Up to date for iOS 11, Xcode 9 & Swift 4

Server Side Swift with Vapor

FIRST EDITION

Building Web APIs and Web Apps in Swift

By the raywenderlich.com Tutorial Team

Tim Condon, Tanner Nelson,
Jonas Schwartz & Logan Wright

Table of Contents: Overview

Table of Contents: Extended

About This Book Sample

Whether you're looking to create a backend for your iOS app, or want to create fully-featured web apps, Vapor is the perfect platform for you.

The *Server Side Swift with Vapor* book starts with the basics of web development and introduces the basics of Vapor; it then walks you through creating APIs and web backends; creating and configuring databases; deploying to Heroku, AWS, or Docker; testing your creations and more!

This book sample contains Chapter 2 where you'll start by installing the Vapor Toolbox, then use it to build and run your first project. It continues by teaching you about routing, accepting data and returning JSON.

We hope that this hands-on look inside the book will give you a good idea of what's available in the full version and show you why Vapor is an exciting proposition for your server-side code.

The full book is available for purchase at:

- https://store.raywenderlich.com/products/server-side-swift-with-vapor

Enjoy!

— Tim, Tanner, Jonas, Logan, and the *Server Side Swift with Vapor* team

Server Side Swift with Vapor

By Tim Condon, Tanner Nelson, Jonas Schwartz & Logan Wright

Notice of Rights

Notice of Liability

Trademarks

Server Side Swift with Vapor

About the Authors

Tim Condon is a software engineer who has worked in most areas of the industry, including security, back-end, front-end and mobile! He is the founder of Broken Hands, a company specializing in Vapor and also currently works for the BBC on their mobile team. On Twitter he can be found sporadically tweeting @0xTim. You can find more about him at www.brokenhands.io.

Jonas Schwartz had a long career as a backend and server operations developer, before he teamed up with Tanner and Logan in 2017. He came onboard after talks about building a cloud platform. Since then, his focus has been on making hosting easy and more scalable for Vapor applications.

Logan Wright began his career as an iOS Developer working on many categories of applications from navigation, to customized bluetooth communication protocols. Always a major supporter of OSS, Logan met Tanner through the Vapor project. Eventually, that grew into a full-time position and the community as we know it today.

Tanner Wayne Nelson is an American software engineer based in New York City. He started programming in elementary school and went on to study Computer Science at New York University. Tanner created Vapor in 2016 and now works full time maintaining the core framework and the dozens of packages around it.

About the Editors

Richard Critz did double duty as editor and tech editor for this book. He is the iOS Team Lead at raywenderlich.com and has been doing software professionally for nearly 40 years, working on products as diverse as CNC machinery, network infrastructure, and operating systems. He discovered the joys of working with iOS beginning with iOS 6. Yes, he dates back to punch cards and paper tape. He's a dinosaur; just ask his kids. On Twitter, while being mainly read-only, he can be found @rcritz. The rest of his professional life can be found at www.rwcfoto.com.

Darren Ferguson is the final pass editor for this book. He's an experienced software developer and works for M.C. Dean, Inc, a systems integration provider from North Virginia. When he's not coding, you'll find him enjoying EPL Football, traveling as much as possible and spending time with his wife and daughter. Find Darren on Twitter at @darren102.

Book License

By purchasing *Server Side Swift with Vapor*, you have the following license:

- You are allowed to use and/or modify the source code in *Server Side Swift with Vapor* in as many apps as you want, with no attribution required.
- You are allowed to use and/or modify all art, images and designs that are included in *Server Side Swift with Vapor* in as many apps as you want, but must include this attribution line somewhere inside your app: "Artwork/images/designs: from *Server Side Swift with Vapor*, available at www.raywenderlich.com".
- The source code included in *Server Side Swift with Vapor* is for your personal use only. You are NOT allowed to distribute or sell the source code in *Server Side Swift with Vapor* without prior authorization.
- This book is for your personal use only. You are NOT allowed to sell this book without prior authorization, or distribute it to friends, coworkers or students; they would need to purchase their own copies.

All materials provided with this book are provided on an "as is" basis, 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.

All trademarks and registered trademarks appearing in this guide are the properties of their respective owners.

What You Need

To follow along with this book, you'll need the following:

- **Swift 4.1** Vapor 3 requires Swift 4.1 minimum in both Xcode and from the command line.
- **Xcode 9 or later.** Xcode is the main development tool for writing code in Swift. You need Xcode 9.4 at a minimum, since that version includes Swift 4.1. You can download the latest version of Xcode for free from the Mac App Store.

If you haven't installed the latest version of Xcode, be sure to do that before continuing with the book. The code covered in this book depends on Swift 4.1 and Xcode 9 — you may get lost if you try to work with an older version.

This book provides the building blocks for developers who wish to use Vapor to create server-side Swift applications. It shows you how to take the familiar type-safe, compiler-driven world of Swift you know from iOS and use it on the server.

The only prerequisites for this book are an intermediate understanding of Swift and iOS development. If you've worked through our classic beginner books — *Swift Apprentice* https://store.raywenderlich.com/products/swift-apprentice and *iOS Apprentice* https://store.raywenderlich.com/products/ios-apprentice — or have similar development experience, you're ready to read this book.

Chapter 2: Hello Vapor

By Tim Condon

Beginning a project using a new technology can be daunting. Vapor makes it easy to get started. It even provides handy scripts to make sure your computer is configured correctly.

In this chapter, you'll start by installing the Vapor Toolbox, then use it to build and run your first project. You'll finish by learning about routing, accepting data and returning JSON.

