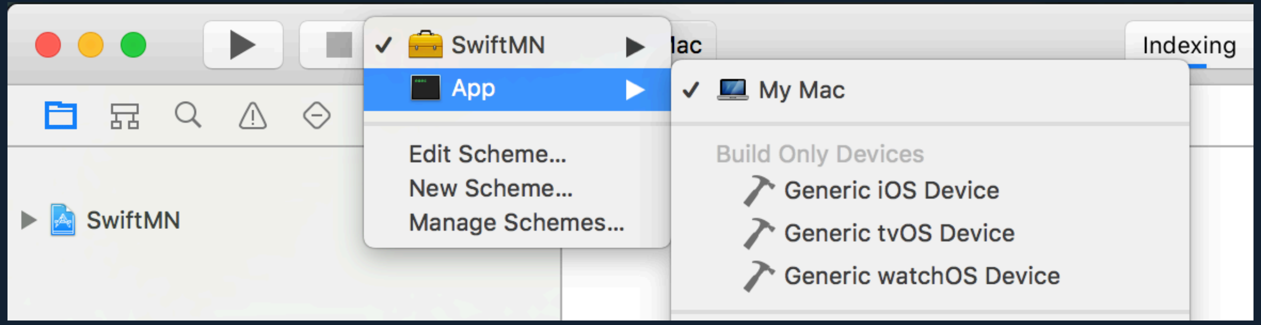


It works.

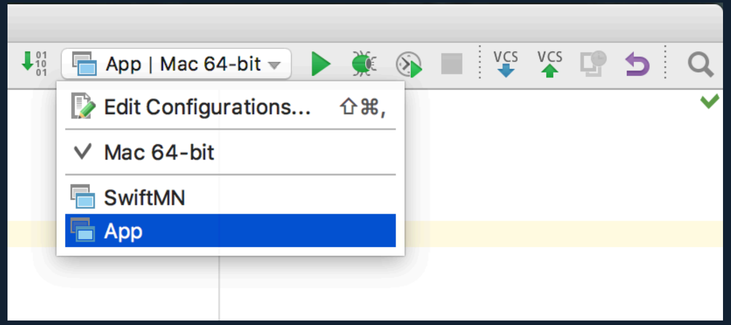
DEBUGGING

Make sure you change your scheme to App

Xcode



AppCode



A NOTE ABOUT THE DOCS:

They're all over the place and incomplete

- ~~docs.vapor.codes~~
- ~~beta.docs.vapor.codes~~
- docs.vapor.codes/2.0/
- api.vapor.codes
- read the code github.com.vapor/vapor
- ask on Slack

BUILDING THE APP

main.swift

```
Droplet()
```

The Droplet is a service container that gives you access to many of Vapor's facilities. It is responsible for registering routes, starting the server, appending middleware, and more.

main.swift

```swift
import Vapor
let drop = try Droplet()
try drop.run()
```

SIMPLE ROUTES

```swift
import Vapor
let drop = try Droplet()

drop.get { req in
    return try drop.view.make("welcome", [
        "message": drop.localization[req.lang, "welcome", "title"]
    ])
}

try drop.run()
```

Simple Routes

```
    drop.get { req in
```

All GET requests to "/" will execute the block.

Simple Routes

```
drop.get { req in
    return try drop.view.make("welcome", [
```

The Droplet will try to build and return welcome.leaf

SIMPLE ROUTES

```swift
public func make(_ path: String) throws -> View {
    return try make(path, Node.null)
}

public func make(_ path: String, _ context: NodeRepresentable) throws -> View {
    return try make(path, try context.makeNode())
}

public func make(_ path: String, _ context: [String: NodeRepresentable]) throws -> View {
    return try make(path, try context.makeNode())
}
```

Simple Routes

```
drop.get { req in
    return try drop.view.make("welcome", [
        "message": drop.localization[req.lang, "welcome", "title"]
    ])
}
```

We pass a localized message in our context object

SIMPLE ROUTES

The verbs are Variadic functions

```
drop.get("about") { req in
    // handle GET requests to "/about"
}

drop.post("some", "other", "place") { req in
    // handle POST requests to /some/other/place
}

drop.get("anything", "*") { req in
    // wildcards match anything after /anything
}
```

