



THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 9

BACKENDS FOR MOBILE APPLICATIONS

CLASS NEWS

CLASS NEWS

- Office hours Thursday from 10-11:30 in Young 308

CLASS NEWS

- Week 7 - CloudKit
 - Assignment 5 assigned
- Week 8 - CloudKit Server Extensions; Swift on the Server
 - Case Study assigned
 - Finish server component of Assignment 5
- Week 9 - Case Studies in Class (Assignment 5 Due)
- Week 10 - Cloud Roundup (Realm, Serverless, etc.)

CLASS NEWS

- Case Studies
 - Find an area of interest in mobile/cloud computing and present to class
 - Ideas:
 - New service (serverless, azure, aws lambda)
 - Technique (sharding :), mysql with app engine)
 - Case study (Snapchat on App Engine, what does XX company use)
 - Deeper dive on topic covered (eg. Google Vision API, etc.)
 - 30 minutes

CLASS NEWS

- Week 7 - CloudKit
 - Assignment 5 assigned
- Week 8 - CloudKit Server Extensions; Swift on the Server
 - Case Study assigned
 - Finish server component of Assignment 5
- Week 9 - Case Studies in Class (Assignment 5 Due)
- Week 10 - Cloud Roundup (Realm, Serverless, etc.)

MAKE-UP CLASS
MATERIALS AFTER
PRESENTATIONS

CLASS NEWS

- Final Projects
 - Opportunity for you to apply what you learned in class to a project you care about
 - Can use old iOS project or extend assignment
 - Talk me about your ideas early

BREAK TIME



OPEN SOURCE SWIFT

OPEN SOURCE SWIFT

Dec 3, 2015

Swift is Open Source

Swift is now open source. Today Apple launched the open source Swift community, as well as amazing new tools and resources including:

- [Swift.org](#) – a site dedicated to the open source Swift community
- Public source code repositories at [github.com/apple](#)
- A new Swift package manager project for easily sharing and building code
- A Swift-native core libraries project with higher-level functionality above the standard library
- Platform support for all Apple platforms as well as Linux

Now anyone can download the code and in-development builds to see what the team is up to. More advanced developers interested in contributing to the project can file bugs, participate in the community, and contribute their own fixes and enhancements to make Swift even better. For production App Store development you should always use the stable releases of Swift included in Xcode, and this remains a requirement for app submission.

- It all started here

Swift.org

Swift.org is an entirely new site dedicated to open source Swift. This site hosts resources for the community of developers that want to help evolve Swift, contribute fixes, and most importantly, interact with each other. Swift.org hosts:

- A bug reporting and tracking system
- Mailing lists
- A blog dedicated to the engineering of Swift
- Community guidelines
- Getting started tutorials
- Contributing instructions
- Documentation on Swift

© 2015 Apple Inc. All rights reserved.

OPEN SOURCE SWIFT

- The promise
 - Same code runs in both places
 - Reduce development time by sharing code
 - Leverage (some) frameworks and APIs



OPEN SOURCE SWIFT



- Something is missing...

OPEN SOURCE SWIFT

- Linux and Mac Platform support
- Standard Library Foundation, Dispatch, and XCTest Compiler
- Command Line Tools



Swift

ABOUT SWIFT

BLOG

DOWNLOAD

GETTING STARTED

DOCUMENTATION

SOURCE CODE

COMMUNITY

CONTRIBUTING

CONTINUOUS
INTEGRATION

PROJECTS

COMPILER AND
STANDARD LIBRARY

PACKAGE MANAGER

CORE LIBRARIES

REPL AND DEBUGGER

Welcome to Swi

Swift is now open source!

We are excited by this new chapter in the story of Swift. When Apple unveiled the Swift programming language, it quickly became one of the fastest growing languages in history. Swift is a modern, safe, and efficient software that is incredibly fast and safe by design. Now that Swift is open source, you can help make the best general-purpose programming language available everywhere.

For students, learning Swift has been a great introduction to modern programming concepts and best practices. Now that Swift is open source, their Swift skills will be able to be applied to a broader range of platforms, from mobile devices to the web to the cloud.

Welcome to the Swift community. Together we can build a better programming language for everyone.

– **The Swift Team**

STATE OF SWIFT ON THE SERVER

STATE OF SWIFT ON THE SERVER

- Many Swift web frameworks in development
- Most popular (15,000+ stars)



STATE OF SWIFT ON THE SERVER

- Perfect
 - A complete framework
 - 11,448 ★
 - Rails inspired

The screenshot shows the homepage of perfect.org. At the top, there is a navigation bar with links for "WHAT IS PERFECT?", "DOCUMENTATION", "DEVELOPER RESOURCES", and "EVENTS". Social media icons for LinkedIn, Twitter, Facebook, YouTube, and GitHub are also present, along with a search icon. Below the navigation, there are four main sections: "Get Started" (with an orange bird icon), "Documentation" (with an orange open book icon), "Assistant" (with an orange laptop icon), and "Commercial" (with an orange hexagonal icon). Each section has a brief description and a call-to-action button at the bottom. A large orange banner at the bottom features a play button icon and the text "TAKE ME TO THE LEARNING CENTRE". Another banner to the right encourages users to "Show me Ray Wenderlich's video tutorials >>". The central part of the page contains the heading "What is Perfect?" and a brief description of the service.

perfect.org

WHAT IS PERFECT? DOCUMENTATION DEVELOPER RESOURCES EVENTS

Get Started Documentation Assistant Commercial

Access Perfect's feature set and start your project.

Get support from our comprehensive set of documentation.

Meet the Perfect Assistant to help you do more with Perfect and Swift, more easily.

Use Perfect to build apps for business.

Get Started >> Documentation >> Meet the Assistant >> Find Out How >>

TAKE ME TO THE LEARNING CENTRE

Show me Ray Wenderlich's video tutorials >>

What is Perfect?

Perfect is a web server and toolkit for developers using the Swift programming language to build applications and other REST services. It lets developers build using only Swift to program both the client-facing and server-side of their

STATE OF SWIFT ON THE SERVER

- Perfect
 - A complete framework
 - Runs its own server or as a fastCGI module for Apache or NGINX

```
import PerfectLib

public func PerfectServerModuleInit() {
    Routing.Handler.registerGlobally()

    Routing.Routes["/] = { _ in return HelloWorldHandler() }

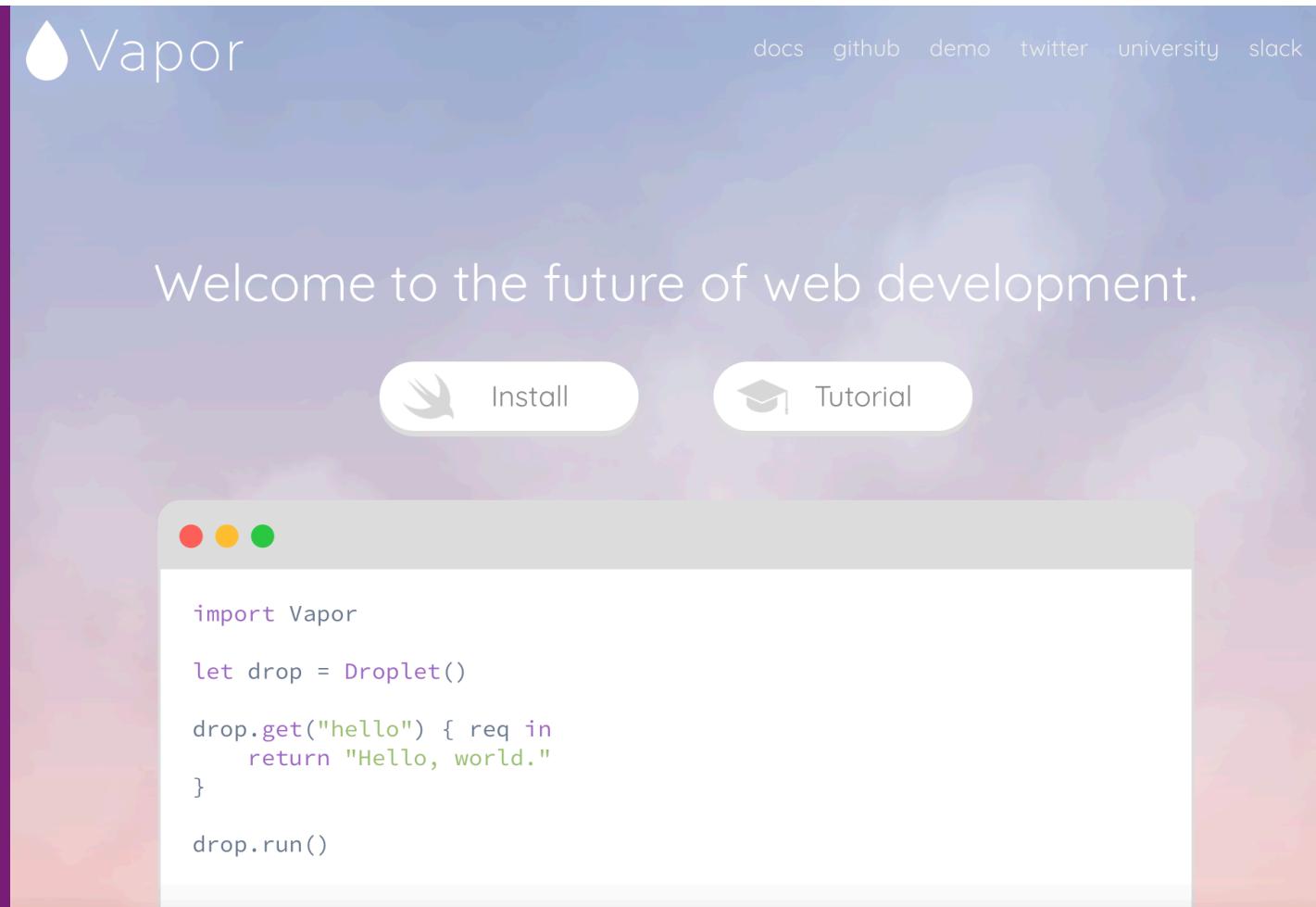
}

class HelloWorldHandler: RequestHandler {
    func handleRequest(request: WebRequest, response: WebResponse) {
        response.appendBodyString("Hello, World!")
        response.requestCompletedCallback()
    }
}
```

STATE OF SWIFT ON THE SERVER

- Vapor

- 9,479 ★
- Laravel (php)
inspired



The image shows the Vapor website homepage. At the top left is the Vapor logo (a white water droplet icon) followed by the word "Vapor". To the right are links for "docs", "github", "demo", "twitter", "university", and "slack". Below the header is a large, semi-transparent text box containing the slogan "Welcome to the future of web development." In the center of this box are two buttons: "Install" with a bird icon and "Tutorial" with a graduation cap icon. At the bottom of the page is a code editor window displaying Swift code:

```
import Vapor

let drop = Droplet()

drop.get("hello") { req in
    return "Hello, world."
}

drop.run()
```



STATE OF SWIFT ON THE SERVER

- Perfect
 - A complete framework
 - Runs its own server or as a fastCGI module for Apache or NGINX

```
import Vapor

let app = Application()

app.get("/") { request in
    return "Hello, World!"
}

app.start()
```

STATE OF SWIFT ON THE SERVER

- Kitura
 - Swift@IBM
 - 5,686 ★
 - Express.js inspired

Getting started Tutorials API reference Support

A high performance and simple to use web framework for building modern Swift applications.

Get started with Kitura

STATE OF SWIFT ON THE SERVER

- Kitura
 - Swift@IBM
 - 5,686 ★

```
import Kitura

let router = Router()

router.get("/") { request, response, next in
    response.send("Hello, World!")
    next()
}

Kitura.addHTTPServer(onPort: 8090, with: router)
Kitura.run()
```

STATE OF SWIFT ON THE SERVER

- Server Side Swift Standards (S4) project
- Protocols and standards

open-swift / S4

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs

HTTP standards for Swift

264 commits 2 branches 27 releases 12 contributors

Branch: master New pull request Create new file Upload files Find file Clone

noppoMan committed with paulofaria Updating to September 10, 2016 (#78) Latest commit a544ad7 on

File	Last Commit	Author(s)
Sources	Updating to September 10, 2016 (#78)	8
Tests	Updating to August 4, 2016	9
.gitignore	Update to C7 0.4.0	
.swift-version	Updating to September 10, 2016 (#78)	8
.travis.yml	Updated Swift to 07-25 (#74)	10
LICENSE	initial setup	
Package.swift	Updating to August 4, 2016	9
README.md	Updated Swift to 07-25 (#74)	10
S4.podspec	add cocoapods support	

README.md

S4 - HTTP

STATE OF SWIFT ON THE SERVER

- Server Side Swift Standards (S4) project
 - Protocols and standards
- Apple has a working group dedicated to swift on the server

VAPOR

VAPOR

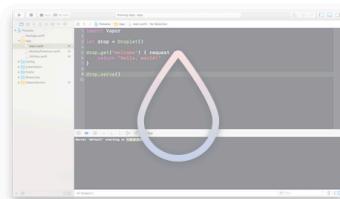
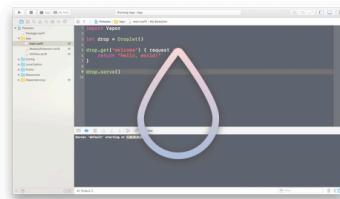
- Web framework and server for Swift
- Works on macOS and Ubuntu

The screenshot shows the GitHub repository page for 'vapor / vapor'. The repository has 3,196 commits, 6 branches, 172 releases, and 77 contributors. The latest commit was made 6 hours ago. The repository uses the MIT license.

File	Description	Time Ago
.Documents	Adding another break line	3 months ago
.Sources	Sessions Middleware doesn't require Foundation anymore	10 hours ago
.Tests	Merge branch 'master' into sessionsMiddlewareImprovements	8 hours ago
.Utilities	allow xcode 8.1	8 months ago
.codecov.yml	remove unnecessary files	a day ago
.gitignore	Make sure sessions cookie is HTTP only	14 hours ago
.travis.yml	prevent double provider boot	a month ago
ISSUE_TEMPLATE.md	Update ISSUE_TEMPLATE.md	a month ago
LICENSE	add MIT License	a year ago
Package.swift	remove perf target	15 days ago
README.md	text updates	13 hours ago
circle.yml	prevent double provider boot	a month ago

VAPOR

- Great documentation and resources



vapor.university



Look Maa! Server Side Swift Using Vapor

A detailed introduction to using Vapor and a SQLite database.

15m • Easy • Mohammad Azam

Social Authentication

Learn how to implement user authentication using Facebook and Google.

8m • Intermediate • Caleb Kleveter

User Authentication

Learn how to authenticate users using Vapor's Auth module and PostgreSQL.

10m • Intermediate • Caleb Kleveter

GETTING STARTED

[Install Swift 3: macOS](#)[Install Swift 3: Ubuntu](#)[Install Toolbox](#)[Hello, World](#)[Manual](#)[Xcode](#)

GUIDE

[Droplet](#)[Folder Structure](#)[JSON](#)[Config](#)[Views](#)[Leaf](#)

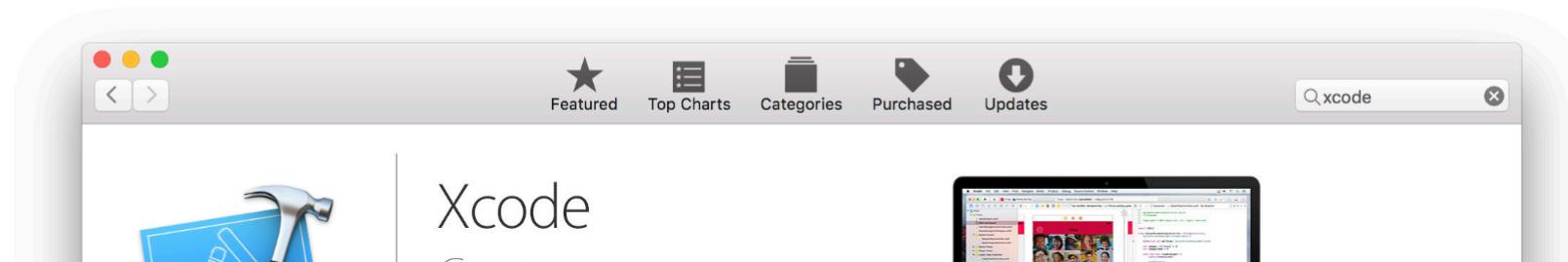
Install Swift 3: macOS

[!\[\]\(db15dda19f000f4725442302275d21aa_img.jpg\) Edit on GitHub](#)

To use Swift 3 on macOS, you just need to have Xcode 8 installed.

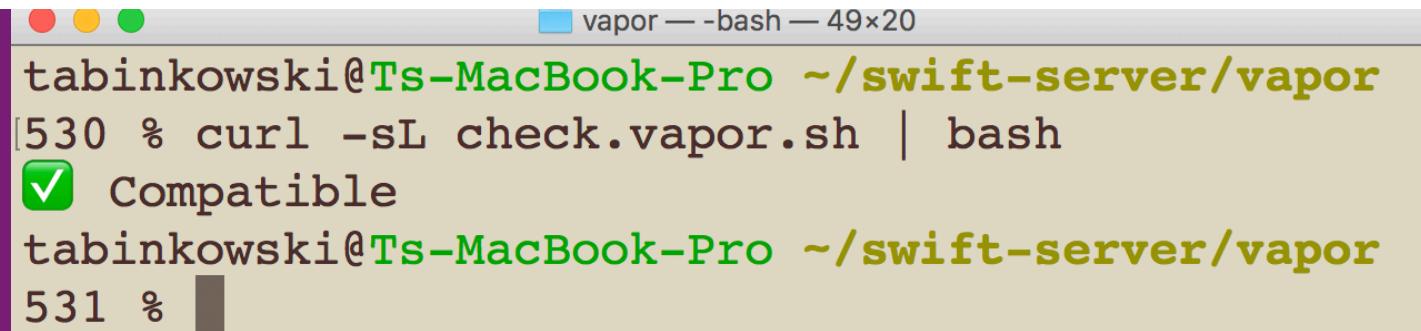
Install Xcode

Install [Xcode 8](#) from the Mac App Store.