Vapor Toolbox

The Vapor Toolbox is a command line interface (CLI) tool that you use when developing Vapor apps. It provides several features, including:

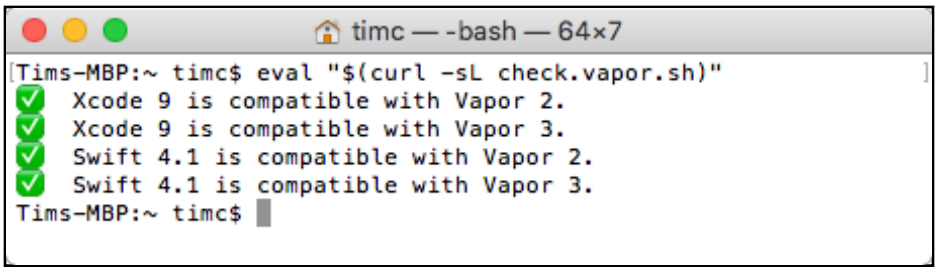
- Creating new apps from templates.
- Building and running projects using the Swift toolchain.
- Generating Xcode projects.
- Deploying projects with Vapor Cloud.

Before you can install the toolbox, you need to ensure your system is compatible. To do this, you run the Vapor Check script, which verifies you have the necessary version of Swift installed.

Open **Terminal**, and execute the following command:

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

This command also works on Linux.



If you receive errors when running this script, you'll need to install Swift. On macOS, simply install Xcode from the Mac App Store. On Linux, use APT to install it as described below.

> Vapor 3 requires Swift 4.1, both in Xcode and from the command line.

Installing on macOS

Vapor uses Homebrew to install the Toolbox.

> If you don't have Homebrew installed, visit https://brew.sh and run the installation command.

In **Terminal** run the following command:

```
brew install vapor/tap/vapor
```

Installing on Linux

This book focuses primarily on using Xcode and macOS for developing your apps. However, everything you build with Vapor will work on versions of Linux that Swift supports. At the time of writing, these are Ubuntu 14.04, Ubuntu 16.04 and Ubuntu 16.10. The Vapor Toolbox works in exactly the same way, with the exception that you can't use Xcode projects on Linux.

You install Vapor on Linux using the Vapor APT repository. Vapor maintains an open source APT repo at https://github.com/vapor/apt. which serves the same purpose as Vapor's Brew tap. It allows for easy installation of packages and manages all the dependencies for you. When you install Vapor, it also installs Swift for you.

This book uses Ubuntu 16.04 throughout when referring to Linux, but the other supported versions of Ubuntu should work in exactly the same way.

To use the Vapor APT repo, add it to your repository list. Enter the following at a shell prompt:

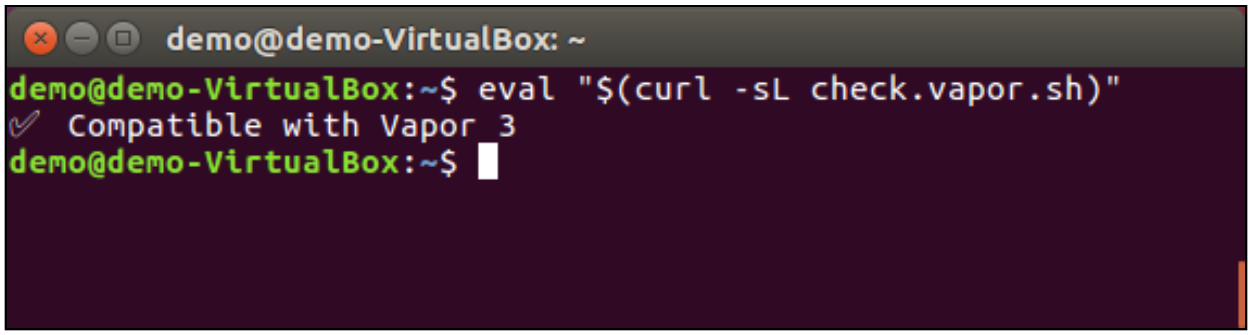
```
eval "$(curl -sL https://apt.vapor.sh)"
```

> **Note**: You may need to install `curl` first before you can run this command if you are starting with a fresh Ubuntu image. Run **sudo apt-get install curl -y** to do so.

Once this has completed, you can then install the Toolbox (and Swift):

```
sudo apt-get install vapor -y
```

When this completes, you should be able to run the check script from above and get a successful result:



Building your first app

Setting up a Vapor project can seem complicated at first as there are a number of required files and directories. To help with this, the Toolbox can create a new project from a template. The default template is a simple API template, but it also has templates for web sites and authentication. You can even create your own templates.

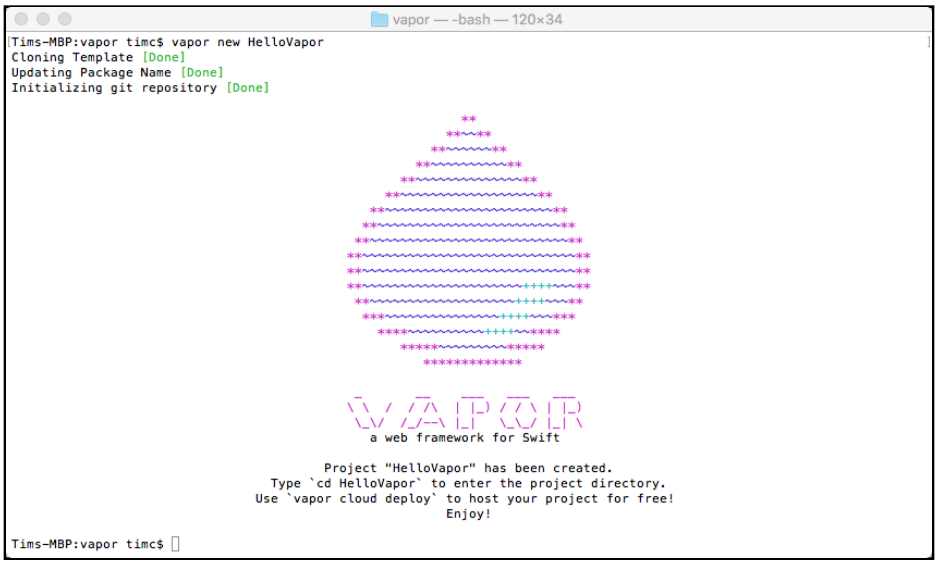
First, create a new directory in your home directory or somewhere sensible to work on your Vapor projects. For example, enter the following commands in Terminal:

```
mkdir ~/vapor
cd ~/vapor
```

This creates a new directory in your home folder called **vapor** and navigates you there. Next, create your project with:

```
vapor new HelloVapor
```