SIMPLE ROUTES

Path Parameters

```
drop.put("events", ":id") { req in
    guard let id = req.parameters["id"]?.string else {
        throw Abort.badRequest
    }
    // Update the event with the given id
}
```

SIMPLE ROUTES

TypeSafe Path Parameters

```
drop.put("events", Int.self) { req, id in
    // Update the event with the given id
}

drop.put("events", Event.self) { req, event in
    // Update the given event
}
```

CONTROLLERS

ROUTING GROUPS

```
drop.group("events") { route in
    let eventController = MeetupController(drop: drop)
    route.get(handler: eventController.listEvents)
    route.get(":id", handler: eventController.getEvent)
}
```

All requests to /events will execute this closure

Routing Groups

```
    drop.group("events") { route in
```

All requests to /events should execute this closure

Routing Groups

```
    drop.group("events") { route in
        let eventController = MeetupController(drop: drop)
```

Create a Controller to handle each request

ROUTING GROUPS

```
drop.group("events") { route in
    let eventController = MeetupController(drop: drop)
    route.get(handler: eventController.listEvents)
    route.get(":id", handler: eventController.getEvent)
}
```

GET requests to "/events" will hit the listEvents function

GET requests to "/events/:id" will hit the getEvent function

MeetupController

```swift
func listEvents(_ request: Request) throws -> ResponseRepresentable {
    let events: [Event] = try fetchEvents(request)

    var upcoming: [Event] = []
    var past: [Event] = []

    events.forEach {
        switch $0.status {
            case .upcoming: upcoming.append($0)
            case .past: past.append($0)
        }
    }

    return try drop.view.make("listEvents", [
        "allEvents": events.makeNode(),
        "upcomingEvents": upcoming.makeNode(),
        "pastEvents": past.makeNode()
    ])
}
```

MeetupController

```
private func fetchEvents(_ request: Request) throws -> [Event] {
    let path = "https://api.meetup.com/SwiftMN/events"
    let headers: [HeaderKey: String] = [
        HeaderKey.contentType: "application/json"
    ]
    let query: [String: CustomStringConvertible] = [
        "status": "upcoming,past",
        "desc": true
    ]

    // synchronous request to the meetup API
    let eventsResponse = try drop.client.get(path, headers: headers, query: query)

    ... // a lot of parsing and data conversion happens

    return events
}
```

MeetupController

```swift
func listEvents(_ request: Request) throws -> ResponseRepresentable {
    let events: [Event] = try fetchEvents(request)

    var upcoming: [Event] = []
    var past: [Event] = []

    events.forEach {
        switch $0.status {
            case .upcoming: upcoming.append($0)
            case .past: past.append($0)
        }
    }

    return try drop.view.make("listEvents", [
        "allEvents": events.makeNode(),
        "upcomingEvents": upcoming.makeNode(),
        "pastEvents": past.makeNode()
    ])
}
```

LEAF

base.leaf

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="/styles/app.css">
        #import("head")
        #embed("bootstrap")
    </head>
    <body>
        #import("body")
    </body>
</html>
```

listEvents.leaf

```
#extend("base")

#export("head") {
    <title>SwiftMN Events</title>
}

#export("body") {
    ...
}
```

Inside our #export("body") { ... }

```
<div class=row>
    #loop(upcomingEvents, "event") {
        <div class="col-xs-6 col-lg-2">
            #(event.formattedDate)
        </div>
        <div class="col-xs-6 col-lg-10">
            <a href="/events/#(event.id)">#(event.name)</a></li>
        </div>
    }
</div>
```

#loop

```
#loop(upcomingEvents, "event") {
    ...
}
```

#loop iterates over the upcomingEvents from our context object and provides us with a variable named event

MeetupController.listEvents

A reminder of where upcomingEvents came from

```
    return try drop.view.make("listEvents", [
        "allEvents": events.makeNode(),
        "upcomingEvents": upcoming.makeNode(),
        "pastEvents": past.makeNode()
    ])
```

```
#(variable)
```

```
    #(event.formattedDate)
```

Display the formattedDate variable on our event

```
    <a href="/events/#(event.id)">#(event.name)</a></li>
```

Build an achor tag using event.id in the url and event.name as the link text

Inside our #export("body") { ... }