VAPOR

- Install Xcode
- Check version for vapor



A screenshot of a macOS terminal window titled "vapor — -bash — 49x20". The window shows the following command and its output:

```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
[530 % curl -sL check.vapor.sh | bash
✓ Compatible
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
531 % ]
```

The terminal uses color coding: green for host and path, grey for command, brown for output, and black for the prompt.

VAPOR

- Toolbox provides command line and shortcuts for common tasks
- Install with homebrew 🍺

```
vapor — git-remote-https • brew install vapor/tap/vapor — 49×20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
534 % brew install vapor/tap/vapor
```

VAPOR

- Toolbox provides command line and shortcuts for common tasks
- Install with homebrew 

```
vapor -- bash -- 50x20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
542 % vapor --help
[Usage: vapor <new|build|update|run|fetch|clean|test|xcode|version|self|heroku>
Join our Slack if you have questions, need help,
or want to contribute: http://vapor.team
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
543 % ]
```

VAPOR

```
vapor — -bash — 80x20
[ tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
542 % vapor --help
Usage: vapor <new|build|update|run|fetch|clean|test|xcode|version|self|heroku>
Join our Slack if you have questions, need help,
or want to contribute: http://vapor.team
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
543 % ]
```

VAPOR

```
# The toolbox can update itself. This may be useful if # you
experience any issues in the future.
vapor self update

# Templates
# The toolbox can create a project from the Vapor basic-
template or any other git repo.
vapor new <name> [--template=<repo-url-or-github-path>]
```

VAPOR

- `vapor new vapor-hello-world`
 - Build a new project from basic templates
 - Can specify other templates

```
vapor — bash — 50x20
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ * *
* * ~~~~~ +++++ * *
* * ~~~~~ ++++ * *
*** ~~~~~ +++ * ***
**** ~~~~~ ++++ * ***
***** ~~~~~ +--- * ***
***** ~*****
```

\ \ / / \ \ / \ [P) / \ \ / [R)
a web framework for Swift

VAPOR

- `vapor new vapor-hello-world`
- Managing vapor projects
 - Xcode
 - Swift Package Manager for other builds

```
vapor-hello-world — swift-package ▾ vapor xcode — 50×20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
[557 % cd vapor-hello-world/
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[558 % vapor xcode
No .build folder, fetch may take a while...
Fetching Dependencies [• ]
```

VAPOR

BUILDS A FRAMEWORK AND APP

The screenshot shows a Mac OS X desktop with an Xcode project window open. The title bar says "vapor-hello-world: Ready | Today at 9:47 PM". The left sidebar shows the project structure for "vapor-hello-world" with files like Package.swift, main.swift, and various models and controllers. The main editor area displays the following Swift code:

```
import PackageDescription

let package = Package(
    name: "vapor-hello-world",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 1, minor: 5)
    ],
    exclude: [
        "Config",
        "Database",
        "Localization",
        "Public",
        "Resources",
    ]
)
```

A yellow speech bubble points from the word "BUILDS" in the header to the ".Package" line in the code. Another yellow speech bubble points from the word "USES" in the footer to the "Dependencies" section of the code.

VAPOR

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure for "vapor-hello-world". It includes a "Sources" folder containing "App" (with "main.swift", "Controllers", and "Models"), "Tests", "Config", "Localization", "Public", and "Resources". A yellow callout bubble labeled "DEPENDENCIES" points to the "Dependencies" folder.
- File Navigator:** Shows the "Package.swift" file selected.
- Editor:** Displays the contents of "Package.swift":

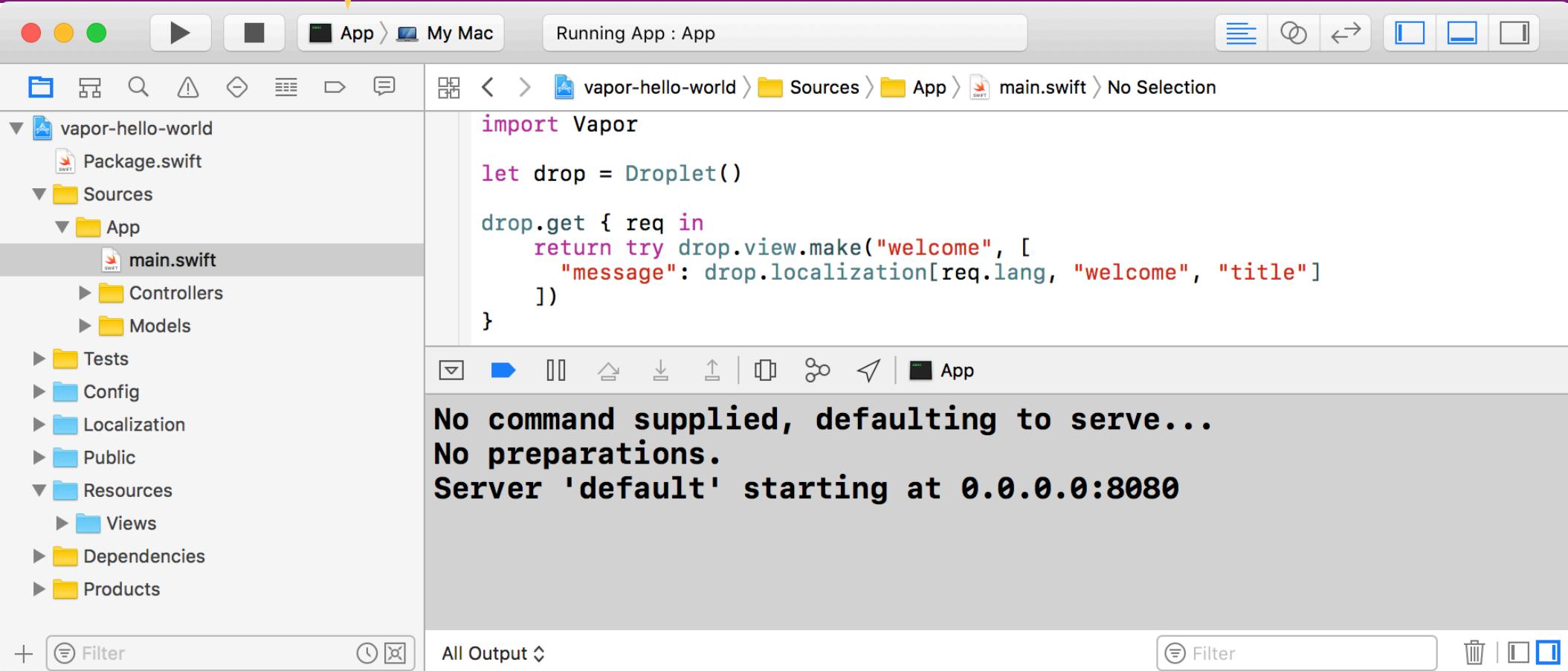
```
import PackageDescription

let package = Package(
    name: "vapor-hello-world",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 1, minor: 5)
    ],
    exclude: [
        "Config",
        "Database",
        "Localization",
        "Public",
        "Resources",
    ]
)
```

The status bar at the top indicates "Building vapor-hello-world: Cipher | Copying swiftdocs".

VAPOR

RUN APP



A screenshot of a Mac OS X desktop environment showing a terminal window. The window title is "Running App : App". The terminal output is as follows:

```
import Vapor

let drop = Droplet()

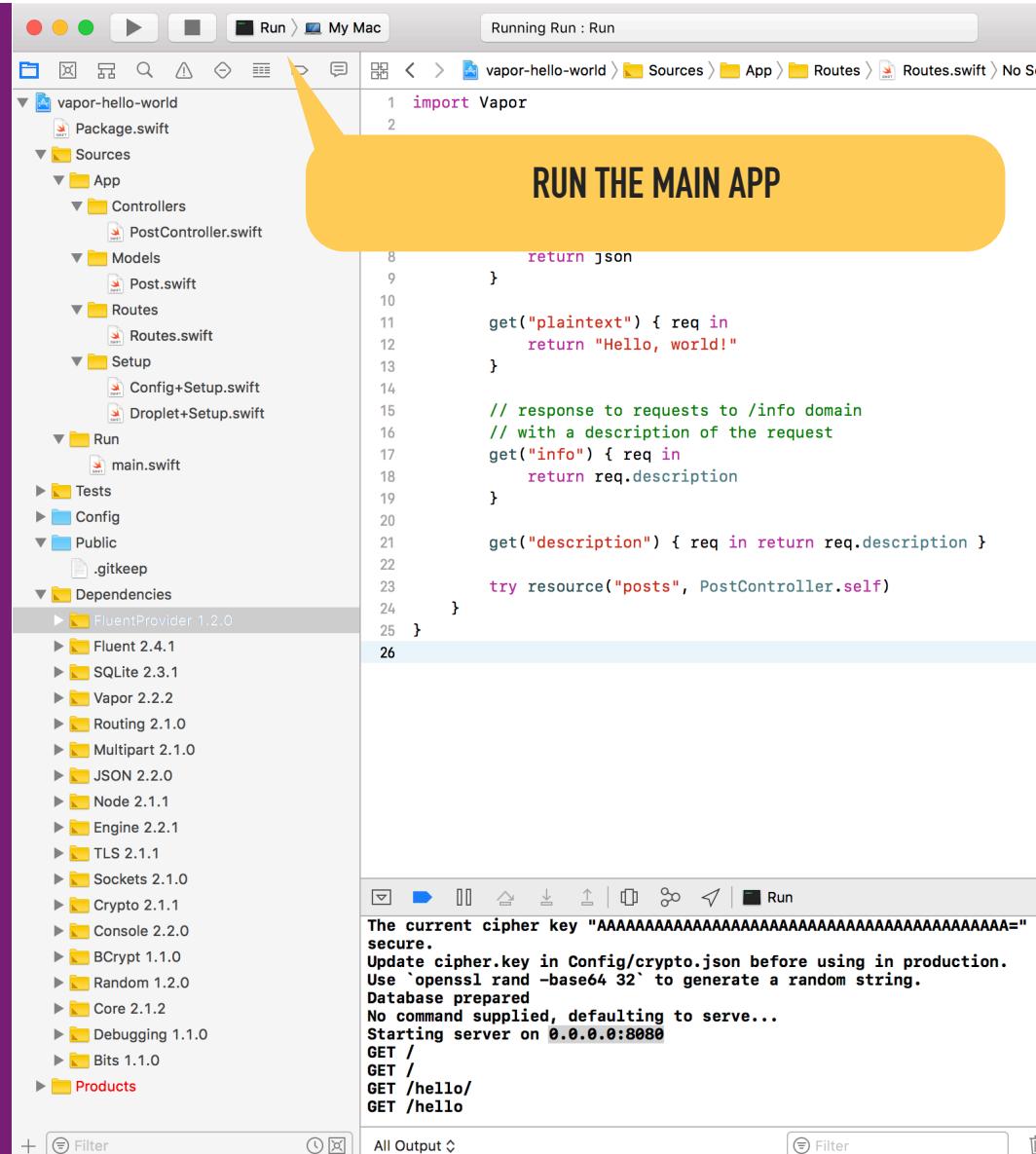
drop.get { req in
    return try drop.view.make("welcome", [
        "message": drop.localization[req.lang, "welcome", "title"]
    ])
}

No command supplied, defaulting to serve...
No preparations.
Server 'default' starting at 0.0.0.0:8080
```

The left sidebar shows the project structure of "vapor-hello-world" with files like Package.swift, main.swift, and various models/controllers.

VAPOR

- Builds all the package dependencies and links them



The current cipher key "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=" is secure.
Update cipher.key in Config/crypto.json before using in production.
Use `openssl rand -base64 32` to generate a random string.
Database prepared
No command supplied, defaulting to serve...
Starting server on 0.0.0.0:8080
GET /
GET /
GET /hello/
GET /hello

RUN THE MAIN APP

```
import Vapor

get("plaintext") { req in
    return "Hello, world!"
}

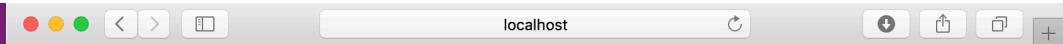
// response to requests to /info domain
// with a description of the request
get("info") { req in
    return req.description
}

get("description") { req in
    return req.description
}

try resource("posts", PostController.self)
```

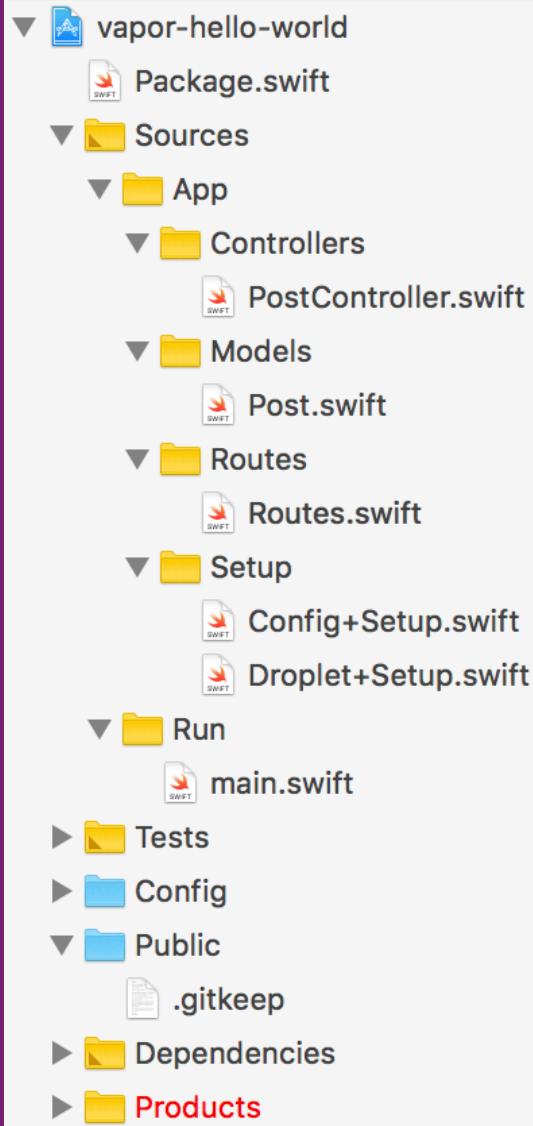
VAPOR

- Server runs on [http://localhost:
8080](http://localhost:8080)



VAPOR

- File structure is designed to make testing easier 🙌



```
1 import Vapo
2
3 extension D
4     func se
5         get
6
7
8
9
10
11     get
12
13 }
14
15 // 
16 // 
17 get
18
19 }
20
21 get
22
23 try
24
25 }
26
```

VAPOR

The screenshot shows a Mac OS X desktop with an Xcode window open. The title bar says "Running App : App". The file browser sidebar shows a project structure for "vapor-hello-world" with files like "Package.swift", "Sources/main.swift", and "Tests". The main editor pane contains Swift code for a Vapor application:

```
import Vapor

let drop = Droplet()

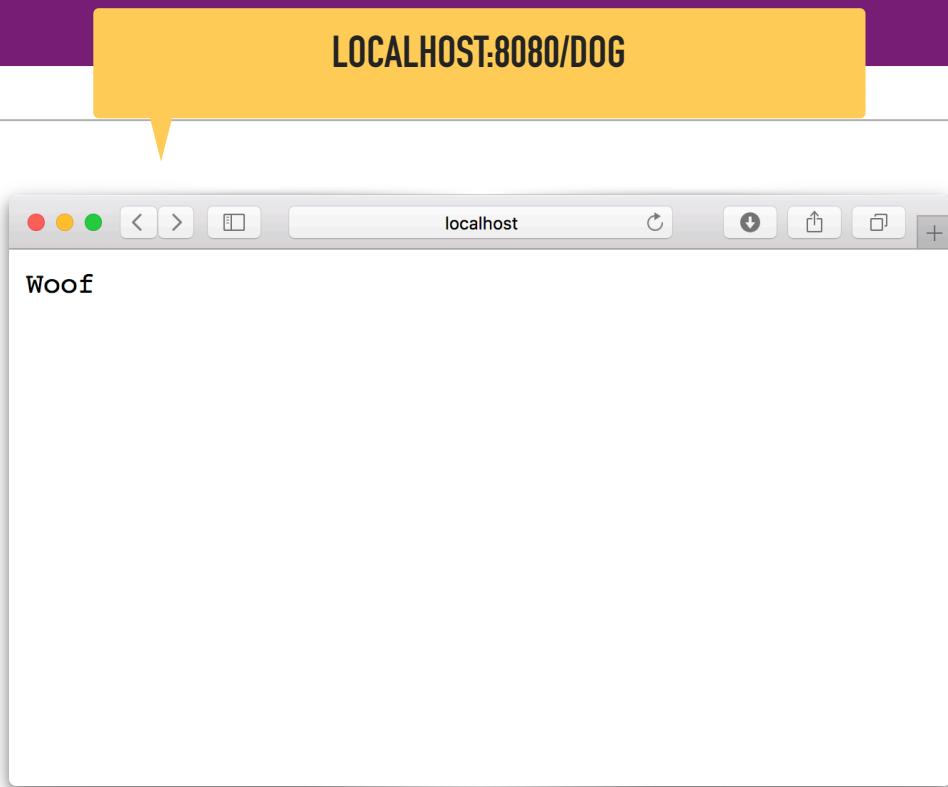
drop.get { req in
    return try JSON(node:
        ["message": "Hello Vapor"]
    )
}

drop.run()
```

A small preview window in the bottom right corner shows the output of the application running on "localhost", displaying the JSON response {"message": "Hello Vapor"}.