You should see the following:

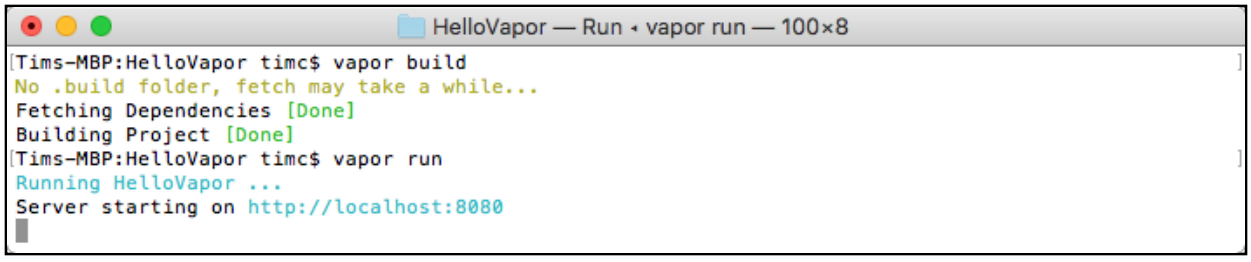


To build and start your app, run:

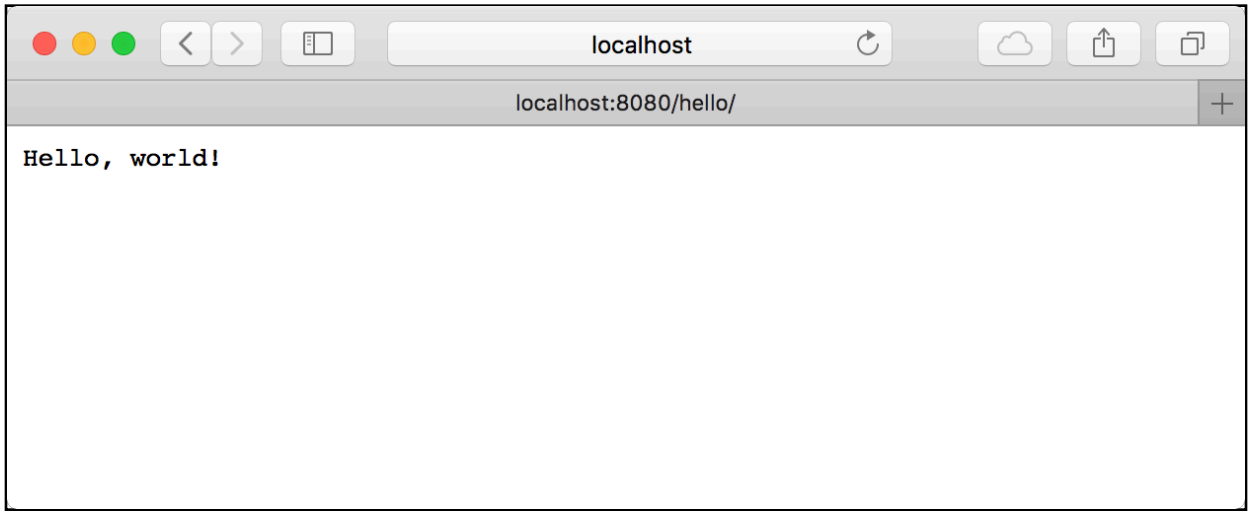
```
# 1
cd HelloVapor
# 2
vapor build
# 3
vapor run
```

Here's what this does:

1. `cd` is the "Change Directory" command and takes you into the project directory.
2. This builds the app. This can take some time the first time since it must fetch all the dependencies.
3. This runs the app. If the macOS Application Firewall pops up asking you to allow network connections, click **Allow**.



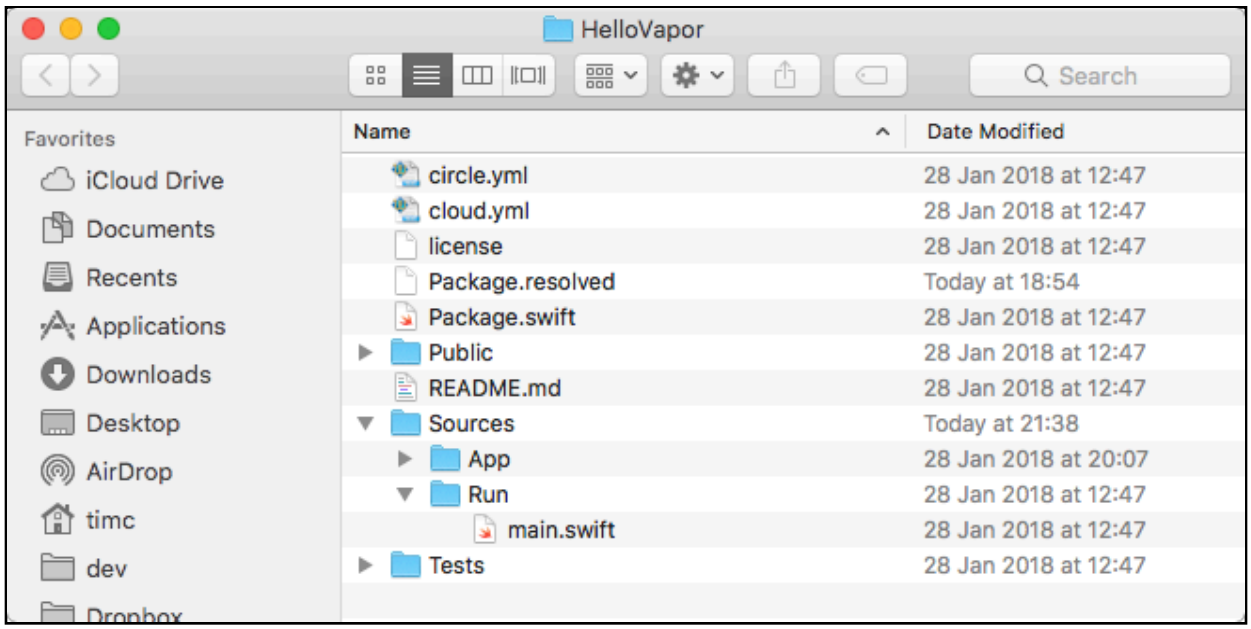
The template has a predefined route, so open your browser and visit **http://localhost:8080/hello** and see the response!