```
<div class=row>
    #loop(upcomingEvents, "event") {
        <div class="col-xs-6 col-lg-2">
            #(event.formattedDate)
        </div>
        <div class="col-xs-6 col-lg-10">
            <a href="/events/#(event.id)">#(event.name)</a></li>
        </div>
    }
</div>
```

VAPOR.SWIFT.MN

Built-in tags

- build on top of an existing leaf

```
#extend("base")
```

- import code from an extended leaf

```
#import("template")
```

- export code to the leaf that you've extended

```
#export("template") { <a href="#()"></a> }
```

- embed another document

```
#embed("commonCSS")
```

BUILT-IN TAGS

- variables

  ```
  #(event.name)
  ```

- literal "#" character

  ```
  #()
  ```

- equality checking

  ```
  #equal(thisVar, thatVar) {
      thisVar and thatVar are equal 👏
  ```

Built-in Tags

» `if / else if / else`

```
#if(entering) {
    Hello, there!
} ##if(leaving) {
    Goodbye!
} ##else() {
    I've been here the whole time.
}
```

Built-in Tags

- iterate over an array

  ```
  #loop(friends, "friend") { <li>#(friend.name)</li> }
  ```

- grab a single item out of an array using it's index

  ```
  #index(events, 0)
  ```

- grab a single item out of a Dictionary

  ```
  #index(friends, "best")
  ```

Built-in tags

- render as html/css/js instead of as a leaf document

```
#raw() {
    <a href="#raw">Anything goes!@#$%^&*</a>
}
```

- render an html string stored in a variable

```
#raw(event.description)
```

Custom Tags

```
class Index: BasicTag {
let name = "index"

func run(arguments: [Argument]) throws -> Node? {
    guard
        arguments.count == 2,
        let array = arguments[0].value?.nodeArray,
        let index = arguments[1].value?.int,
        index < array.count
    else { return nil }
        return array[index]
    }
}
```

main.swift

After conforming to the BasicTag protocol, register your tag in main.swift

```
if let leaf = drop.view as? LeafRenderer {
    leaf.stem.register(Index())
}
```

MIDDLEWARE

Middleware

```
public protocol Middleware {
    func respond(to request: Request, chainingTo next: Responder) throws -> Response
}
```

AuthMiddleware

```swift
func respond(to request: Request, chainingTo next: Responder) throws -> Response {
    guard let bearer = request.auth.header?.bearer?.string else {
        throw Abort.custom(status: .unauthorized, message: "Not Authorized")
    }

    // throws if not authenticated
    let token = try validateToken(bearer)

    request.storage["token"] = token

    return try next.respond(to: request)
}
```

```swift
extension Request {
    func token() throws -> AuthToken {
        guard let token = request.storage["token"] as? AuthToken else {
            throw notAuthorizedError
        }
        return token
    }
}

// anywhere after AuthMiddleware
let token = try request.token()
```

main.swift

```swift
let secure = drop.grouped(AuthMiddleware())

secure.group("user") { route in
    // everything here is secured
}
```

EtagMiddleware

```
func respond(to request: Request, chainingTo next: Responder) throws -> Response {
    if checkEtag(request.headers[.ifNoneMatch]) {
        return Response(status: .notModified)
    }
    return try next.respond(to: request)
}
```

main.swift

```swift
let secure = drop.grouped(AuthMiddleware(), EtagMiddleware())

secure.group("user") { route in
    // everything here is secured
}
```

SHIP IT

Install Vapor on your server

```
docs.vapor.codes v1.5

docs.vapor.codes v2.0
```

Install Vapor on your server

» You'll need curl

```
apt install curl
```

» Easily add Vapor's APT repo with this handy script

```
eval "$(curl -sL https://apt.vapor.sh)"
```

» Install Swift and Vapor

```
sudo apt-get install swift vapor
```

» Double check the installation was successful

```
eval "$(curl -sL check2.vapor.sh)"
```

BUILD

» clone the repo

```
cd /home/ubuntu
git clone https://github.com/vlaminck/SwiftMN.git
cd SwiftMN
```

» Update Swift Packages