VAPOR

```
vapor-hello-world > Sources > App > main.swift > No Selection  
import Vapor  
  
let drop = Droplet()  
  
drop.get { req in  
    return try JSON(node:  
        ["message": "Hello Vapor"]  
    )  
}  
  
drop.get("dog") { req in  
    return "Woof"  
}  
  
drop.get("dog", "speak") { req in  
    return "Bark! Bark!"  
}  
  
drop.run()
```



VAPOR

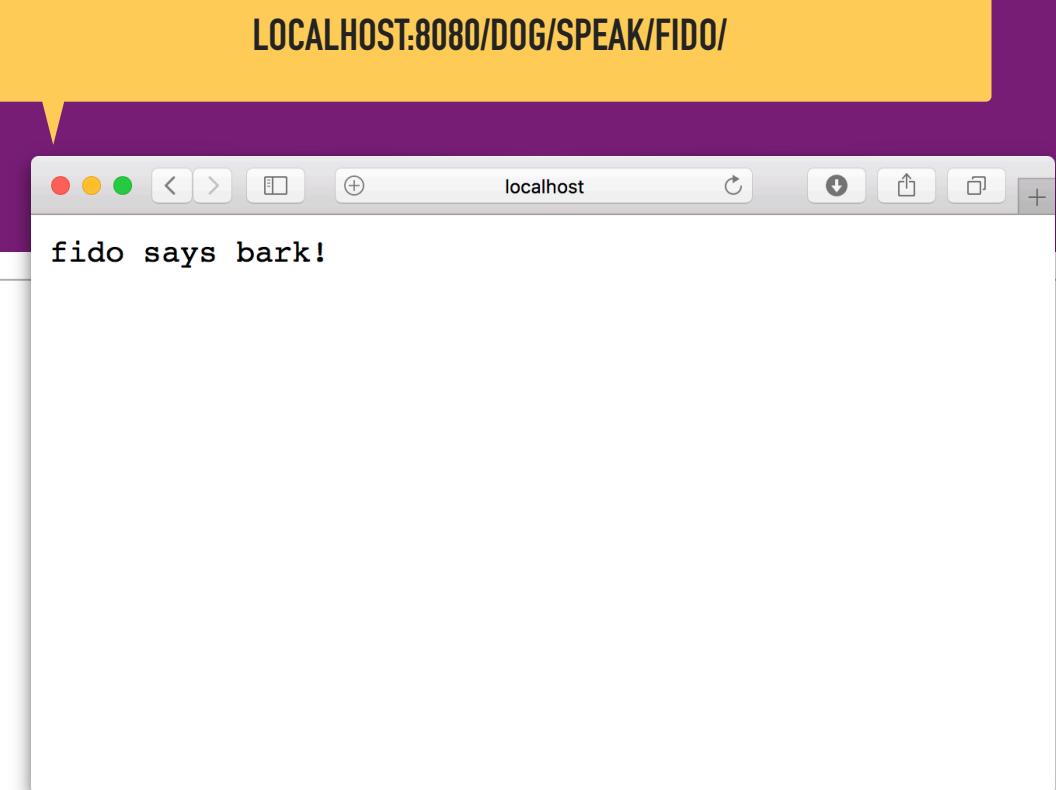
```
drop.get { req in
    return try JSON(node:
        ["message": "Hello Vapor"]
    )
}

drop.get("dog") { req in
    return "Woof"
}

drop.get("dog", "speak") { req in
    return "Bark! Bark!"
}

drop.get("dog", "speak", String.self) { req, name in
    return "\(name) says bark!"
}

drop.run()
```



VAPOR

LOCALHOST:8080/DOG/SPEAK/1/



```
drop.get("dog", "speak", Int.self) { req, number in
    return "My favorite number is \(number)"
}
```

DEPLOY VAPOR APP TO HEROKU

DEPLOY VAPOR APP TO HEROKU

- Heroku is a platform as a service company
- Free 'hobby' tier to play around



Log in to your account

Email address

Password

Log In

New to Heroku? [Sign Up](#)

[Log in via SSO](#) [Forgot your password?](#)

DEPLOY VAPOR APP TO HEROKU

```
# Install tools  
brew install heroku
```

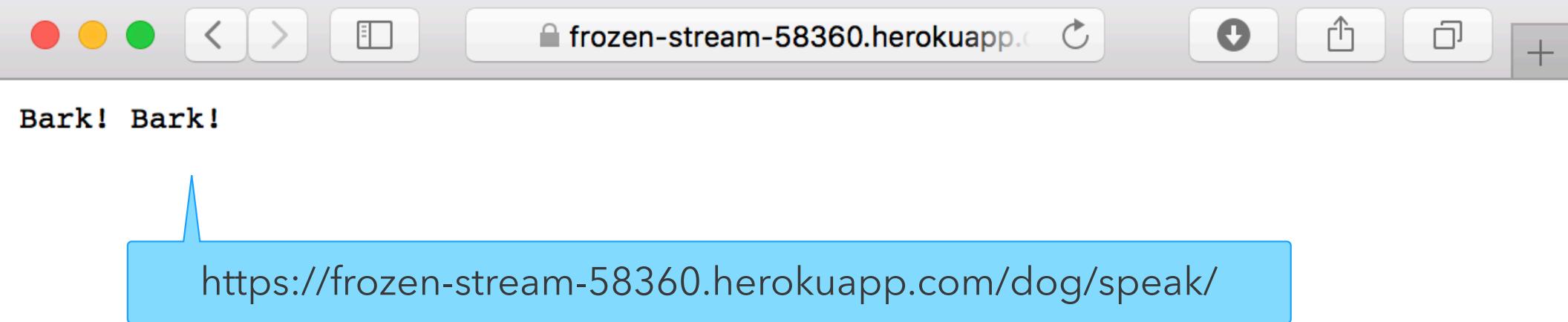
```
# Set credentials  
heroku login
```

```
# Upload  
vapor heroku init
```

DEPLOY VAPOR APP TO HEROKU

```
Building on Heroku ... ~5-10 minutes [D] •
Building on Heroku ... ~5-10 minutes [D]
Building on Heroku ... ~5-10 minutes [Do]
Building on Heroku ... ~5-10 minutes [Do]
Building on Heroku ... ~5-10 minutes [Do] •
Building on Heroku ... ~5-10 minutes [Done]
Spinning up dynos [Done]
Visit https://dashboard.heroku.com/apps/
App is live on Heroku, visit
https://frozen-stream-58360.herokuapp.com/ | https://git.heroku.com/frozen-stream-58360.git
```

DEPLOY VAPOR APP TO HEROKU



VAPOR WITH POSTGRESQL

VAPOR WITH POSTGRESQL

- Vapor support many different databases
 - MySQL
 - MongoDB
 - PostgreSQL
- Databases run independently of the web app

The world's most advanced open source database.

Home | About | Download | Documentation | Community | Developers | Support | Your account

11th May 2017

PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21 Released!

The PostgreSQL Global Development Group is pleased to announce the availability of PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21.

These new releases contain bug fixes over previous releases. All users should plan to upgrade their systems as soon as possible.

» [Release Announcement](#)
» [Release Notes](#)
» [Download](#)



FEATURED USER

... mission-critical technology tasks can continue to depend on the power, flexibility and robustness of PostgreSQL.

Ram Mohan, CTO, [Afiliias](#)
» [Case Studies](#) | [More Quotes](#) | [Featured Users](#)

LATEST NEWS

2017-05-11
[2017-05-11 Security Update Release](#)

2017-05-07
[pg_chameleon 1.0 released](#)

2017-05-04
[Announcing The Release Of pglogical 2.0](#)

2017-04-25
[DB Doc 4.1 released](#)

2017-04-19
[New version of MySQL-to-PostgreSQL has been released](#)

2017-04-07
[PgComment for PostgreSQL released](#)

2017-03-20
[Announcing AgensGraph v1.1 Release](#)

» [More](#) | [Submit News](#) | [RSS](#)

UPCOMING EVENTS

2017-05-18 – 2017-05-20
[PostgreSQL Booth at PyCon 2017](#)
(Portland, Oregon, United States)

2017-05-23 – 2017-05-26
[PGCon 2017](#)
(Ottawa, Ontario, Canada)

2017-06-08
[Pg Day France 2017](#)
(Toulouse, France)

2017-06-09
[PgDay Argentina 2017](#)
(Santa Fe, Santa Fe, Argentina)

2017-06-26 – 2017-06-28
[Postgres Vision 2017](#)
(Boston, MA, United States)

» [More](#) | [Submit Event](#) | [RSS](#)

UPCOMING TRAINING

There are 11 training events in 3 countries scheduled over the next six months from PostgresCourse.com,

LATEST RELEASES

9.6.3 · May 11, 2017 · [Notes](#)
9.5.7 · May 11, 2017 · [Notes](#)
9.4.12 · May 11, 2017 · [Notes](#)
9.3.17 · May 11, 2017 · [Notes](#)
9.2.21 · May 11, 2017 · [Notes](#)

[Download](#) | [RSS](#)
Why should I upgrade?
Upcoming releases

SHORTCUTS

» [Security](#)
» [International Sites](#)
» [Mailing Lists](#)
» [Wiki](#)
» [Report a Bug](#)
» [FAQs](#)

SUPPORT US

PostgreSQL is free. Please support our work by making a [donation](#).

PLANET POSTGRES

2017-05-16
[Joshua Drake: PGConf US Local: Seattle Call for Papers](#)

2017-05-16
[Michael Paquier: Postgres 10 highlight - Incompatible changes](#)

2017-05-15
[Bruce Momjian: RAID Controllers and SSDs](#)

2017-05-15
[Holly Orr: DevOps: Can't we just all get along?](#)

2017-05-13
[Federico Campoli: pg_chameleon v1.1 out](#)

2017-05-13
[Leo Hsu and Regina Obe: PostgreSQL JQuery extension Windows binaries](#)

2017-05-12
[Bruce Momjian: Postgres Window Magic](#)

» [More](#) | [RSS](#)

VAPOR WITH POSTGRESQL

- Importing Vapor also imports the `Fluent` framework
- ORM tool for Swift
 - Works with different databases
- Can be used on app for consistency

vapor / fluent

Code Issues 0 Pull requests 0 Wiki Pulse Graphs

Swift models, relationships, and querying for NoSQL and SQL databases.

vapor orm swift database sql nosql

608 commits 3 branches 97 releases 31 contributors

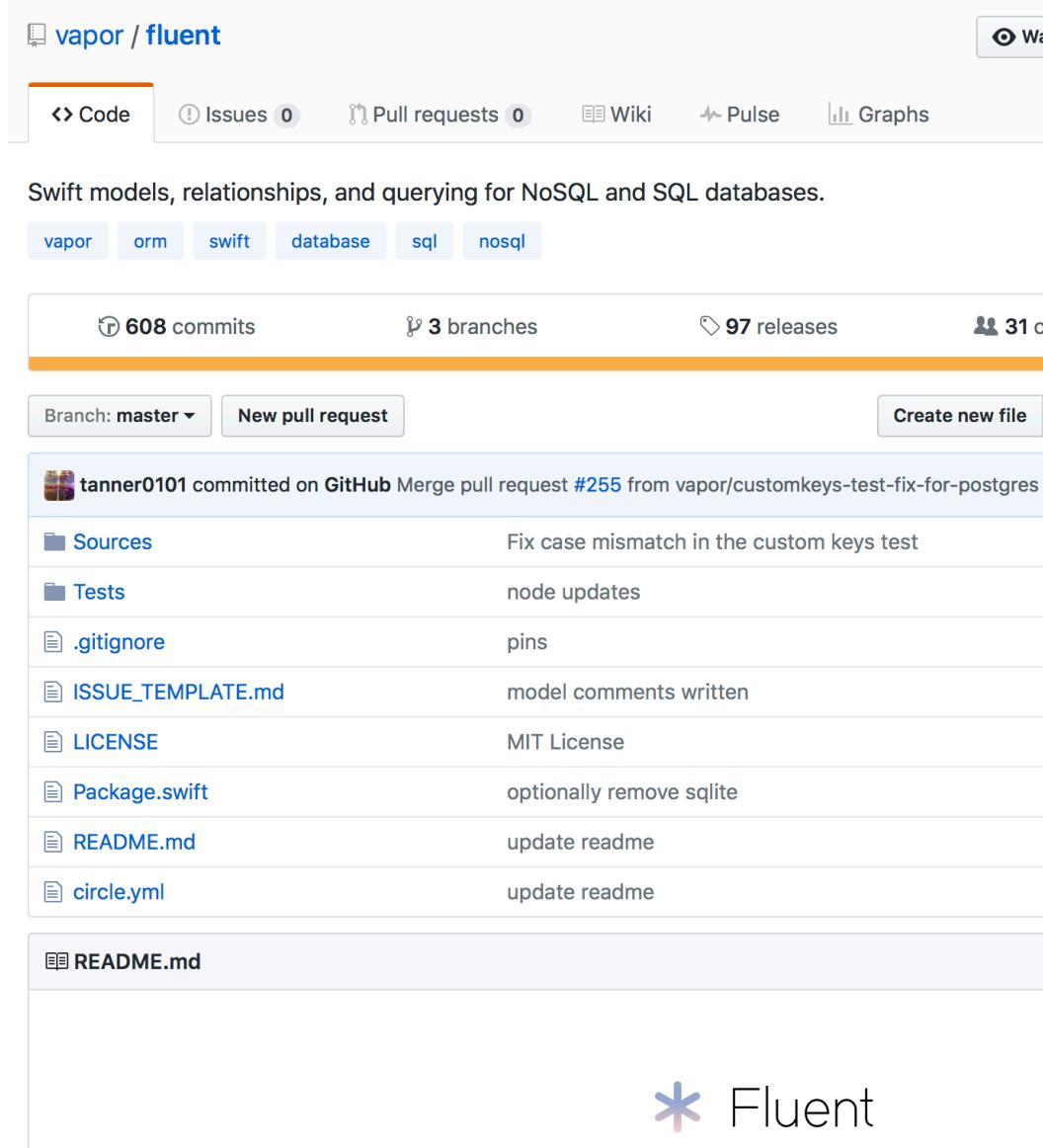
Branch: master New pull request Create new file

tanner0101 committed on GitHub Merge pull request #255 from vapor/customkeys-test-fix-for-postgres

Sources	Fix case mismatch in the custom keys test
Tests	node updates
.gitignore	pins
ISSUE_TEMPLATE.md	model comments written
LICENSE	MIT License
Package.swift	optionally remove sqlite
README.md	update readme
circle.yml	update readme

README.md

* Fluent



VAPOR WITH POSTGRESQL

- Import package
- Configure droplet
- Add configuration file

The world's most advanced open source database.

Home | About | Download | Documentation | Community | Developers | Support | Your account

11th May 2017

PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21 Released!

The PostgreSQL Global Development Group is pleased to announce the availability of PostgreSQL 9.6.3, 9.5.7, 9.4.12, 9.3.17 and 9.2.21.

These new releases contain bug fixes over previous releases. All users should plan to upgrade their systems as soon as possible.

» [Release Announcement](#)
» [Release Notes](#)
» [Download](#)



FEATURED USER

... mission-critical technology tasks can continue to depend on the power, flexibility and robustness of PostgreSQL.

Ram Mohan, CTO, Afilias

» [Case Studies](#) | [More Quotes](#) | [Featured Users](#)

LATEST NEWS

2017-05-11
[2017-05-11 Security Update Release](#)

2017-05-07
[pg_chameleon 1.0 released](#)

2017-05-04
[Announcing The Release Of pglogical 2.0](#)

2017-04-25
[DB Doc 4.1 released](#)

2017-04-19
[New version of MySQL-to-PostgreSQL has been released](#)

2017-04-07
[PgComment for PostgreSQL released](#)

2017-03-20
[Announcing AgensGraph v1.1 Release](#)

» [More](#) | [Submit News](#) | [RSS](#)

UPCOMING EVENTS

2017-05-18 – 2017-05-20
[PostgreSQL Booth at PyCon 2017](#)
(Portland, Oregon, United States)

2017-05-23 – 2017-05-26
[PGCon 2017](#)
(Ottawa, Ontario, Canada)

2017-06-08
[PG Day France 2017](#)
(Toulouse, France)

2017-06-09
[PgDay Argentina 2017](#)
(Santa Fe, Santa Fe, Argentina)

2017-06-26 – 2017-06-28
[Postgres Vision 2017](#)
(Boston, MA, United States)

» [More](#) | [Submit Event](#) | [RSS](#)

UPCOMING TRAINING

There are 11 training events in 3 countries scheduled over the next six months from PostgresCourse.com,

LATEST RELEASES

9.6.3 · May 11, 2017 · [Notes](#)
9.5.7 · May 11, 2017 · [Notes](#)
9.4.12 · May 11, 2017 · [Notes](#)
9.3.17 · May 11, 2017 · [Notes](#)
9.2.21 · May 11, 2017 · [Notes](#)

[Download](#) | [RSS](#)
[Why should I upgrade?](#)
[Upcoming releases](#)

SHORTCUTS

» [Security](#)
» [International Sites](#)
» [Mailing Lists](#)
» [Wiki](#)
» [Report a Bug](#)
» [FAQs](#)

SUPPORT US

PostgreSQL is free. Please support our work by making a [donation](#).