Swift Package Manager

Vapor Toolbox uses Swift Package Manager, or SPM, — a dependency management system similar to Cocoapods on iOS — to configure and build Vapor apps. Open your project directory and look at the structure. On macOS in **Terminal**, enter:

```
open .
```



Notice there's no Xcode project in your template even though you've built and run the app. This is deliberate. In fact, the project file is explicitly excluded from source control using the **.gitignore** file. When using SPM, Xcode projects are discardable and regenerated whenever you make project changes.

An SPM project is defined in the **Package.swift** manifest file. It declares targets, dependencies and how they link together. The project layout is also different from a traditional Xcode project. There is a **Tests** directory for tests. There is a **Sources** directory for source files. Each module defined in your manifest has its own directory inside **Sources**. Your sample app has an `App` module and a `Run` module, so **Sources** contains an **App** directory and a **Run** directory.

Inside the **Run** directory, there's a single **main.swift** file. This is the entry point required by all Swift apps.

> On iOS this is usually synthesized with an `@UIApplicationMain` attribute on the `AppDelegate`.

The template contains everything you need to set up your app and you shouldn't need to change **main.swift** or the `Run` module. Your code lives in `App` or any other modules you define.

Creating your own routes

> This section, as does most of the book, uses Xcode. If you're developing on Linux, use your favorite editor, then use the commands **vapor build** and **vapor run** to build and run your app.

Now that you've made your first app, it's time to see how easy it is to add new routes with Vapor. If the Vapor app is still running, stop it by pressing **Control-C** in Terminal. Next enter:

```
vapor xcode -y
```

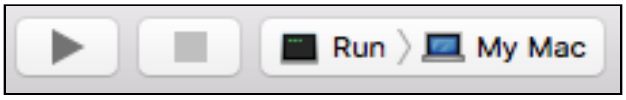
This generates an Xcode project and open it. Open **routes.swift** in **Sources/App**. You'll see the route you visited above. To create another route, add the following after the `router.get("hello")` closure:

```
router.get("hello", "vapor") { req -> String in
  return "Hello Vapor!"
}
```

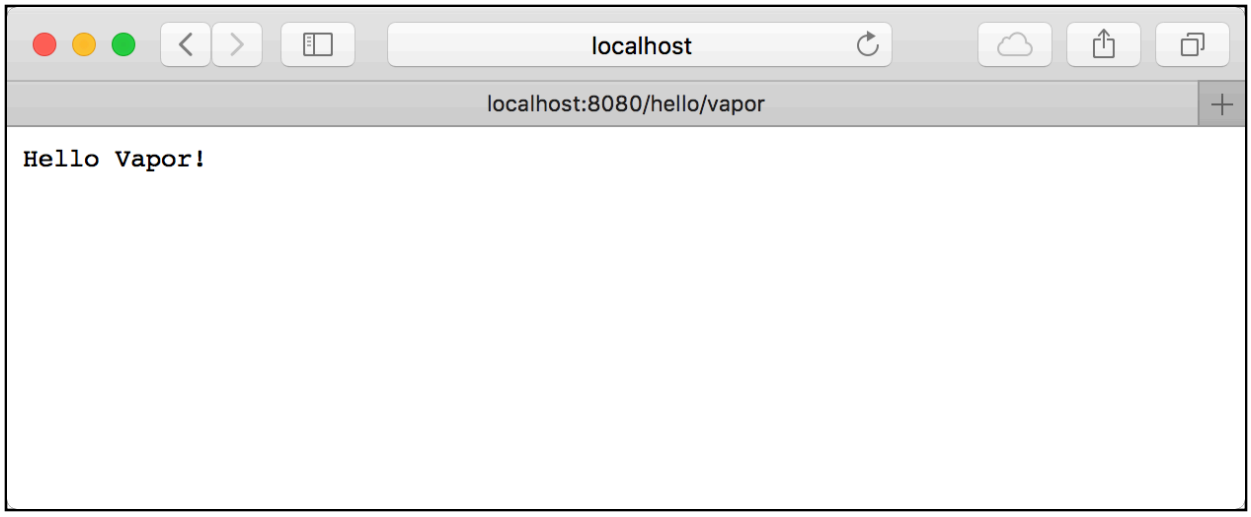
Here's what this does:

- Add a new route to handle a GET request. Each parameter to `router.get` is a path component in the URL. This route is invoked when a user enters **http://localhost:8080/hello/vapor** as the URL.
- Supply a closure to run when this route is invoked. The closure receives a `Request` object; you'll learn more about these later.
- Return a string as the result for this route.

In the Xcode toolbar, select the **Run** scheme and choose **My Mac** as the device.



Build and run. In your browser, visit **http://localhost:8080/hello/vapor**.



What if you want to say hello to anyone who visits your app? Adding every name in the world would be quite impractical! There must be a better way. There is and Vapor makes it easy.