```
swift package update
```

» build for release

```
vapor build --release
```

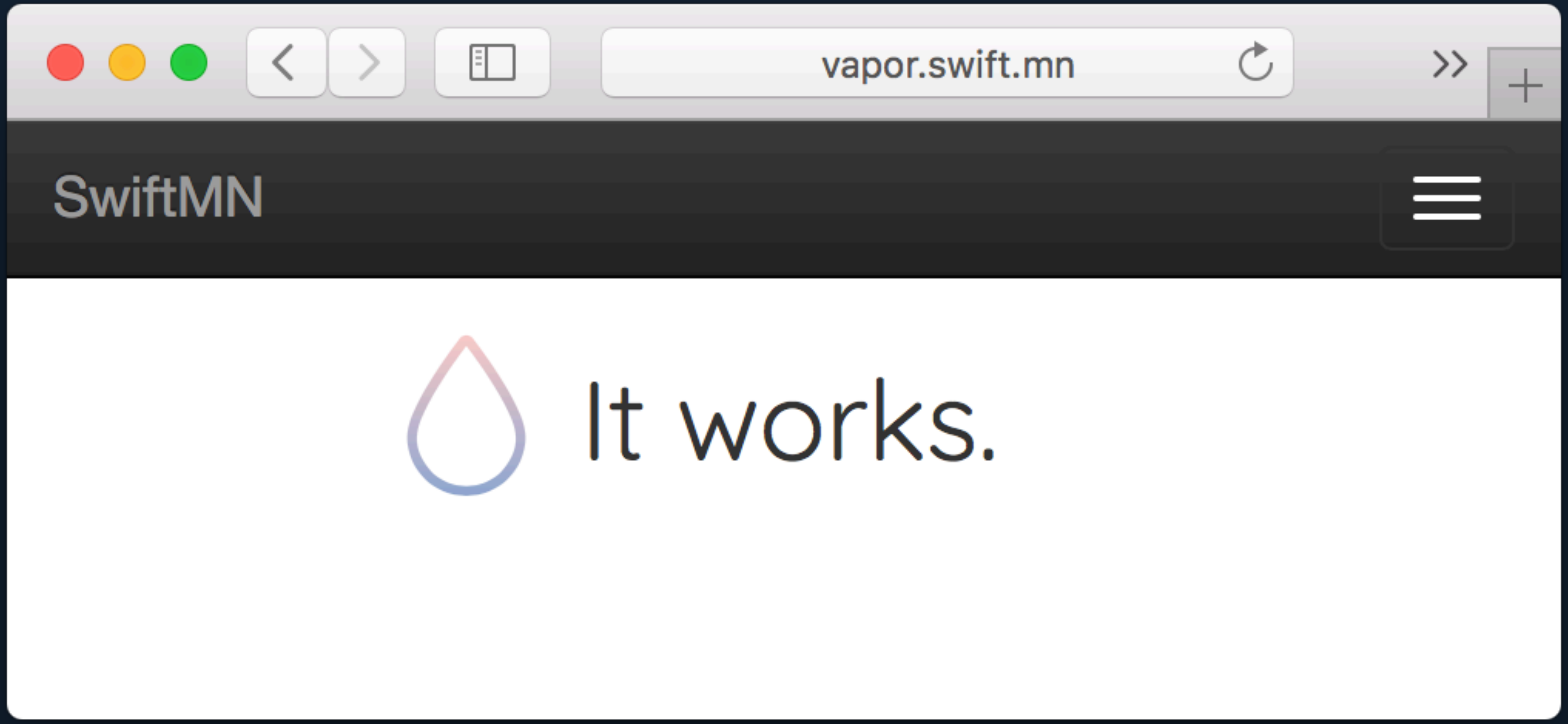
Verify

- Run your app manually to verify that it works

```
.build/release/App --env=production
```

VERIFY

» Open a browser and navigate to your server



Your app as a system service

```
sudo mv etc/app.service /lib/systemd/system
sudo chown root:root /lib/systemd/system/app.service
sudo systemctl daemon-reload
systemctl status app
```

Auto reloading

```
sudo systemctl enable app
sudo systemctl restart app
```

Auto Deployments

    // TODO:

» 3rd party options
  » Flock

Quick Recap

- Available frameworks
- Getting started with vapor
- Routing
- Controllers / meetup API
- Leaf (templating engine)
- vapor.swift.mn
- Manual Deployments

NEXT STEPS

- More meetup API integration
- talk suggestions
  - 👍
- Anything you want
  - github.com/SwiftMN

Next Month

WWDC Micro Talks

SLACK.SWIFT.MN