PLANET POSTGRES

2017-05-16
[Joshua Drake: PGConf US Local: Seattle Call for Papers](#)

2017-05-16
[Michael Paquier: Postgres 10 highlight - Incompatible changes](#)

2017-05-15
[Bruce Momjian: RAID Controllers and SSD's](#)

2017-05-15
[Holly Orr: DevOps: Can't we just all get along?](#)

2017-05-13
[Federico Campoli: pg_chameleon v1.1 out](#)

2017-05-13
[Leo Hsu and Regina Obe: PostgreSQL JQuery extension Windows binaries](#)

2017-05-12
[Bruce Momjian: Postgres Window Magic](#)

» [More](#) | [RSS](#)

VAPOR WITH POSTGRESQL

```
# Install postgres
brew install postgres

# Start postgres server
postgres -D /usr/local/var/postgres/

# Create a database
createdb dogs
```

VAPOR WITH POSTGRESQL

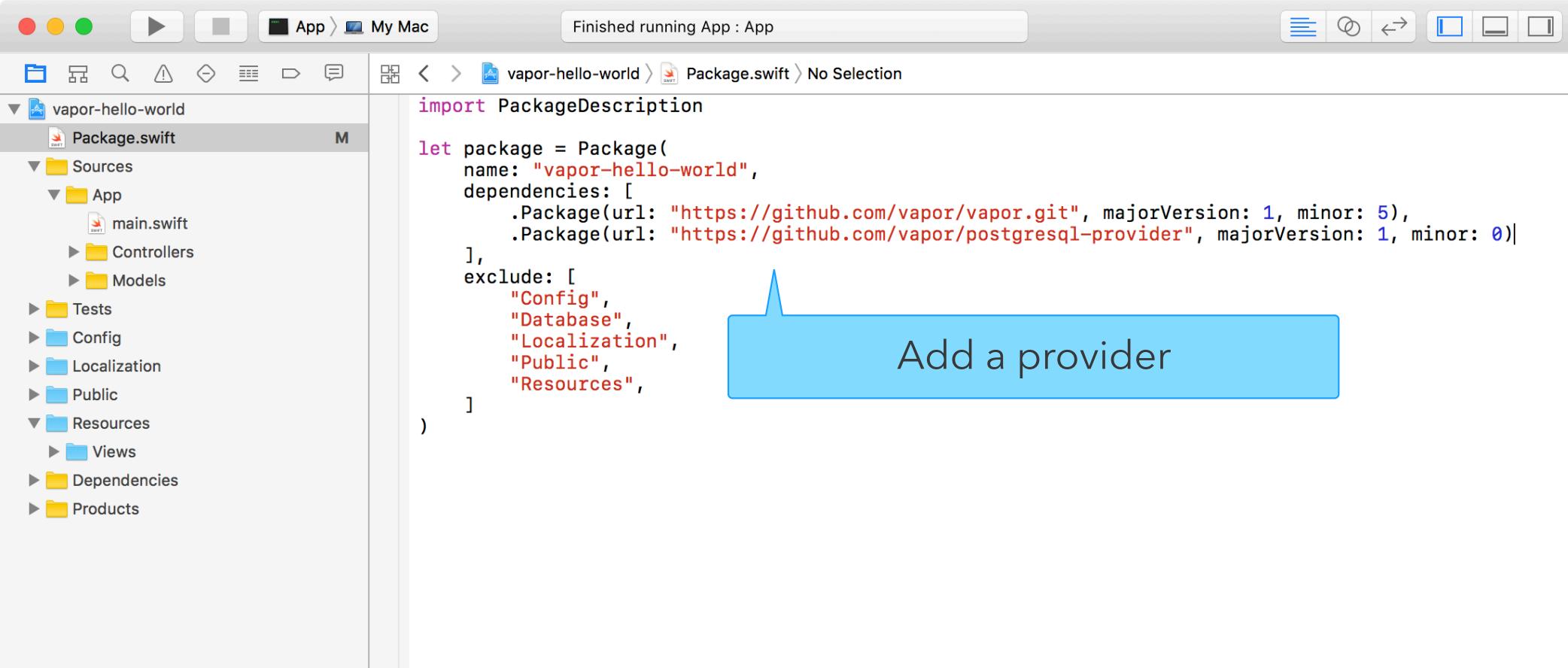
```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[592 % psql
psql (9.6.3)
Type "help" for help.

[tabinkowski=# \l
                                         List of databases
  Name | Owner | Encoding | Collate | Ctype | Access privileges
-----+-----+-----+-----+-----+-----+
dogs   | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 |
postgres | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 |
tabinkowski | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 |
template0 | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 | =c/tabinkowski          +
                                         |                   tabinkowski=CTc/tabinkowski
template1 | tabinkowski | UTF8    | en_US.UTF-8 | en_US.UTF-8 | =c/tabinkowski          +
                                         |                   tabinkowski=CTc/tabinkowski
(5 rows)

tabinkowski=# ]
```

- Vapor support many different databases
 - MySQL

VAPOR WITH POSTGRESQL



Finished running App : App

vapor-hello-world

Package.swift

Sources

App

main.swift

Controllers

Models

Tests

Config

Localization

Public

Resources

Views

Dependencies

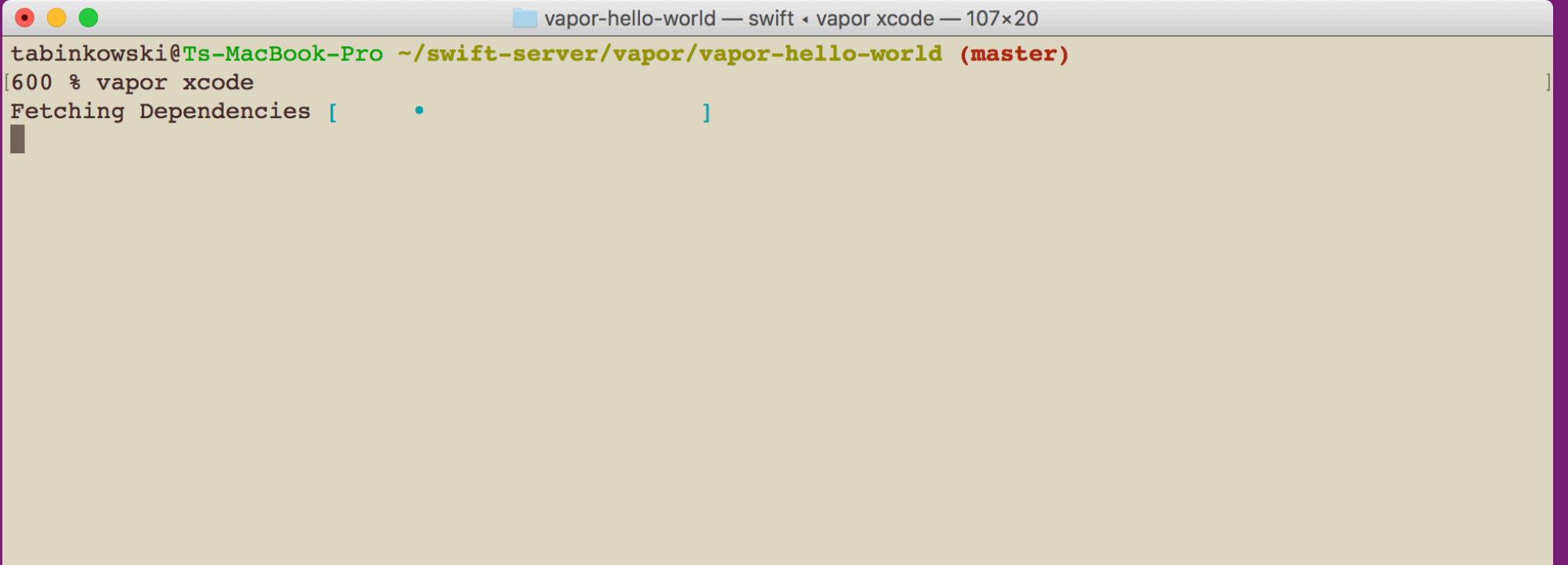
Products

```
import PackageDescription

let package = Package(
    name: "vapor-hello-world",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 1, minor: 5),
        .Package(url: "https://github.com/vapor/postgresql-provider", majorVersion: 1, minor: 0)
    ],
    exclude: [
        "Config",
        "Database",
        "Localization",
        "Public",
        "Resources",
    ]
)
```

Add a provider

VAPOR WITH POSTGRESQL



A screenshot of a macOS terminal window titled "vapor-hello-world — swift ▾ vapor xcode — 107x20". The window shows the command "tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)" followed by "[600 % vapor xcode". Below this, the text "Fetching Dependencies [•]" is displayed, indicating a progress bar.

- Update the project and rebuild your framework in Xcode

VAPOR WITH POSTGRESQL

< > vapor-hello-world > Sources > App > main.swift > No Selection

```
import Vapor
import VaporPostgreSQL

let drop = Droplet()

// Add provider
do {
    try drop.addProvider(VaporPostgreSQL.Provider.self)
} catch {
    assertionFailure("Error adding provider: \(error)")
}
```

Add a provider

VAPOR WITH POSTGRESQL

vapor-hello-world | Build **Succeeded**

vapor-hello-world > Config > secrets > postgresql.json > No Selection

```
{
    "host": "127.0.0.1",
    "user": "tabinkowski",
    "password": "",
    "database": "dogs",
    "port": 5432
}
```

Database configuration file

Add a new file

vapor-hello-world

- Package.swift
- Sources
 - App
 - main.swift
 - Controllers
 - Models
- Tests
- Config
 - app.json
 - clients.json
 - crypto.json
 - droplet.json
 - production
 - secrets
 - postgresql.json
 - servers.json
- Localization
- Public
- Resources
- Dependencies

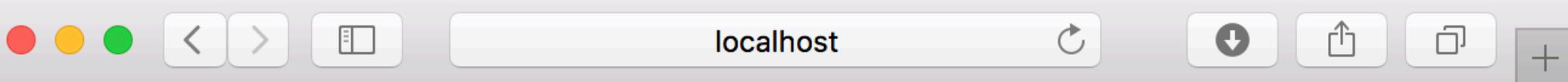
VAPOR WITH POSTGRESQL

```
// Add provider
do {
    try drop.addProvider(VaporPostgreSQL.Provider.self)
} catch {
    assertionFailure("Error adding provider: \(error)")
}

// Route to print out the db version
drop.get("version") { req in
    if let db = drop.database?.driver as? PostgreSQLDriver {
        let version = try db.raw("SELECT version()")
        return try JSON(node: version)
    } else {
        return "No database connection"
    }
}
```

Print out database version

VAPOR WITH POSTGRESQL



IT'S BORING, BUT IT WORKS

VAPOR WITH POSTGRESQL

The screenshot shows the Xcode interface with a Vapor application running. The title bar says "Running App : App". The file path in the sidebar is "vapor-hello-world > Sources > App > main.swift > No Selection". The code in main.swift is:

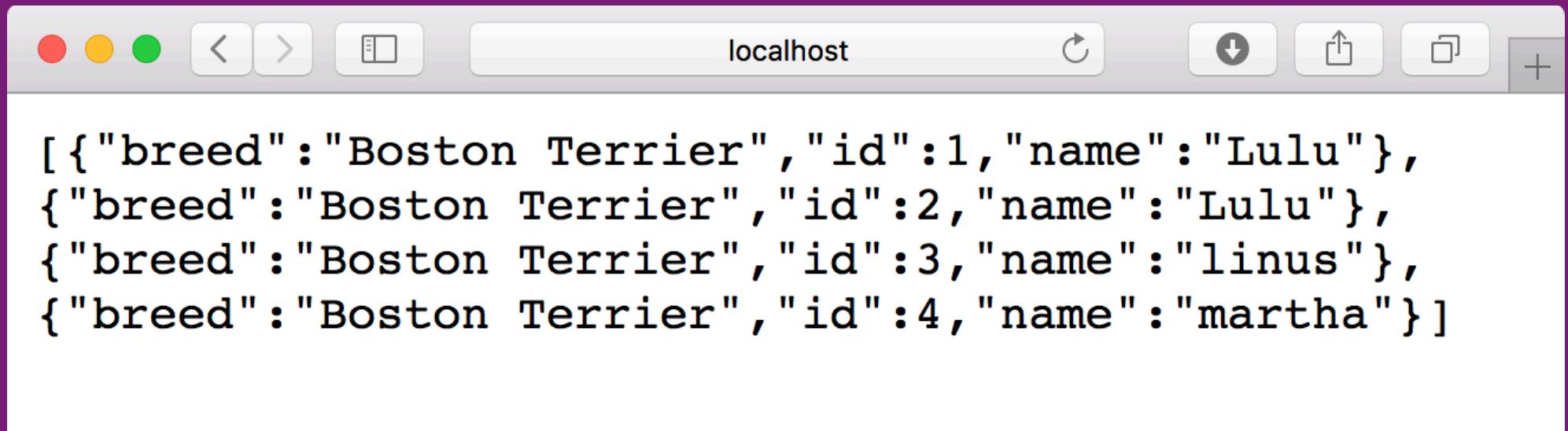
```
let drop = Droplet()
drop.preparations.append(Dog.self)

// Add provider
do {
    try drop.addProvider(VaporPostgreSQL.Provider.self)
} catch {
    assertionFailure("Error adding provider: \(error)")
}

drop.get("dog", "create") { req in
    let dog = [Dog(name:"Lulu", breed: "Boston Terrier"),
              Dog(name:"Snoopy", breed: "Beagle"),
              Dog(name:"Fifi", breed: "Poodle")]
    let dogNode = try dog.makeNode()
    let nodeDictionary = ["dogs": dogNode]
    return try JSON(node: nodeDictionary)
}
```

A yellow callout box points to the line `drop.preparations.append(Dog.self)` with the text "ADD DOG MODEL". Another yellow callout box points to the line `let dog = [Dog(name:"Lulu", breed: "Boston Terrier"), Dog(name:"Snoopy", breed: "Beagle"), Dog(name:"Fifi", breed: "Poodle")]` with the text "CREATE DATA".

VAPOR WITH POSTGRESQL



```
// List all dogs
drop.get("dog", "pound") { req in
    return try JSON(node: Dog.all().makeNode())
}
```

VAPOR WITH POSTGRESQL

- Check that data is being persisted using psql
 - \l - list db
 - \c - switch db
 - \d - list tables

```
vapor-hello-world — psql — 60x20
(5 rows)

tabinkowski=# \dt
No relations found.
tabinkowski=# \c dogs
You are now connected to database "dogs" as user "tabinkowski".
dogs=# select * from dogs;
 id | name      |      breed
----+-----+-----
    1 | Lulu      | Boston Terrier
    2 | Lulu      | Boston Terrier
    3 | linus     | Boston Terrier
    4 | martha    | Boston Terrier
    5 | martha    | Boston Terrier
    6 | martha    | Boston Terrier
    7 | martha    | Boston Terrier
(7 rows)

dogs=#
```

DEPLOY TO HEROKU WITH POSTGRESQL

DEPLOY TO HEROKU WITH POSTGRESQL

- Set up postgresql on heroku
 - Built-in, just need to add it
- Tell Postgresql where the database is located
 - Do not upload the secrets/postgresql.json file

```
{  
  "host": "127.0.0.1",  
  "user": "tabinkowski",  
  "password": "",  
  "database": "dogs",  
  "port": 5432  
}
```

DEPLOY TO HEROKU WITH POSTGRESQL

Add postgres to your heorku app

```
vapor-hello-world — -bash — 89x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
613 % heroku addons:create heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⚙ frozen-stream-58360... free
Database has been created and is available
! This database is empty. If upgrading, you can transfer
! data from another database with pg:copy
Created postgresql-vertical-49786 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
tabinkowski@Ts-MacBook-Pro /swift-server/vapor/vapor-hello-world (master)
614 %
```

New database

DEPLOY TO HEROKU WITH POSTGRESQL

```
vapor-hello-world bash 89x15
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[613 % heroku addons:create heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⚙️ frozen-stream-58360... free
Database has been created and is available
! This database is empty. If upgrading, you can transfer
! data from another database with pg:copy
Created postgresql-vertical-49786 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[614 % heroku config
==== frozen-stream-58360 Config Vars
DATABASE_URL: postgres://hzgtyzqowaztwa:4ad9847249b30c97f1835744ca06b1bae2b9012422c21d32d
4dad5e5088def04@ec2-23-23-227-188.compute-1.amazonaws.com:5432/d9hapt1pulfgrf
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
615 %
```

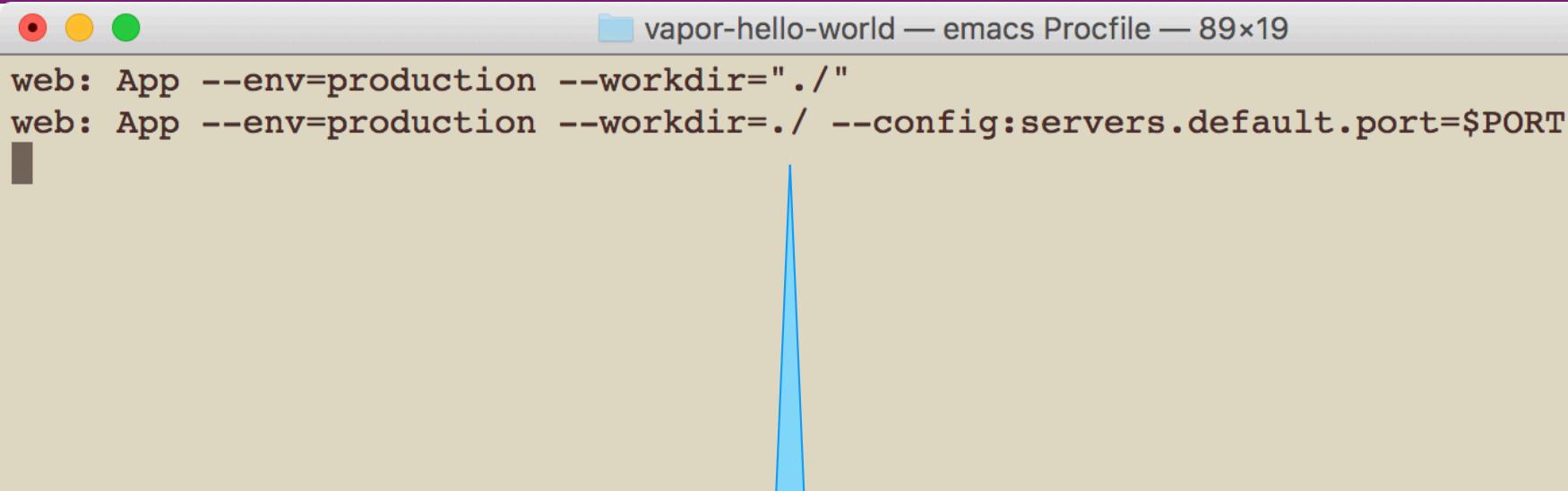
Database location

DEPLOY TO HEROKU WITH POSTGRESQL

```
vapor-hello-world — -bash — 89x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[618 % ls
Config                               Procfile
Localization                         Public
Package.pins                          README.md
Package.swift                         Resources
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
619 % ]
```