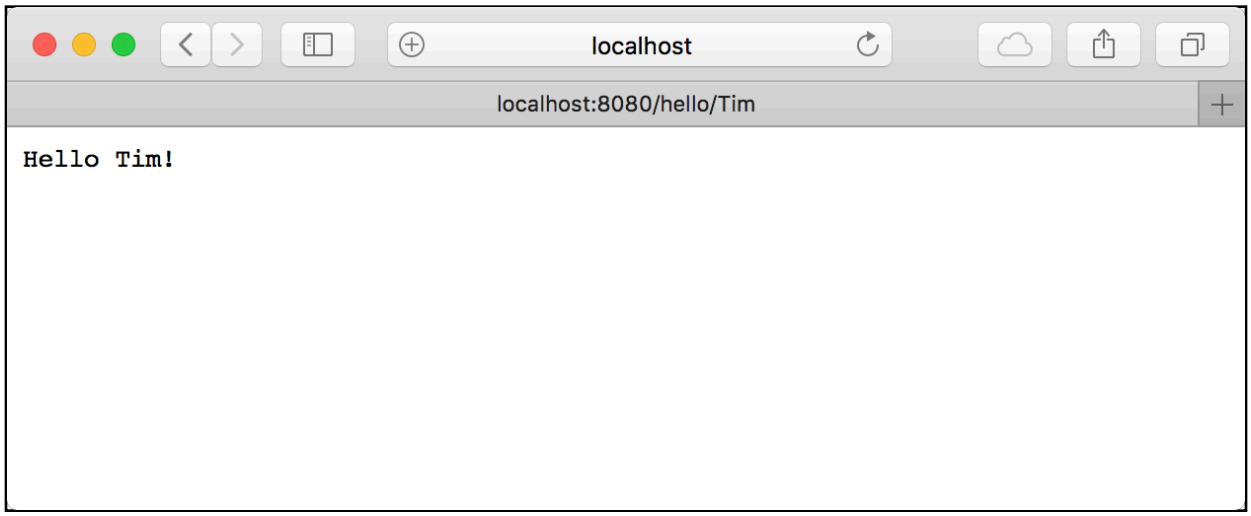
Add a new route that says hello to whomever visits. For example, if your name is Tim, you'll visit the app using the URL **http://localhost:8080/hello/Tim** and it says "Hello, Tim!". Add the following after the code you just entered:

```
// 1
router.get("hello", String.parameter) { req -> String in
  //2
  let name = try req.parameters.next(String.self)
  // 3
  return "Hello, \(name)!"
}
```

Here's the play-by-play:

1. Use `String.parameter` to specify that the second parameter can be any `String`.
2. Extract the user's name which is passed in the `Request` object.
3. Use the name to return your greeting.

Build and run. In your browser, visit **http://localhost:8080/hello/Tim**. Try replacing **Tim** with some other values.



Accepting data

Most web apps must accept data. A common example is user login. To do this, a client sends a POST request with a JSON body which the app must decode and process. To learn more about POST requests and how they work, see Chapter 3, "HTTP Basics."

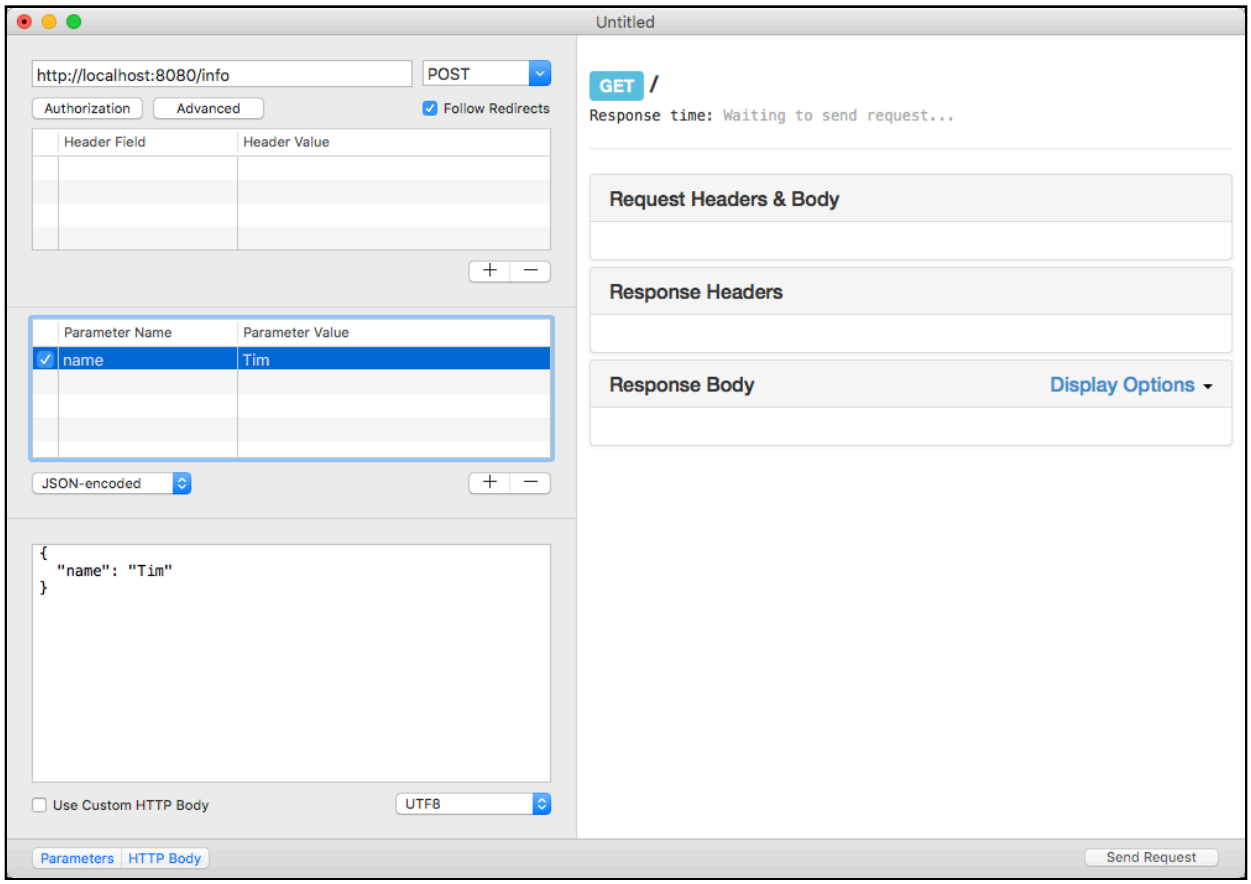
Vapor 3 makes decoding data easy thanks to its strong integration with Swift 4's `Codable`. You give Vapor a `Codable` struct that matches your expected data and Vapor does the rest. Create a POST request to see how this works.

> This book uses the RESTed app, available as a free download from the Mac App Store. If you like, you may use another REST client to test your APIs.