Update the procfile with the database location

DEPLOY TO HEROKU WITH POSTGRESQL



```
vapor-hello-world — emacs Procfile — 89x19
web: App --env=production --workdir=". /"
web: App --env=production --workdir= ./ --config:servers.default.port=$PORT
```

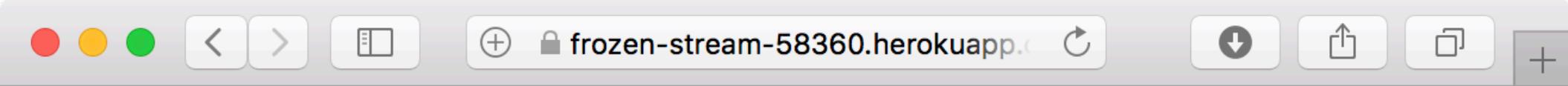
Update the procfile with the database location

DEPLOY TO HEROKU WITH POSTGRESQL

```
vapor-hello-world — -bash — 89x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
629 % git push heroku master
```

Push the changes to heroku

DEPLOY TO HEROKU WITH POSTGRESQL



```
[{"breed": "Boston Terrier", "id": 1, "name": "martha"}, {"breed": "Boston Terrier", "id": 2, "name": "martha"}]
```

<https://frozen-stream-58360.herokuapp.com/dog/pound/>

VAPOR WITH POSTGRESQL

```
// List all dogs
drop.get("dog", "pound") { req in
    return try JSON(node: Dog.all().makeNode())
}
```

Get all entries

VAPOR WITH POSTGRESQL

Filter

```
// Get all bostons
drop.get("query-boston") { req in
    return try JSON(node: Dog.query().filter("breed", .equals, "Boston Terrier").all().makeNode())
}
```

Query

VAPOR WITH POSTGRESQL

```
//  
drop.get("update-poodle") { req in  
    guard var dog = try Dog.query().first() else {  
        throw Abort.badRequest  
    }  
    dog.breed = "Poodle"  
    try dog.save()  
    return dog  
}
```

Get the first entry

Change values

Save

PERFECT

PERFECT

- Web framework and server for Swift server applications

The screenshot shows the homepage of perfect.org. At the top left is the Perfect logo (a stylized bird) and the text "perfect.org". At the top right is a three-line menu icon. Below the header are four main sections arranged in a grid:

- Get Started**: Features a large orange bird icon. Description: "Access Perfect's feature set and start your project." Call-to-action: "Get Started >>"
- Documentation**: Features an orange book icon. Description: "Get support from our comprehensive set of documentation." Call-to-action: "Documentation >>"
- Assistant**: Features an orange laptop icon. Description: "Meet the Perfect Assistant to help you do more." Call-to-action: "Get Started >>"
- Commercial**: Features an orange bar chart icon. Description: "Use Perfect to build apps for business." Call-to-action: "Get Started >>"

PERFECT

- Requirements
 - Xcode 9 Mac
 - Linux toolchain

The screenshot shows the homepage of perfect.org. At the top left is the Perfect logo (a stylized bird) and the URL 'perfect.org'. At the top right is a three-line menu icon. The page features four main sections arranged in a 2x2 grid:

- Get Started**: Features an orange bird icon. Description: "Access Perfect's feature set and start your project." Call-to-action: "Get Started >>"
- Documentation**: Features an open book icon. Description: "Get support from our comprehensive set of documentation." Call-to-action: "Documentation >>"
- Assistant**: Features an orange laptop icon. Description: "Meet the Perfect Assistant to help you do more." Call-to-action: "Get Started >>"
- Commercial**: Features an orange bar chart icon. Description: "Use Perfect to build apps for business." Call-to-action: "Get Started >>"

At the bottom of the page, there is a footer section with the text "Meet the Perfect Assistant to help you do more" and "Use Perfect to build apps for business".

PERFECT

```
[637 % cd perfect/
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect
[638 % ls
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect
[639 % mkdir perfect-hello-world
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect
[640 % cd perfect-hello-world/
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
Swift package manager to create an executable package
[641 % ls
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[642 % swift package init --type executable
Creating executable package: perfect-hello-world
Creating Package.swift
Creating .gitignore
Creating Sources/
Creating Sources/main.swift
Creating Tests/
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
643 %
```

PERFECT

```
perfect-hello-world — -bash ▶ postgres: stats collector process — 70x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
644 % swift package generate-xcodeproj
generated: ./perfect-hello-world.xcodeproj
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
645 %
```

Swift package manager to create an Xcode project

PERFECT

```
perfect-hello-world — bash ▶ postgres: stats collector process — 70x19
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
644 % swift package generate-xcodeproj
generated: ./perfect-hello-world.xcodeproj
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[645 % ls
Package.swift                               Tests
Sources                                     perfect-hello-world.xcodeproj
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[646 % ]
```

PERFECT

- Working command line application

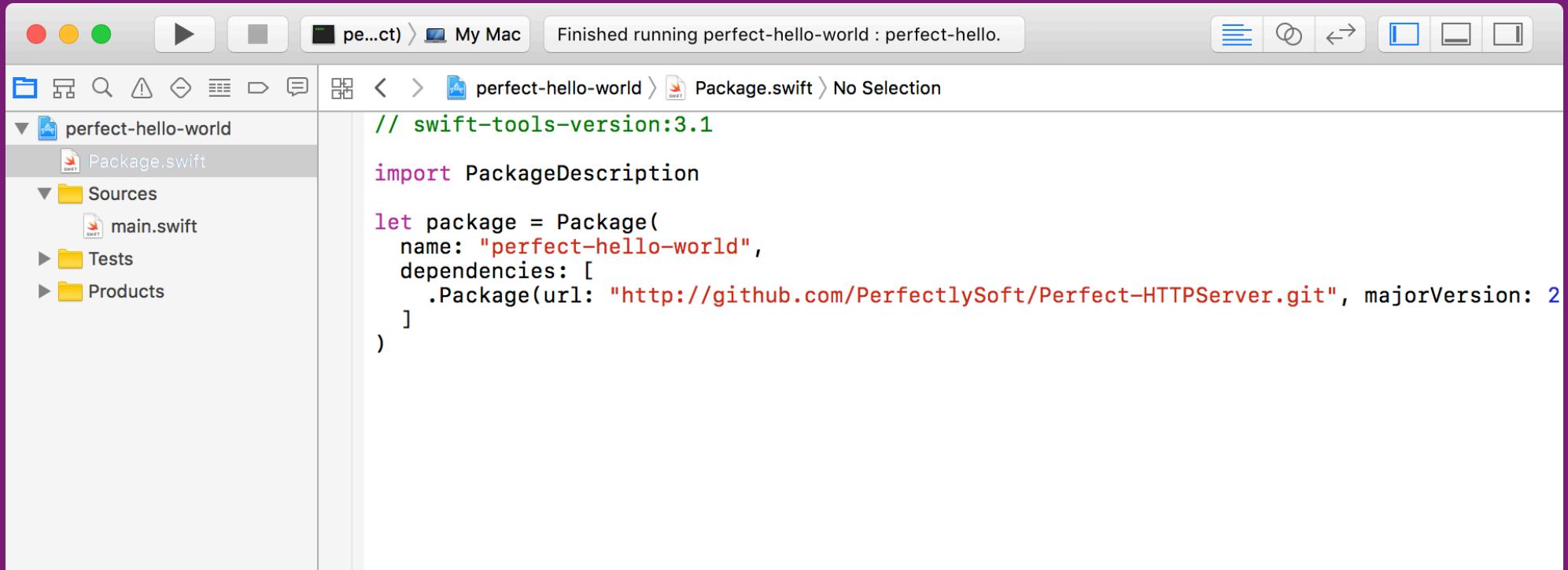
The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure for "perfect-hello-world". It includes a "Package.swift" file, a "Sources" folder containing "main.swift", a "Tests" folder, and a "Products" folder.
- Editor:** The code editor displays the content of "main.swift":

```
print("Hello, world!")
```
- Output Navigator:** The output window shows the results of running the program:

```
Hello, world!
Program ended with exit code: 0
```
- Bottom Bar:** Includes a "Filter" field and a "All Output" dropdown.

PERFECT



The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with a selected "Package.swift" file.
- Editor:** Displays the contents of the "Package.swift" file:

```
// swift-tools-version:3.1
import PackageDescription

let package = Package(
    name: "perfect-hello-world",
    dependencies: [
        .Package(url: "http://github.com/PerfectlySoft/Perfect-HTTPServer.git", majorVersion: 2)
    ]
)
```
- Toolbar:** Shows standard Xcode icons for file operations, including a play button indicating the application has run.
- Status Bar:** Shows the message "Finished running perfect-hello-world : perfect-hello."

- Add Perfect as a project dependency

PERFECT

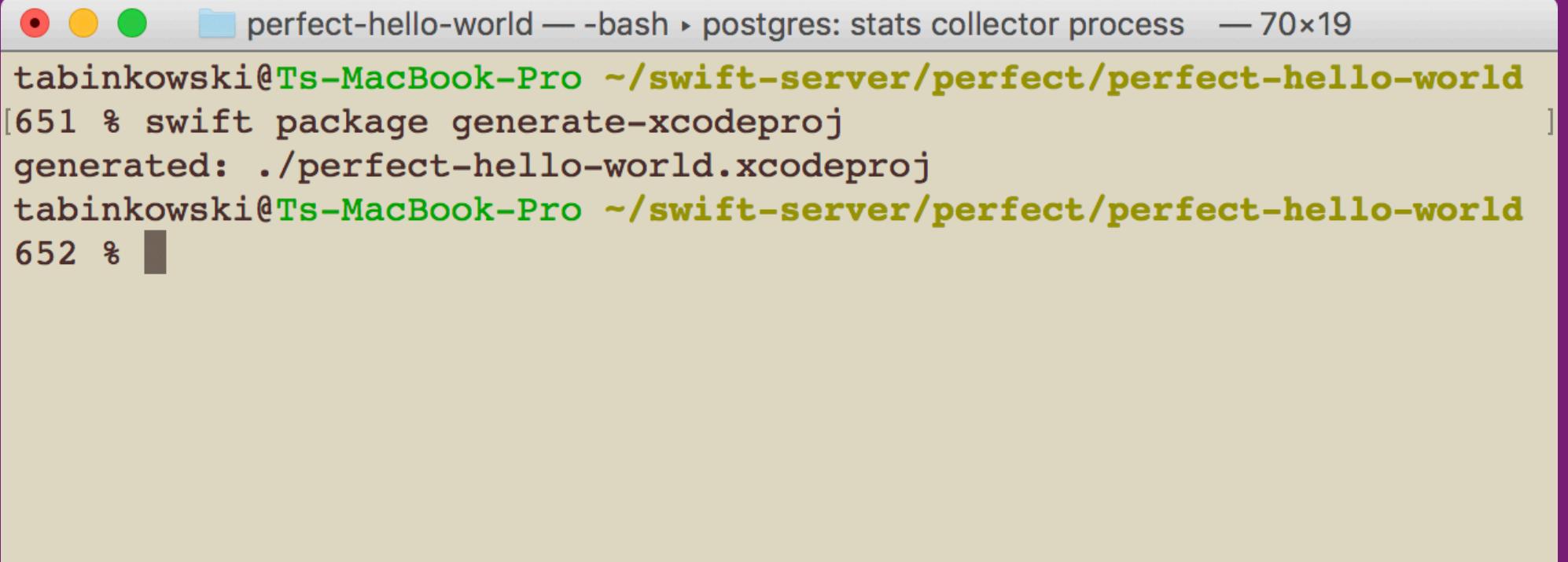


A screenshot of a macOS terminal window titled "perfect-hello-world — swift-package update ▶ git-remote-https — 70x19". The window shows the command "tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world [648 % swift package update" followed by the output "Fetching http://github.com/PerfectlySoft/Perfect-HTTPServer.git" and "Fetching https://github.com/PerfectlySoft/Perfect-HTTP.git". The terminal has a light beige background and a dark grey header bar.

```
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world [648 % swift package update
Fetching http://github.com/PerfectlySoft/Perfect-HTTPServer.git
Fetching https://github.com/PerfectlySoft/Perfect-HTTP.git
```

- Update the package manager

PERFECT



A screenshot of a macOS terminal window titled "perfect-hello-world — bash ▶ postgres: stats collector process — 70x19". The window shows the command "swift package generate-xcodeproj" being run in the directory "~/swift-server/perfect/perfect-hello-world". The output indicates that an Xcode project named "perfect-hello-world.xcodeproj" was generated. The terminal prompt "652 %" is visible at the bottom.

```
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[651 % swift package generate-xcodeproj
generated: ./perfect-hello-world.xcodeproj
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
652 %
```

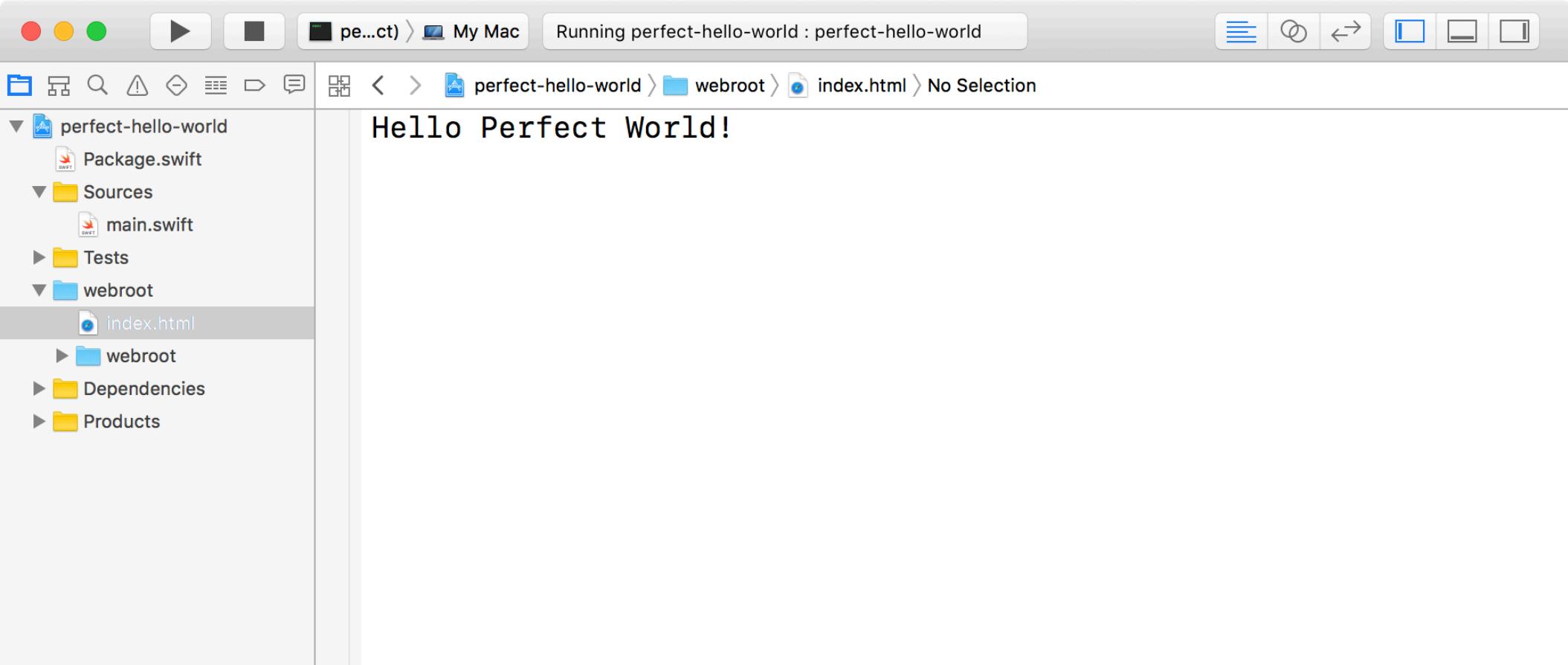
- Update the Xcode project

PERFECT

```
perfect-hello-world — -bash ▶ postgres: stats collector process — 70×19
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[653 % mkdir webroot
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[654 % touch webroot/hello.txt
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[655 % swift package generate-xcodeproj
generated: ./perfect-hello-world.xcodeproj
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
656 %
```

- Create a root directory for the web server

PERFECT



PERFECT

Create a server

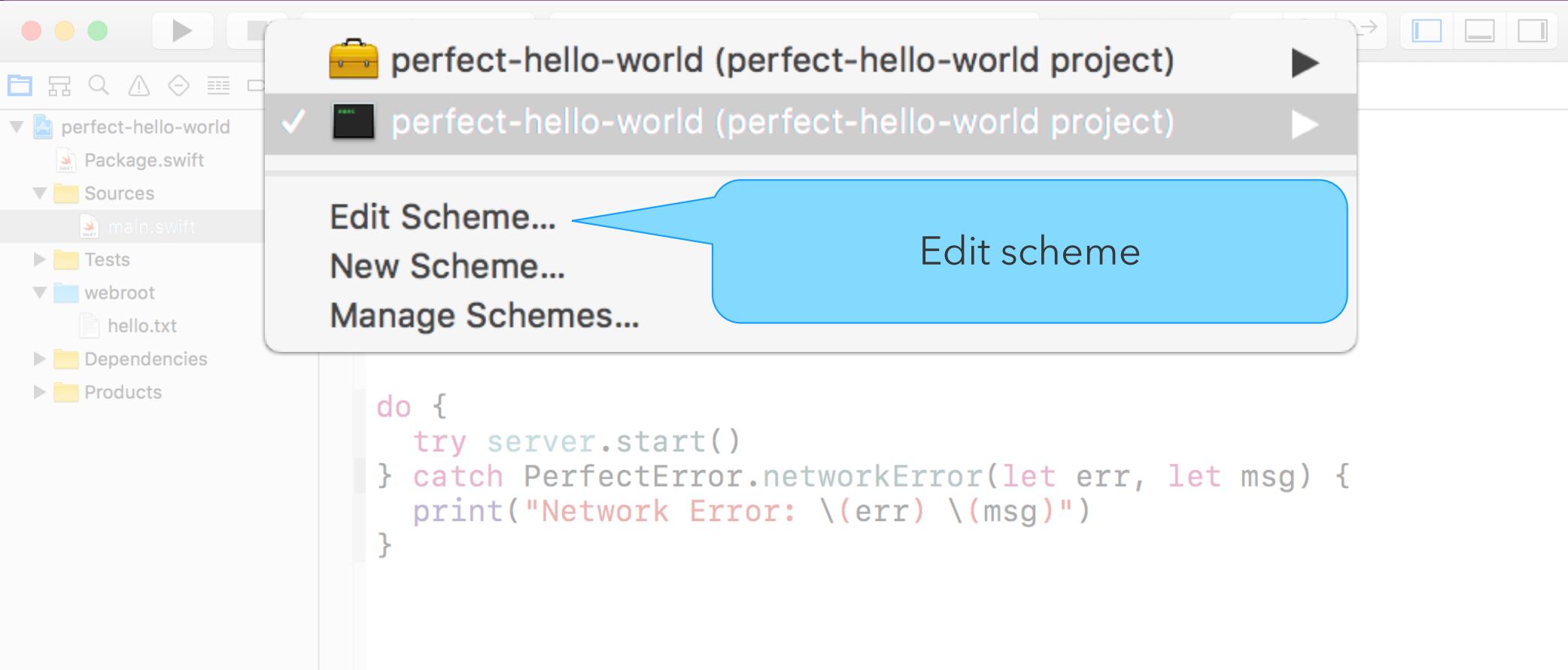
The screenshot shows a Mac OS X desktop with an Xcode interface. The title bar says "pe...ct) My Mac perfect-hello-world | Build Succeeded". The main area displays the code for "main.swift" in the "Sources" folder of the "perfect-hello-world" project. A blue callout bubble points from the text "Create a server" to the "server" variable in the code.

```
import PerfectLib
import PerfectHTTP
import PerfectHTTPServer

let server = HTTPServer()
server.serverPort = 8080
server.documentRoot = "webroot"

do {
    try server.start()
} catch PerfectError.networkError(let err, let msg) {
    print("Network Error: \(err) \(msg)")
}
```

PERFECT



PERFECT

Options

Build
1 target

Run
Debug

Test
Debug

Profile
Release

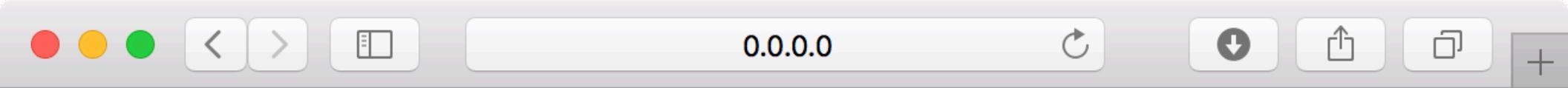
Analyze
Debug

Archive
Release

	Info	Arguments	Options	Diagnostics
Core Location	<input checked="" type="checkbox"/> Allow Location Simulation			
Default Location	<input type="text"/> None			
Application Data	<input type="text"/> None			
Routing App Coverage File	<input type="text"/> None			
GPU Frame Capture	<input type="button"/> Automatically Enabled			
Metal API Validation	<input type="button"/> Enabled			
Persistent State	<input type="checkbox"/> Launch application without state restoration			
Document Versions	<input checked="" type="checkbox"/> Allow debugging			
Working Directory	<input checked="" type="checkbox"/> Use custom working directory: <input type="text"/> /Users/tabinkowski/Google Drive/g-Teaching/uct			
Background Fetch	<input type="checkbox"/> Launch due to a background fetch event			
Localization Debugging	<input type="checkbox"/> Show non-localized strings			

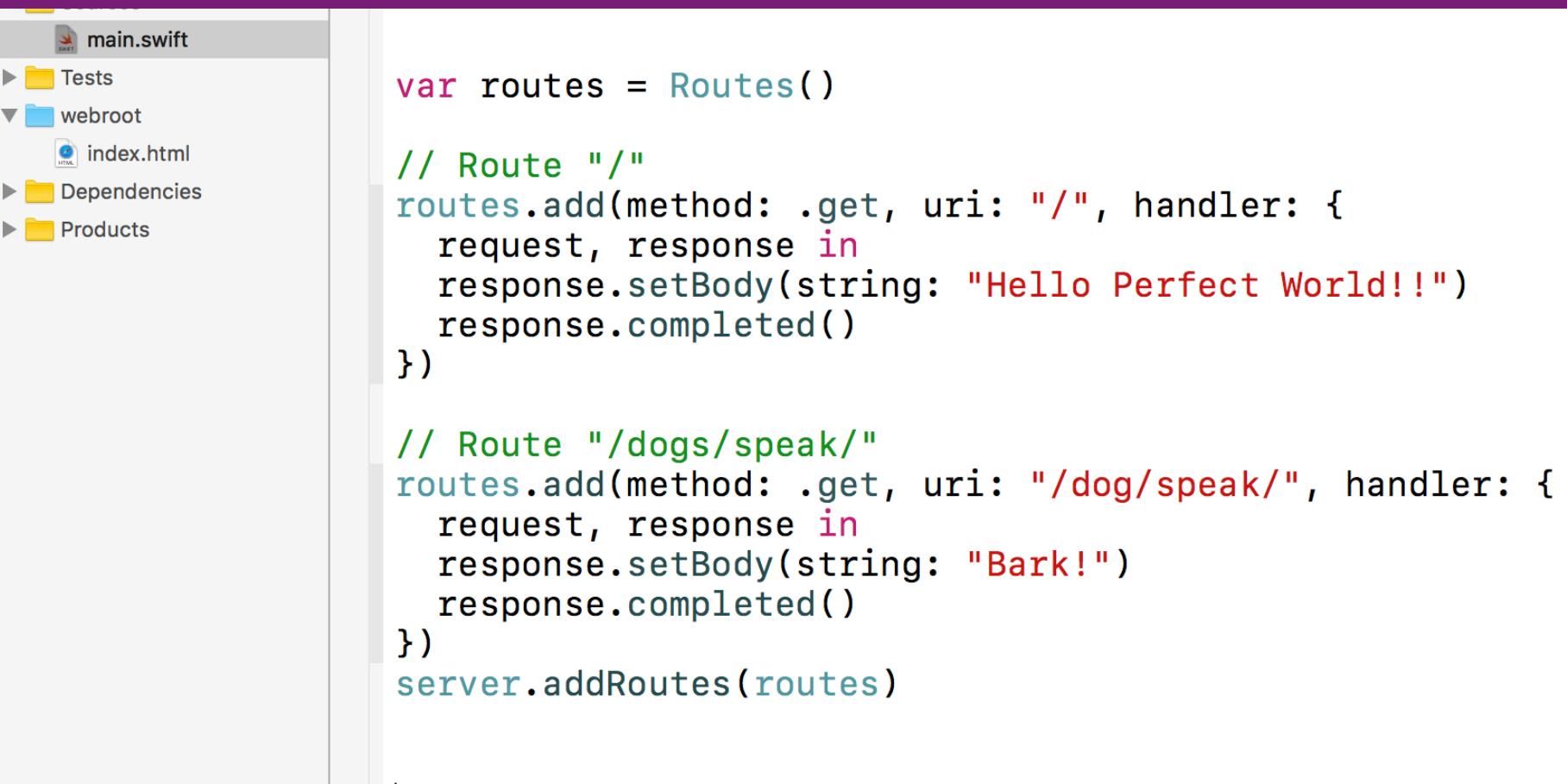
Working Directory

PERFECT



Hello Perfect World!

PERFECT



The screenshot shows a code editor with a sidebar and a main editing area. The sidebar on the left lists project files: Tests, webroot (which contains index.html), Dependencies, and Products. The main area displays the following Swift code:

```
main.swift
Tests
webroot
  index.html
Dependencies
Products

var routes = Routes()

// Route "/"
routes.add(method: .get, uri: "/", handler: {
    request, response in
    response.setBody(string: "Hello Perfect World!!")
    response.completed()
})

// Route "/dogs/speak/"
routes.add(method: .get, uri: "/dog/speak/", handler: {
    request, response in
    response.setBody(string: "Bark!")
    response.completed()
})
server.addRoutes(routes)
```

PERFECT

The image shows a macOS desktop environment. On the left, there is a Xcode project sidebar with the following structure:

- main.swift
- Tests
- webroot
 - index.html
- Dependencies
- Products

The main editor area displays the content of the `main.swift` file:

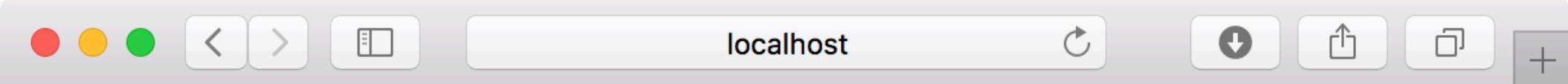
```
var routes = Route.Builder()  
// Route "/"  
routes.add(method: .get, uri: "/", handler: {  
    request, response in  
    response.setBody(string: "Bark!")  
    response.completed()  
})  
server.addRoutes(routes)
```

To the right of the editor, a browser window is open at the URL `localhost`. The page title is "Bark!" and the content of the page is "Bark!".

PERFECT

```
// Route "/dogs/{name}/"
routes.add(method: .get, uri: "/dog/speak/{name}/", handler: {
    request, response in
    guard let name = request.urlVariables["name"],
          let nameString = String(name) else {
        response.completed(status: .badRequest)
        return
    }
    response.setBody(string: "\(nameString) says Bark!")
    response.completed()
})
```

PERFECT



lassie says Bark!

<http://localhost:8088/dog/speak/lassie/>

PERFECT ASSISTANT

PERFECT ASSISTANT

- Set up new projects easily or download existing project templates
- Manage dependencies
- Create simultaneous macOS and Ubuntu builds on your local machine
- Push projects up to AWS, HEROKU, GCP



Perfect Assistant

Create, test and deploy your Xcode projects.

Manager to help you manage dependencies and Docker helps you build for Ubuntu, providing a smooth cross-platform build experience.

Docker is required to be installed and running to perform Xcode integrations and deployments.

Docker Status: **Active**
Docker is installed and running.

[Update Docker Image](#)



Import Existing Project



Create Empty SPM Project

To deploy on EC2, ensure the AWS command line tools are installed and enter your AWS security credentials.

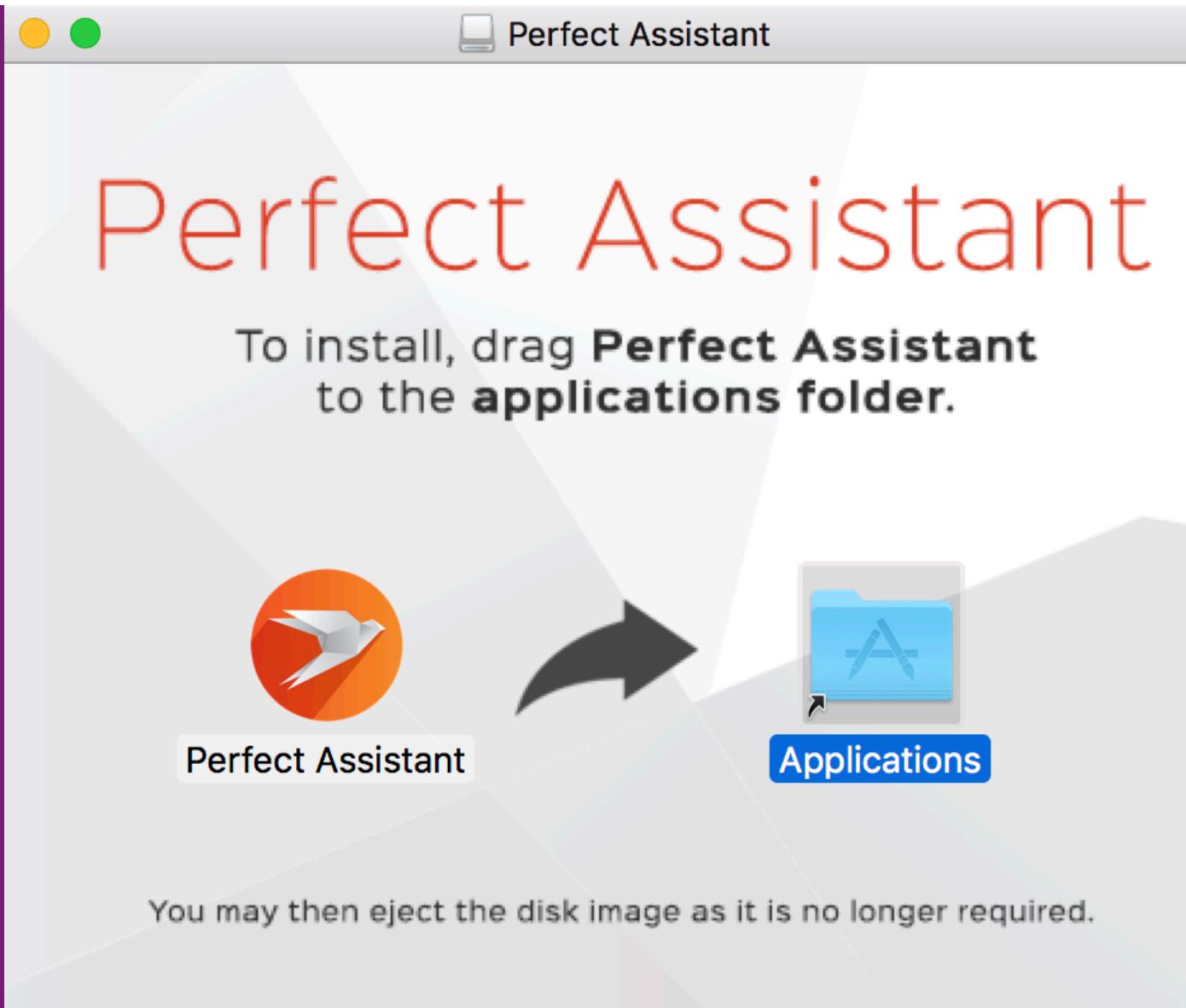
AWS Tools: **Active**
AWS command line tools are installed.

[Configure EC2 Credentials...](#)



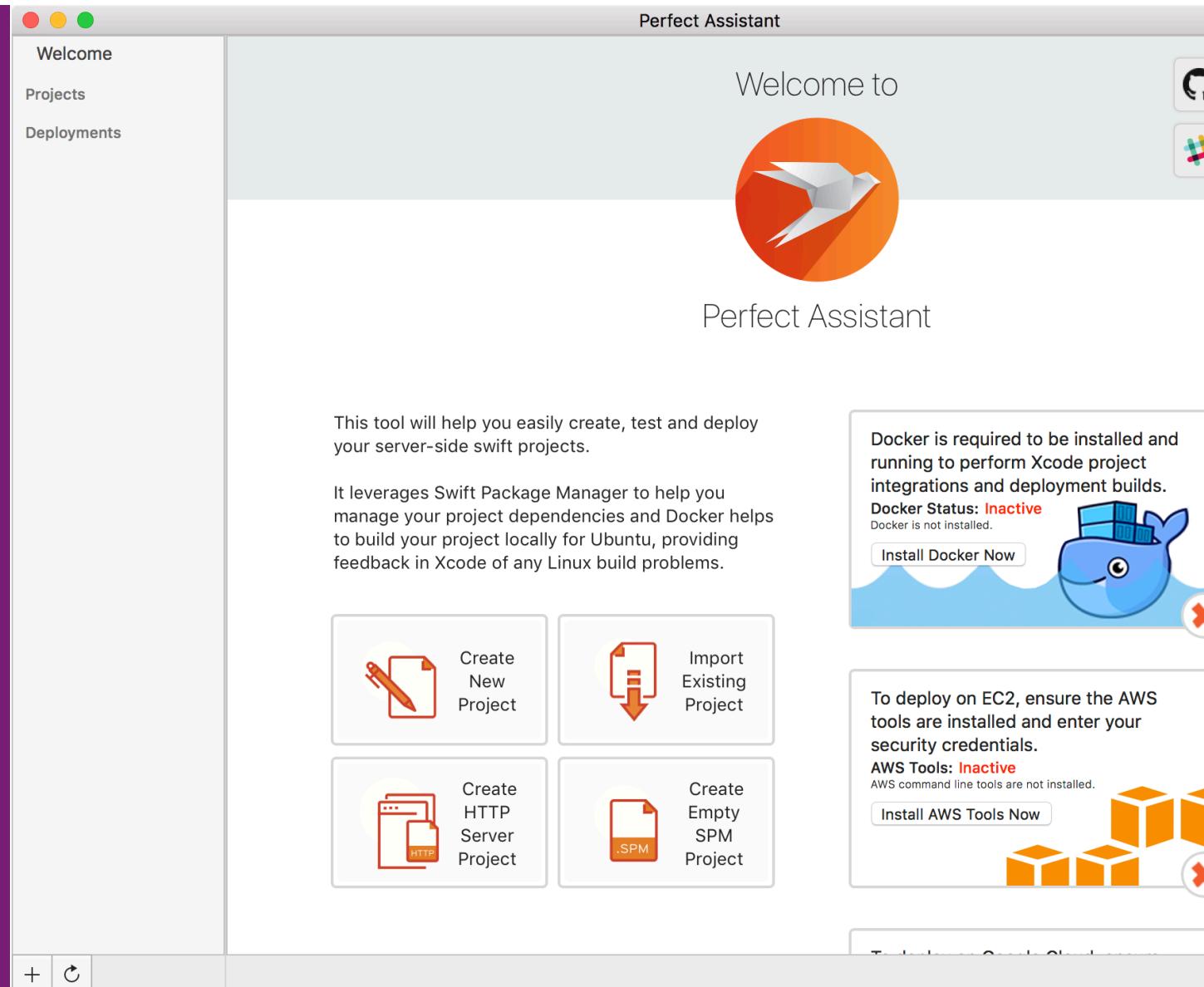
PERFECT ASSISTANT

- Download and install



PERFECT ASSISTANT

- Download and install



PERFECT ASSISTANT

- Deploy using Docker

This tool will help you easily create, test and deploy your server-side swift projects.