Set up the request as follows:

- **URL**: **http://localhost:8080/info**
- **Method**: **POST**
- Add a single parameter called **name**. Use your name as the value.
- Select **JSON-encoded** as the request type. This ensures that the data is sent as JSON and that the `Content-Type` header is set to `application/json`. If you are using a different client you may need to set this manually.

Your request should look similar to the following:



Go back to Xcode, open **routes.swift** and add the following to the end of the file to create a `struct` called `InfoData` to represent this request:

```
struct InfoData: Content {
  let name: String
}
```

This `struct` conforms to `Content` which is Vapor's wrapper around `Codable`. Vapor uses `Content` to extract the request data, whether it's the default JSON-encoded or form URL-encoded. `InfoData` contains the single parameter `name`.

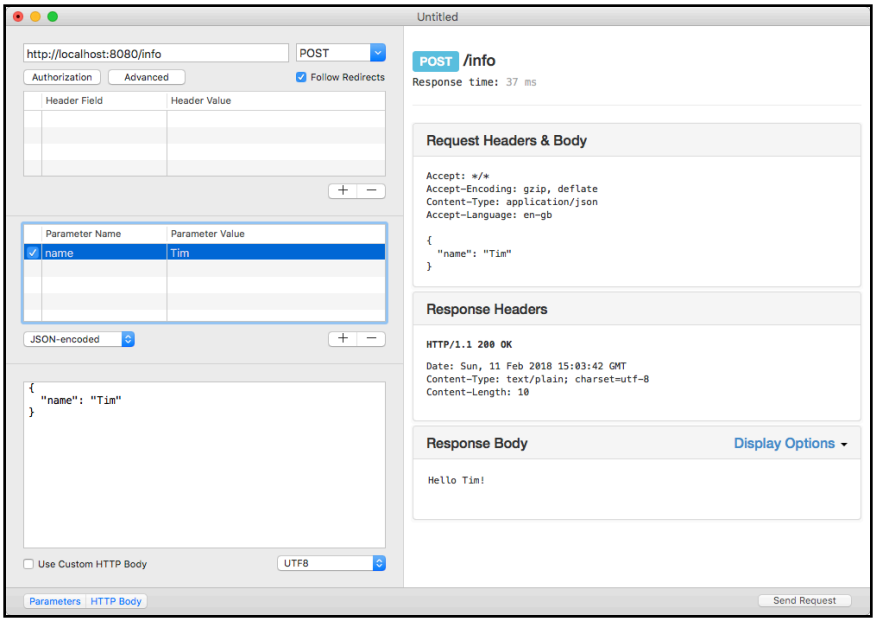
Next add a new route after the `router.get("hello", "vapor")` closure:

```
router.post(InfoData.self, at: "info") { req, data -> String in
  return "Hello \(data.name)!"
}
```

Here's what this does:

- Add a new route handler to handle a POST request for the URL **http://localhost:8080/info**. This route handler returns a `String`. The route handler accepts a `Content` type as the first parameter and any path parameters after the `at:` parameter name. The route handler decodes the data and passes it to the closure as the second parameter.
- Return the string by pulling the name out of the `data` variable.

Build and run the app. Send the request from RESTed and you'll see the response come back:



This may seem like a lot of boilerplate to extract a single parameter from JSON. However, `Codable` scales up and allows you to decode complex, nested JSON objects with multiple types in a single line.

Returning JSON

Vapor also makes it easy to return JSON in your route handlers. This is a common need when your app provides an API service. For example, a Vapor app that processes requests from an iOS app needs to send JSON responses. Vapor again uses `Content` to encode the response as JSON.

Open **routes.swift** and add the following struct to the end of the file called `InfoResponse` to return the incoming request:

```
struct InfoResponse: Content {
  let request: InfoData
}
```

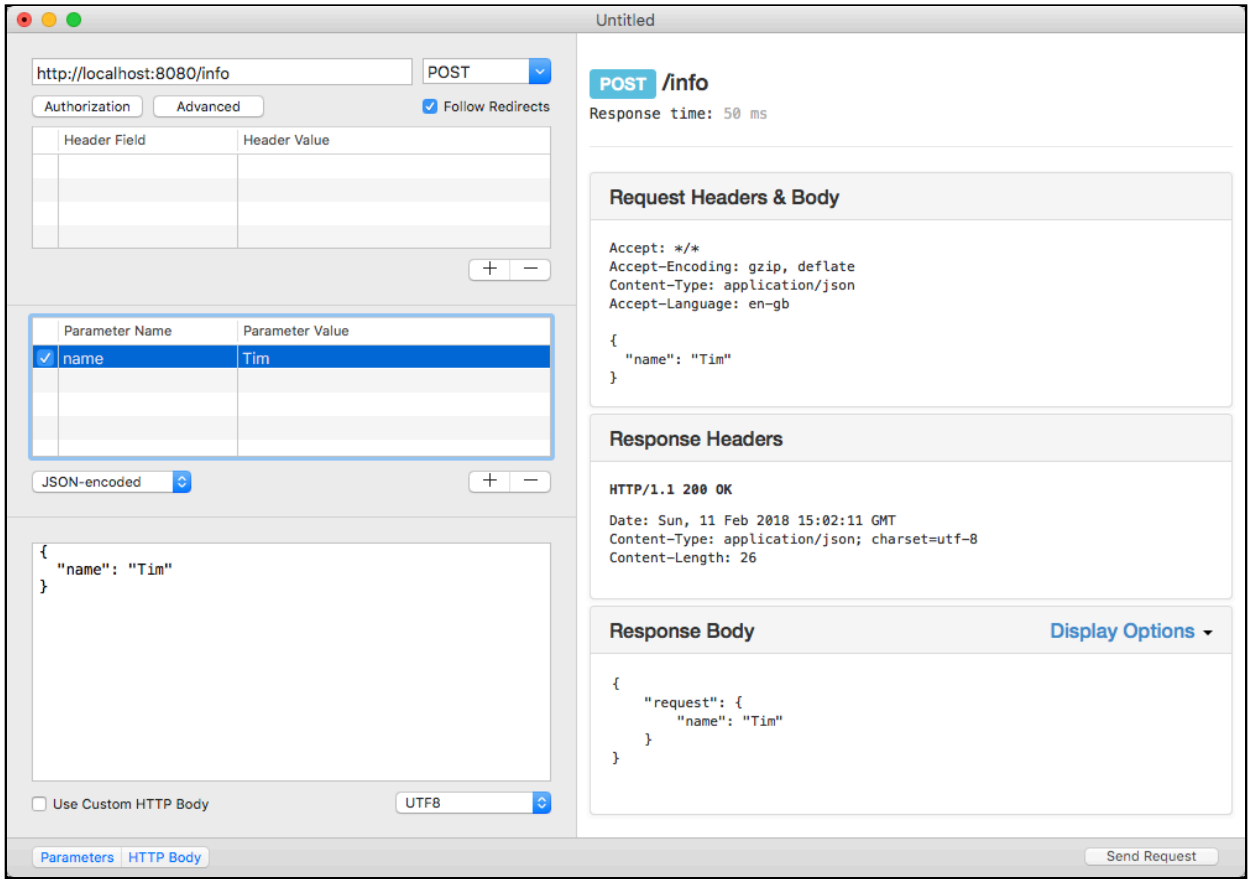
This `struct` conforms to `Content` and contains a property for the request. Next, replace `router.post(InfoData.self, at: "info")` with the following:

```
// 1
router.post(InfoData.self, at: "info") {
  req, data -> InfoResponse in
  // 2
  return InfoResponse(request: data)
}
```

Here's what changed:

1. The route handler now returns the new `InfoResponse` type.
2. Construct a new `InfoResponse` type using the decoded request.

Build and run the app. Send the same request from RESTed. You'll see a JSON response containing your original request data:



Troubleshooting Vapor

Throughout the course of this book, and in any future Vapor apps, you may encounter errors in your projects. There are a number of steps to take to troubleshoot any issues.

Regenerate your project

First and foremost, you should regenerate your Xcode project. In Terminal, type:

```
vapor xcode -y
```

This regenerates your project like the first time you created the Xcode project. This can fix issues such as missing files or project configuration errors.

Update your dependencies

Another scenario you may encounter is hitting a bug in Vapor or another dependency you use. Make sure you are on the latest package version of any dependencies to see if the update fixes the issue. In Terminal, type:

```
vapor update
```

This uses the underlying `swift package update` command to pull down any updates to your dependencies and use the latest releases you support in **Package.swift**. Note that while packages are in the beta or release candidate stages, there may be breaking changes between updates.

Clean and rebuild

Finally, if you are still having issues, you can use the software equivalent of "turn it off and on again". In Xcode use **Command-Option-Shift-K** to clean the build folder. You may also need to remove the build products created by the Vapor Toolbox. To do this, enter the following in Terminal:

```
rm -rf .build
```

This removes all build artifacts and your dependencies. You may also need to clear your derived data for the Xcode project as well. The "nuclear" option involves:

- Remove the **.build** directory to remove any build artifacts.
- Remove your **.xcodeproj** to delete the Xcode project and any misconfigurations.
- Remove **Package.resolved** to ensure you get the latest dependencies next time you build.
- Remove derived data to clear extra Xcode build artifacts.

Vapor Discord

The above steps usually fix most issues encountered that aren't caused by your code. If all else fails, head to Vapor's Discord server. There are thousands of developers discussing Vapor, it's changes, and helping people with issues. Click the **Join Chat** button on Vapor's web site: https://vapor.codes.

Where to Go From Here?

We hope you enjoyed this sample of *Server Side Swift with Vapor*!

If you did, be sure to check out the full book, which contains the following chapters:

1. **Introduction**: Get a history of Vapor and an overview of what's in store for you in the rest of the book.
2. **Hello Vapor!**: Beginning a project using a new technology can be daunting. Vapor makes it easy to get started. It even provides handy scripts to make sure that your computer is configured correctly. In this chapter, you'll start by installing the Vapor Toolbox, then use it to build and run your first project. You'll finish by learning about routing, accepting data and returning JSON.
3. **HTTP Basics**: Before you begin your journey with Vapor, you'll first review the fundamentals of how the web and HTTP operate, including its methods and most common response codes. You'll also learn how Vapor differs from other Swift frameworks, its benefits, and how it can augment your web development experience.
4. **Async**: In this chapter, you'll learn about asynchronous and non-blocking architectures. You'll cover Vapor's approach to these architectures and how to use them. Finally, the chapter will provide a foundational overview of SwiftNIO, a core technology used by Vapor.
5. **Fluent & Persisting Models**: In Chapter 2, "Hello, Vapor!", you learned the basics of creating a Vapor app, including how to create routes. Chapter 5 explains how to use Fluent to save data in Vapor apps. You'll also learn how to deploy your app using Vapor Cloud.

6. **Configuring a Database**: Databases allow you to persist data in your apps. In this chapter, you'll learn how to configure your Vapor app to integrate with the database of your choice. Finally, you'll deploy your app to Vapor Cloud and learn how to set up the database there.

7. **CRUD Database Operations**: Chapter 5, "Fluent and Persisting Models", explained the concept of models and how to store them in a database using Fluent. Chapter 7 concentrates on how to interact with models in the database. You'll learn about CRUD operations and how they relate to REST APIs. You'll also see how to leverage Fluent to perform complex queries on your models. Finally, like all chapters in this section, you'll deploy your code to Vapor Cloud.

8. **Controllers**: In previous chapters, you wrote all the route handlers in one file. This isn't sustainable for large projects as the file quickly becomes too big and cluttered. This chapter introduces the concept of controllers to help manage your routes and models, using both basic controllers and RESTful controllers. Finally, you'll deploy your code to Vapor Cloud.

9. **Parent Child Relationships**: In Chapter 5, "Fluent and Persisting Models", introduced the concept of models. This chapter will show you how to set up a parent child relationship between two models. You'll learn the purpose of these relationships, how to model them in Vapor and how to use them with routes. You'll complete the tutorial by deploying your code to Vapor Cloud.