It leverages Swift Package Manager to help you manage your project dependencies and Docker helps to build your project locally for Ubuntu, providing feedback in Xcode of any Linux build problems.



Create
New
Project



Import
Existing
Project



Create
HTTP
Server
Project



Create
Empty
.SPM
Project

Perfect Assistant

Docker is required to be installed and running to perform Xcode project integrations and deployment builds.

Docker Status: Inactive
Docker is not installed.

[Install Docker Now](#)



To deploy on EC2, ensure the AWS tools are installed and enter your security credentials.

AWS Tools: Inactive
AWS command line tools are not installed.

[Install AWS Tools Now](#)



To deploy on Google Cloud, ensure the cloud tools are installed and enter your security credentials.

Cloud Tools: Active
Cloud command line tools are installed.

[Open Cloud Console...](#)



PERFECT ASSISTANT

Perfect Assistant

Welcome Projects Deployments

New Project

Project Deployment

Choose one of the starter project templates below.

Basic

- Perfect Template App
- Empty Executable Project
- Empty Library Project
- Custom Repository URL
- Perfect Template App Engine

Examples

- Perfect: Upload
- Perfect: URL
- Perfect:
- Perfect:
- Perfect:

What we just did.

PERFECT ASSISTANT

Perfect Assistant

New Project

Project Deployment

Welcome Projects Deployments

Choose one of the starter project templates below.

Basic

Perfect Template App

Empty Executable Project

Empty Library Project

Custom Repository URL

Perfect Template App Engine

Examples

Perfect: Upload

Perfect: URL

Perfect:

Perfect:

Perfect:

Create a project

The screenshot shows the 'New Project' screen of the Perfect Assistant application. On the left, there's a sidebar with 'Welcome', 'Projects', and 'Deployments'. The main area has tabs for 'Project' (selected) and 'Deployment'. It says 'Choose one of the starter project templates below.' Below this, under 'Basic', are five icons: 'Perfect Template App' (file with a paintbrush), 'Empty Executable Project' (file with gear), 'Empty Library Project' (two files), 'Custom Repository URL' (file with pencil), and 'Perfect Template App Engine' (file with paintbrush). Under 'Examples', there are five more icons: 'Perfect: Upload' (file with upload arrow), 'Perfect: URL' (file with URL), 'Perfect:' (file with list), 'Perfect:' (file with list), and 'Perfect:' (file with list). A large blue callout points from the 'Create a project' button at the bottom to the 'Empty Executable Project' icon.

PERFECT ASSISTANT

Perfect Assistant

Welcome Projects Deployments

New Project

Project Deployment

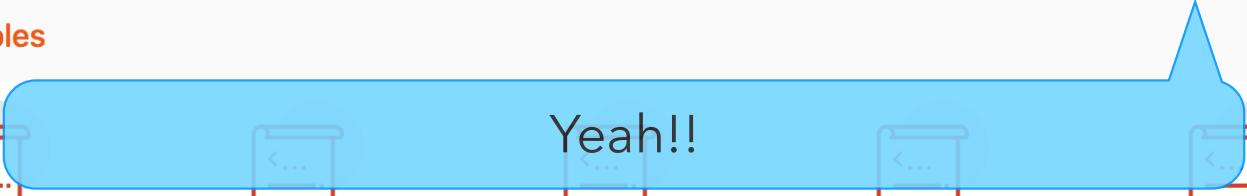
Choose one of the starter project templates below.

Basic

- Perfect Template App
- Empty Executable Project
- Empty Library Project
- Custom Repository URL
- Perfect Template App Engine

Examples

- Perfect: Upload
- Perfect: URL
- Perfect: Yeah!!
- Perfect:
- Perfect:



PERFECT ASSISTANT

 Perfect Template App	 Empty Executable Project	 Empty Library Project	 Custom Repository URL	 Perfect Template App Engine
Examples				
 Perfect: Upload Enumerator	 Perfect: URL Routing	 Perfect: WebSockets Server	 Perfect: Turnstile with SQLite	 Perfect: Turnstile with PostgreSQL
 Perfect: Weather	 Perfect: Polling	 Perfect: Yeah!!	 Perfect: Blog Mustache	 Perfect: JSON API

[Cancel](#) [Previous](#) [Next](#)

PERFECT ASSISTANT

Feedback in Xcode or any Linux build problems.

The screenshot shows the Perfect Assistant interface. At the top, there are two large buttons: "Create New Project" (with a pencil icon) and "Import Existing Project" (with a downward arrow icon). Below these are three smaller buttons: "Create HTTP Server Project" (with a folder icon labeled ".HTTP"), "Create SPM Project" (with a folder icon labeled ".SPM"), and "Empty SPM Project" (with a folder icon labeled ".SPM"). A blue callout bubble points to the "Create HTTP Server Project" button with the text "Create a new project". To the right of this callout, there is a note about deploying to EC2: "To deploy on EC2, ensure the AWS tools are installed and enter your security credentials." Below this note is a button labeled "Install AWS Tools Now" with an orange icon of three cubes. In the bottom left corner, there is a status message: "Creating /Users/tabinkowski/swift-server/perfect/perfect-template • ⚡". In the bottom right corner, there is a "Clear" button. The background features a cartoon character of a blue fish swimming in water.

Create New Project

Import Existing Project

Create a new project

To deploy on EC2, ensure the AWS tools are installed and enter your security credentials.

Install AWS Tools Now

Creating /Users/tabinkowski/swift-server/perfect/perfect-template • ⚡

Clear

Project saved.
Creating working directory and files
Backed up existing package file
Created package file

PERFECT ASSISTANT

Perfect Assistant

Welcome

Projects perfect-template

Deployments

project: perfect-template

location: /Users/tabinkowski/swift-server/perfect/perfect-template

Add dependencies for your project

OPEN

- Project Directory
- Project Terminal
- Xcode Project

BUILD

- Local
- Linux
- Deploy
- Clean

DEPENDENCIES

Selected Dependencies:

- HTTPServer version: 2.0.x
- HTTP version: 2.0.x
- PerfectLib version: 2.0.x

Available Dependencies:

- Turnstile-PostgreSQL
- Turnstile-MYSQL
- Turnstile-MongoDB
- Authentication
- Turnstile-CouchDB
- Turnstile-SQLite

Database Connector

The screenshot shows the Perfect Assistant application interface. On the left is a sidebar with tabs for Welcome, Projects (selected), Deployments, OPEN (with Project Directory, Project Terminal, Xcode Project), and BUILD (with Local, Linux, Deploy, Clean). The main area displays project details: 'project: perfect-template' and 'location: /Users/tabinkowski/swift-server/perfect/perfect-template'. A large blue callout bubble points to the 'DEPENDENCIES' section with the text 'Add dependencies for your project'. The 'DEPENDENCIES' section shows 'Selected Dependencies' for HTTPServer, HTTP, and PerfectLib at version 2.0.x. Below this is a 'Available Dependencies' section listing Turnstile-PostgreSQL, Turnstile-MYSQL, Turnstile-MongoDB, Authentication, Turnstile-CouchDB, and Turnstile-SQLite. A 'Database Connector' section is partially visible at the bottom.

PERFECT ASSISTANT

The screenshot shows the Perfect Assistant application interface. On the left, there are two vertical panels: 'BUILD' and 'RUN'. The 'BUILD' panel contains icons for Local, Linux, Deploy, and Clean. The 'RUN' panel contains icons for Local Tests, Linux Tests, Local Exe, and Linux Exe. At the top, there are five dropdown menus labeled 'HTTPServer', 'HTTP', 'PerfectLIB', 'PostgreSQL', and 'Postgres-STORMI', each with a version selector (e.g., 'version: 2.0.x'). A large blue callout box is positioned over the center of the interface, containing the text 'Available Dependencies:' and a list of components:

- MySQL-StORM
- CouchDB-StORM
- SQLite-StORM
- Postgres-StORM
- MongoDB-StORM

Below this, under 'Server', are:

- WebRedirects
- HTTPServer
- WebSockets
- HTTP
- FastCGI

Under 'Session Management' are:

- Session-CouchDB
- Session-SQLite
- Session-MYSQL
- Session-PostgreSQL
- Session-MongoDB
- Session

PERFECT ASSISTANT



Local

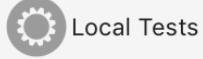


Linux

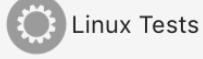


Deploy
Clean

RUN



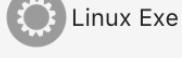
Local Tests



Linux Tests



Local Exe



Linux Exe

XCODE PROJECT



Regenerate

Available Dependencies:

▼ Authentication



Turnstile-
PostgreSQL



Turnstile-
MySQL



Turnstile-
MongoDB



Authenticati-
on



Turnstile-
CouchDB



Turnstile-
SQLite

▼ Database Connector



PostgreSQL



MySQL



CouchDB



FileMaker



Redis



SQLite



MongoDB

▼ ORM



MySQL-
StORM



CouchDB-
StORM



SQLite-
StORM



Postgres-
StORM



MongoDB-
StORM

▼ Server

<https://github.com/SwiftORM/CouchDB-StORM.git>

PERFECT ASSISTANT

Perfect Assistant

project: perfect-template Postgresql setup for swift server
location: /Users/tabinkowski/swift-server/perfect/perfect-template

OPEN

- Project Directory
- Project Terminal
- Xcode Project

BUILD

- Local
- Linux
- Deploy
- Clean

DEPENDENCIES

Selected Dependencies:

- HTTPServer version: 2.0.x
- HTTP version: 2.0.x
- PerfectLib version: 2.0.x
- PostgreSQL version: 2.x.x
- Postgres-StORM version: 1.x.x

Available Dependencies:

▼ Authentication

- Turnstile-PostgreSQL
- Turnstile-MYSQL
- Turnstile-MongoDB
- Authentication
- Turnstile-CouchDB
- Turnstile-SQLite

PERFECT ASSISTANT

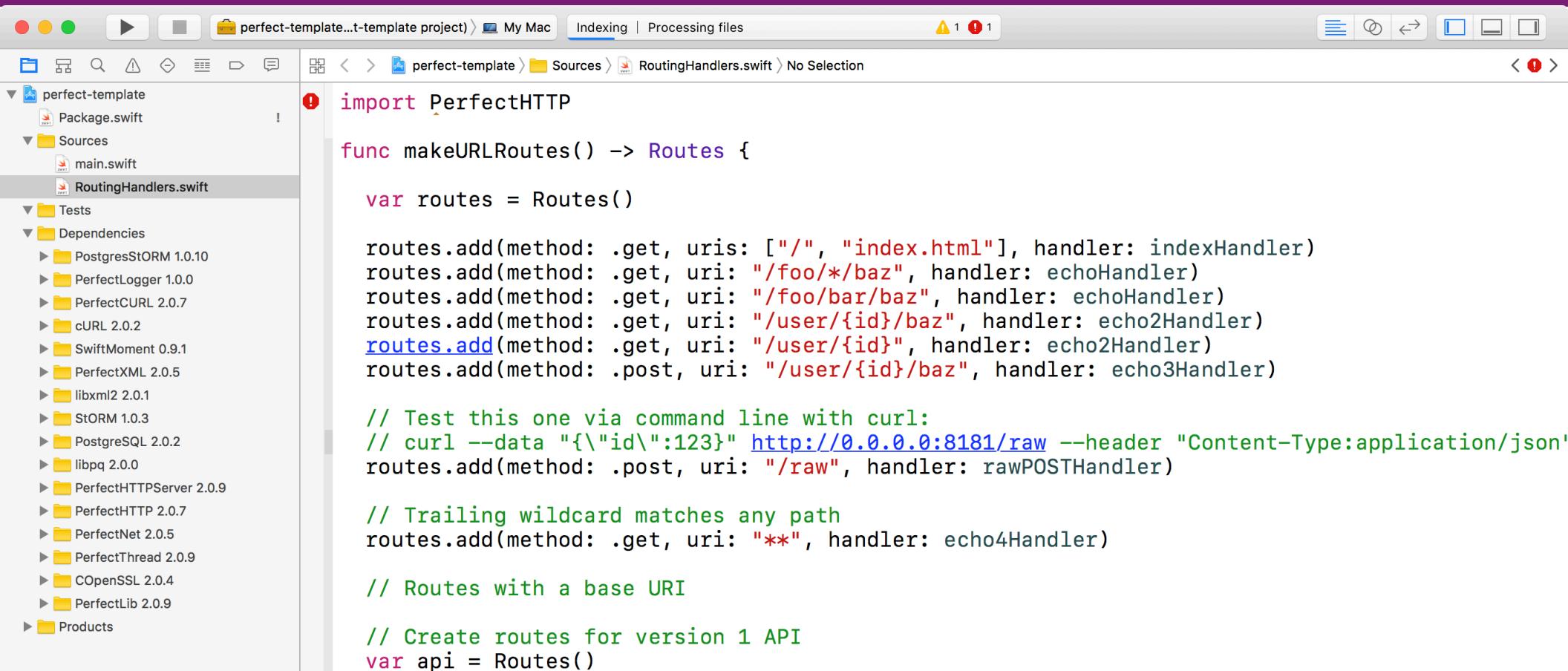
The screenshot shows the Perfect Assistant interface with a dark purple header and a light gray main area. At the top, there are icons for different session drivers: Session-touchDB (gear), Session-SQLite (blue feather icon), Session-MySQL (MySQL logo), Session-PostgreSQL (PG icon), Session-MongoDB (green leaf icon), and Session (red gear icon). Below this, a section titled "Utility" contains icons for questLogger (red gear), Logger (red gear), Notifications (red gear), OpenSSL (red gear), Logger (red gear), Repeater (red gear), XML (red gear), and PerfectLib (red gear). Further down, there are icons for Net (red gear), Zip (red gear), SMTP (red gear), Thread (red gear), Mustache (red gear), and CURL (red gear). A blue callout bubble points to the "Save and update the project" button, which is located at the bottom of the utility section. The entire interface has a clean, modern design with a focus on developer tools.

Add Dependency...

Automatically integrate Xcode
when regenerating project

Save Changes

PERFECT ASSISTANT



The screenshot shows a Mac OS X desktop with an Xcode interface. The title bar reads "perfect-template...t-template project > My Mac". The status bar shows "Indexing | Processing files" and "1 1 1". The main window displays a Swift file named "RoutingHandlers.swift" under the "Sources" folder of the "perfect-template" project. The code implements a routing system using the PerfectHTTP framework.

```
! import PerfectHTTP

func makeURLRoutes() -> Routes {
    var routes = Routes()

    routes.add(method: .get, uris: ["/", "index.html"], handler: indexHandler)
    routes.add(method: .get, uri: "/foo/*baz", handler: echoHandler)
    routes.add(method: .get, uri: "/foo/bar/baz", handler: echoHandler)
    routes.add(method: .get, uri: "/user/{id}/baz", handler: echo2Handler)
    routes.add(method: .get, uri: "/user/{id}", handler: echo2Handler)
    routes.add(method: .post, uri: "/user/{id}/baz", handler: echo3Handler)

    // Test this one via command line with curl:
    // curl --data "{\"id\":123}" http://0.0.0.0:8181/raw --header "Content-Type:application/json"
    routes.add(method: .post, uri: "/raw", handler: rawPOSTHandler)

    // Trailing wildcard matches any path
    routes.add(method: .get, uri: "**", handler: echo4Handler)

    // Routes with a base URI

    // Create routes for version 1 API
    var api = Routes()
```

APP ENGINE

PERFECT ASSISTANT

Perfect Assistant

A simple server listening on port 8080. Ready for Google App Engine deployment. Includes health check handler.

Choose a location for the project directory and enter a name for your new SPM package.

Project Location: [Browse](#)

ProjectName: [Filter](#)

Integrate Linux builds with Xcode project

Selecting this option will lengthen Xcode build times but will provide feedback in Xcode of Linux related compilation issues. You can add or remove Xcode project integration at any time in the project editor view.

Cancel [Create](#)

RL Enumerator WebSockets Server Perfect: URL Shortener Perfect: JSON API Perfect: Blog Mustache

Projects [PerfectAppEngine](#)
Deployments

Get Help Slack Visit Us GitHub

PERFECT ASSISTANT

/Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-code-samples/ [Locate](#)

Linux Swift Toolchain Version 4.0 

Automatically Integrate Xcode Project

Dev Ports Deploy Ports

Project Dependencies

 **HTTPServer**
version: 2.x.x 

Available Dependencies 

- ▶ No Category
- ▶ Session Management
- ▶ Database Connector
- ▶ Authentication
- ▶ Utility
- ▶ Server
- ▶ ODM

Project Deployments 

New Google Cloud: 
App Engine

Not configured.