10. **Sibling Relationships**: In Chapter 9, "Parent Child Relationships", you learned how to use Fluent to build parent child relationships between models. Chapter 10 will show you how to implement the other type of relationship: sibling relationships. You'll learn how to model them in Vapor and how to use them in routes. Finally, you'll deploy your code to Vapor Cloud.

11. **Testing**: In this chapter, you'll learn how to write tests for your Vapor applications. You'll learn why testing is important and how it works with Swift Package Manager. Then, you'll learn how to write tests for the TIL application from the previous chapters. Finally, you'll see why testing matters on Linux and how to test your code on Linux using Docker.

12. **Creating a Simple iPhone App I**: In the previous chapters, you created an API and interacted with it using RESTed. However, users expect something a bit nicer to use TIL! The next two chapters show you how to build a simple iOS app that interacts with the API. In this chapter, you'll learn how to create different models and get models from the database.

13. **Creating a Simple iPhone App II**: In this chapter, you'll expand the app to include viewing details about a single acronym. You'll also learn how to perform the final CRUD operations: edit and delete. Finally, you'll learn how to add acronyms to categories.

14. **Templating with Leaf**: In a previous section of the book, you learned how to create an API using Vapor and Fluent. This section explains how to use Leaf to create dynamic websites in Vapor applications. Just like the previous section, you'll deploy the website to Vapor Cloud.

15. **Beautifying Pages**: In this chapter, you'll learn how to use the Bootstrap framework to add styling to your pages. You'll also learn how to embed templates so you only have to make changes in one place. Next, you'll also see how to serve files with Vapor. Finally, like every chapter in this section, you'll deploy the new website to Vapor Cloud.

16. **Making a Simple Web App I**: In the previous chapters, you learned how to display data in a website and how to make the pages look nice with Bootstrap. In this chapter, you'll learn how to create different models and how to edit acronyms.

17. **Making a Simple Web App II**: In this chapter, you'll learn how to allow users to add categories to acronyms in a user-friendly way. Finally, you'll deploy your completed web application to Vapor Cloud.

18. **API Authentication, Part I**: In this chapter, you'll learn how to protect your API with authentication. You'll learn how to implement both HTTP basic authentication and token authentication in your API.

19. **API Authentication, Part II**: Once you've implemented API authentication, neither your tests nor the iOS application work any longer. In this chapter, you'll learn the techniques needed to account for the new authentication requirements, and you'll also deploy the new code to Vapor Cloud.

20. **Web Authentication, Cookies and Sessions**: In this chapter, you'll see how to implement authentication for the TIL website. You'll see how authentication works on the web and how Vapor's Authentication module provides all the necessary support. You'll then see how to protect different routes on the website. Next, you'll learn how to use cookies and sessions to your advantage. Finally, you'll deploy your code to Vapor Cloud.

21. **Validation**: In this chapter, you'll learn how to use Vapor's Validation library to verify some of the information users send the application. You'll create a registration page on the website for users to sign up. You'll validate the data from this form and display an error message if the data isn't correct. Finally, you'll deploy the code to Vapor Cloud.

22. **Database/API Versioning & Migration**: Up until this point in the book, whenever you made a change to your model, you had to delete your database and start over. That's no problem when you don't have any data. Once you have data, or move your project to the production stage, you can no longer delete your database. What you want to do instead is modify your database, which in Vapor, is done using migrations - and that's what you'll learn about in this chapter.

23. **Caching**: Whether you're creating a JSON API, building an iOS app, or even designing the circuitry of a CPU, you'll eventually need a cache. In this chapter, you'll learn the philosophy behind and uses of caching to make your app feel snappier and more responsive.

24. **Middleware**: In the course of building your application, you'll often find it necessary to integrate your own steps into the request pipeline. The most common mechanism for accomplishing this is to use one or more pieces of middleware. They allow you to do things like log incoming requests, catch errors and display messages, and more. In this chapter, you'll learn how to create your own middleware in Vapor.

25. **Deploying with Heroku**: Heroku is a popular hosting solution that simplifies deployment of web and cloud applications. It supports a number of popular languages and database options. In this chapter, you'll learn how to deploy a Vapor web app with a Postgres database on Heroku.

26. **Websockets**: WebSockets, like HTTP, define a protocol used for communication between two devices. Unlike HTTP, the WebSocket protocol is designed for realtime communication. Vapor provides a succinct API to create a WebSocket server or client. In this chapter, you'll build a simple server/client application that allows users to share their current location with others, who can then view this on a map in realtime.

You can find the book on the raywenderlich.com store here:

- https://store.raywenderlich.com/products/server-side-swift-with-vapor

We hope you enjoy the book! :]

— Tim, Tanner, Jonas, Logan, and the *Server Side Swift with Vapor* team

Learn how to use Swift on the server!

Server Side Swift with Vapor introduces you to the world of server development with the added bonus of using Swift. You'll learn how to build APIs, web sites, databases, application servers and use Vapor's very own Vapor Cloud to host your solutions off-site. You'll use many of Vapor's modules such as Fluent, Vapor's ORM, and Leaf, the templating engine for building web pages.

Who This Book Is For

This book is for iOS developers who already know the basics of iOS and Swift development and want to transfer that knowledge to writing server based applications.

Topics Covered in Server Side Swift with Vapor:

- **HTTP:** Learn the basics of how to make requests to and from servers.
- **Fluent:** Learn how to use Fluent to save and manage your models in databases.
- **Controllers:** Learn how to use controllers to route your requests and responses.
- **Leaf:** Learn how Vapor's Leaf module and its templating language allow you to build dynamic web sites directly.
- **Middleware:** Learn how built-in Vapor modules can assist with common tasks such as validating users, settings required response headers, serving static files and more.

One thing you can count on: After reading this book, you'll be prepared to write your own server-side applications using Vapor and, of course, Swift.

About the Vapor Team

The Vapor book team is comprised of some of the most prolific Vapor developers, including Core Developers Tanner Nelson, Logan Wright and Jonas Schwartz. Another stalwart of the book is active Vapor community member Tim Condon, who really got his teeth into this project and, along with the Core Developers, made the book possible.

If you enjoyed the book, there's also a video series produced by our own Tim Condon. You'll find these videos and more resources on our website at www.raywenderlich.com.