PERFECT ASSISTANT

≡ Google Cloud Platform mpcs51033-2017-autumn... 🔍

DASHBOARD ACTIVITY CUSTOMIZE

Project info

Project name
mpcs51033-2017-autumn-photos

Project ID
mpcs51033-2017-autumn-photos

Project number
699588220284

[Go to project settings](#)

Resources

- App Engine 2 versions
- Compute Engine

App Engine

Summary (count/sec)

7:30 7:45 8 PM 8:15

- http/server/response_count:
- http/server/response_count:

[Go to the App Engine dashboard](#)

Google Cloud Platform status

All services normal

[Go to Cloud status dashboard](#)

Billing

Estimated charges \$0.00

For the billing period Nov 1 – 20, 2017

[View detailed charges](#)

Error Reporting

PERFECT ASSISTANT

- Docker file

```
FROM ibmcom/swift-ubuntu:latest
LABEL Description="Docker image for Swift + Perfect on Google App Engine"

# Get extra dependencies for Perfect
RUN apt-get update && apt-get install -y \
    openssl \
    libssl-dev \
    uuid-dev

# Expose default port for App Engine
EXPOSE 8080

# Copy sources
RUN mkdir /root/perfectcmdline
ADD ./Sources/perfectcmdline/main.swift /root/perfectcmdline/
ADD Package.swift /root/perfectcmdline

# Build the app
RUN cd /root/perfectcmdline && swift build

# Run the app
USER root
CMD ["/root/perfectcmdline/.build/debug/perfectcmdline"]
```

PERFECT ASSISTANT

```
runtime: custom
env: flex
```

- app.yaml

PERFECT ASSISTANT

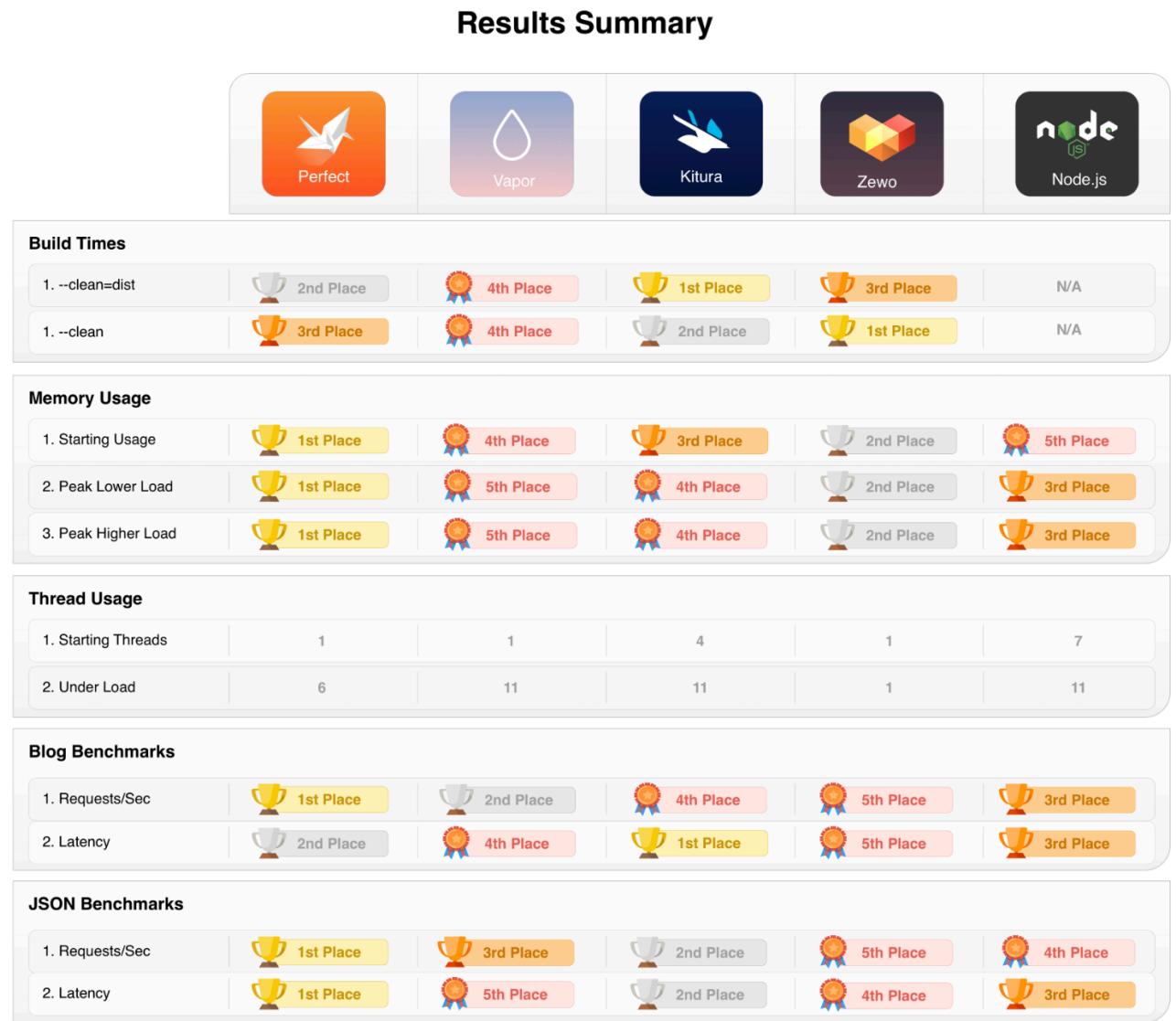
```
gcloud app deploy --project  
animated-beacon-186702
```

- Deploy

THE FUTURE OF SWIFT ON THE SERVER

PERFORMANCE

- <https://medium.com/@rymcol/benchmarks-for-the-top-server-side-swift-frameworks-vs-node-js-24460cfe0beb>



FUTURE OF SWIFT ON THE SERVER

- Great interest, tools, support
- Many projects that are highly active (and with industry support)
 - Apple
 - IBM
 - Prefect

PRODUCTION READY

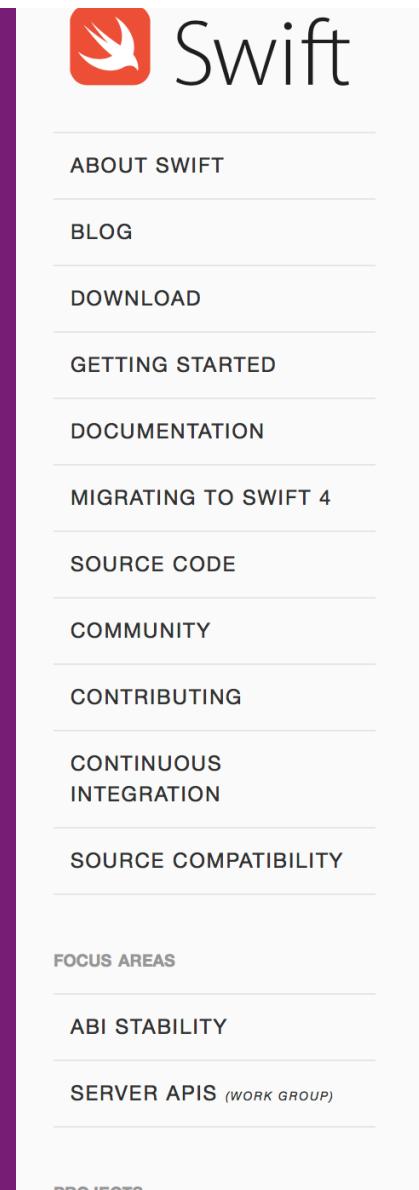
- Is it ready for prime time? Probably not yet.
- Missing frameworks
 - No URLSession
- The frameworks will make the shared code promise hard to keep
 - Custom models still need to be converted

SWIFT PACKAGE MANAGER

FOR REFERENCE

SWIFT PACKAGE MANAGER

- The Swift Package Manager is a tool for managing the distribution of Swift code
- Officially part of Swift



Package Manager

The Swift Package Manager is a tool for managing the distribution of Swift code. It's integrated with the Swift build system to automate the process of downloading, compiling, and linking dependencies.

The Package Manager is included in Swift 3.0 and above.

Conceptual Overview

This section describes the basic concepts that motivate the functionality of the Swift Package Manager.

Modules

Swift organizes code into *modules*. Each module specifies a namespace and enforces access controls on which parts of that code can be used outside of the module.

A program may have all of its code in a single module, or it may import

SWIFT PACKAGE MANAGER

- Consistent and convenient
 - Share and reuse code
 - Build libraries and executables
 - Manage dependencies

Swift

SWIFT
OAD
G STARTED
IENATION
TING TO SWIFT 4
E CODE
JUNITY
BIBUTING
LIUOUS ATION
E COMPATIBILITY
AS
ABILITY
R APIs (WORK GROUP)
ED AND

Package Manager

The Swift Package Manager is a tool for managing the distribution of Swift code. It's integrated with the Swift build system to automatically handle the process of downloading, compiling, and linking dependencies.

The Package Manager is included in Swift 3.0 and above.

Conceptual Overview

This section describes the basic concepts that motivate the functionality of the Swift Package Manager.

Modules

Swift organizes code into *modules*. Each module specifies a namespace and enforces access controls on which parts of that namespace can be used outside of the module.

A program may have all of its code in a single module, or it may depend on other modules as *dependencies*. Aside from the handful of system

SWIFT PACKAGE MANAGER

- Modules
 - A group of related code
 - Share namespace and access controls
 - All of our current code has been in a single module

Swift

SWIFT
OAD
G STARTED
IENATION
TING TO SWIFT 4
E CODE
JUNITY
BUTURING
UOUS
ATION
E COMPATIBILITY
AS
ABILITY
R APIs (WORK GROUP)
ED AND

Package Manager

The Swift Package Manager is a tool for managing the distribution of Swift code. It's integrated with the Swift build system to automate the process of downloading, compiling, and linking dependencies.

The Package Manager is included in Swift 3.0 and above.

Conceptual Overview

This section describes the basic concepts that motivate the functionality of the Swift Package Manager.

Modules

Swift organizes code into *modules*. Each module specifies a namespace and enforces access controls on which parts of that module can be used outside of the module.

A program may have all of its code in a single module, or it may depend on other modules as *dependencies*. Aside from the handful of system

SWIFT PACKAGE MANAGER

- Packages
 - Contains source code that will be compile to a module/library
 - Contains a manifest file `Package.swift` to define it

Swift

SWIFT
OAD
G STARTED
IENТАTION
TING TO SWIFT 4
E CODE
JUNITY
BUTURING
UOUS ATION
E COMPATIBILITY
AS
ABILITY
R APIs (WORK GROUP)
ED AND

Package Manager

The Swift Package Manager is a tool for managing the distribution of Swift code. It's integrated with the Swift build system to automatically handle the process of downloading, compiling, and linking dependencies.

The Package Manager is included in Swift 3.0 and above.

Conceptual Overview

This section describes the basic concepts that motivate the functionality of the Swift Package Manager.

Modules

Swift organizes code into *modules*. Each module specifies a namespace and enforces access controls on which parts of that module can be used outside of the module.

A program may have all of its code in a single module, or it may depend on other modules as *dependencies*. Aside from the handful of system

SWIFT PACKAGE MANAGER

- Executable
 - Package designed to run as a program
 - Contains `main.swift` file

Swift

SWIFT
OAD
G STARTED
IENATION
TING TO SWIFT 4
E CODE
JUNITY
BIBUTING
LIUOUS
ATION
E COMPATIBILITY
AS
ABILITY
R APIs (WORK GROUP)
ED AND

Package Manager

The Swift Package Manager is a tool for managing the distribution of Swift code. It's integrated with the Swift build system to automatically handle the process of downloading, compiling, and linking dependencies.

The Package Manager is included in Swift 3.0 and above.

Conceptual Overview

This section describes the basic concepts that motivate the functionality of the Swift Package Manager.

Modules

Swift organizes code into *modules*. Each module specifies a namespace and enforces access controls on which parts of that module can be used outside of the module.

A program may have all of its code in a single module, or it may depend on other modules as *dependencies*. Aside from the handful of system

SWIFT PACKAGE MANAGER

- Products
 - Target of a package or executable

Swift

SWIFT
OAD
G STARTED
IENATION
TING TO SWIFT 4
E CODE
JUNITY
IBUTING
IUOUS ATION
E COMPATIBILITY
AS
ABILITY
R APIs (WORK GROUP)
ED AND

Package Manager

The Swift Package Manager is a tool for managing the distribution of Swift code. It's integrated with the Swift build system to automate the process of downloading, compiling, and linking dependencies.

The Package Manager is included in Swift 3.0 and above.

Conceptual Overview

This section describes the basic concepts that motivate the functionality of the Swift Package Manager.

Modules

Swift organizes code into *modules*. Each module specifies a namespace and enforces access controls on which parts of that module can be used outside of the module.

A program may have all of its code in a single module, or it may depend on other modules as *dependencies*. Aside from the handful of system

SWIFT PACKAGE MANAGER

- Dependencies
 - Modules that are required for code in the package
 - Not all programs will have dependencies
 - Package manager coordinates all of the dependencies of dependencies of dependencies...

Package Manager

The Swift Package Manager is a tool for managing Swift code. It's integrated with the Swift build system process of downloading, compiling, and linking dependencies.

The Package Manager is included in Swift 3.0 and later.

Conceptual Overview

This section describes the basic concepts that make up the functionality of the Swift Package Manager.

Modules

Swift organizes code into *modules*. Each module defines a namespace and enforces access controls on which code can be used outside of the module.

A program may have all of its code in a single module or it may depend on other modules as *dependencies*. Aside from the

SWIFT PACKAGE MANAGER



SWIFT PACKAGE MANAGER

PACKAGE

MODULE



SWIFT PACKAGE MANAGER

PACKAGE

MODULE

MODULE



SWIFT PACKAGE MANAGER

PACKAGE

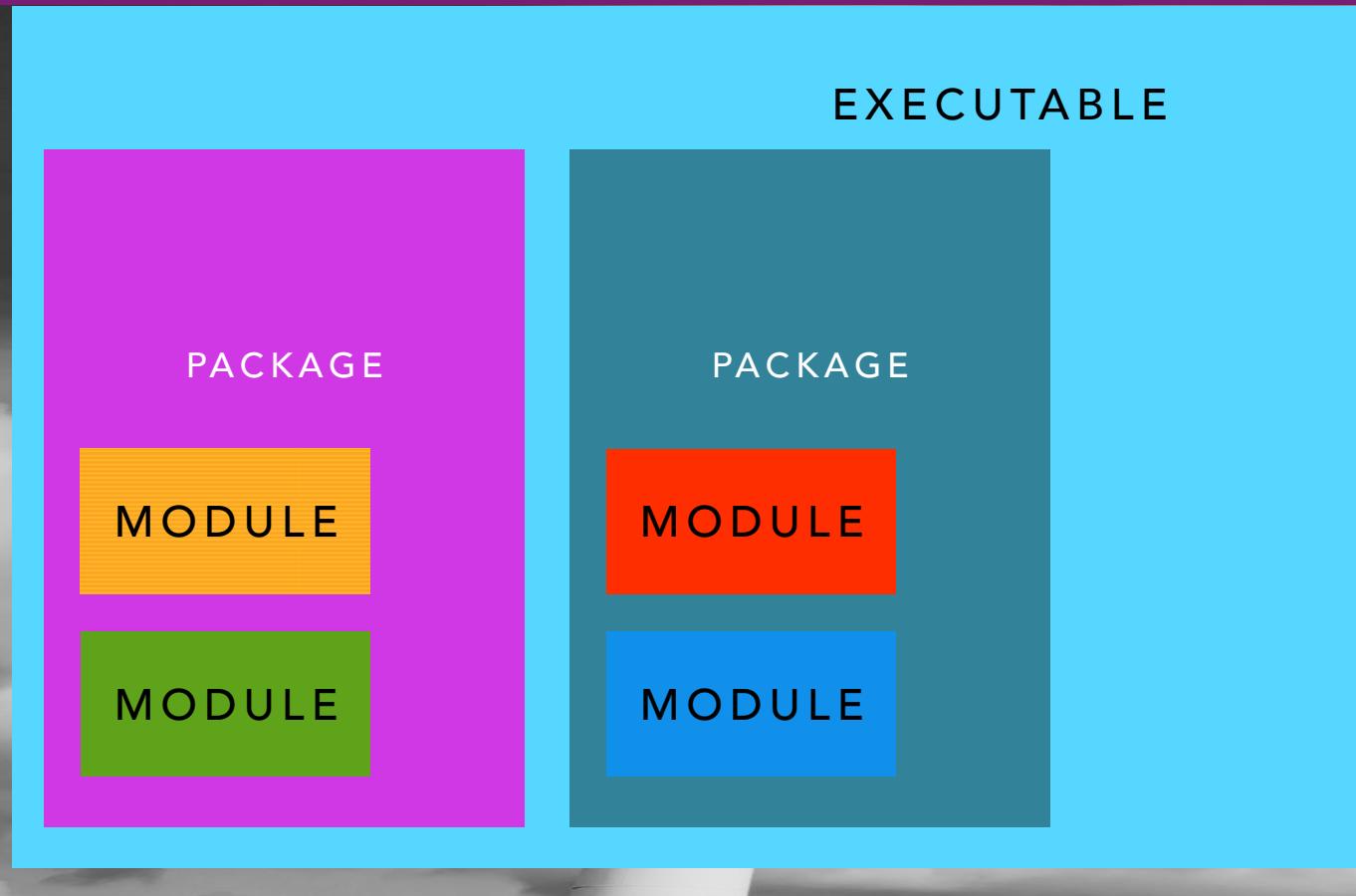
MODULE

MODULE

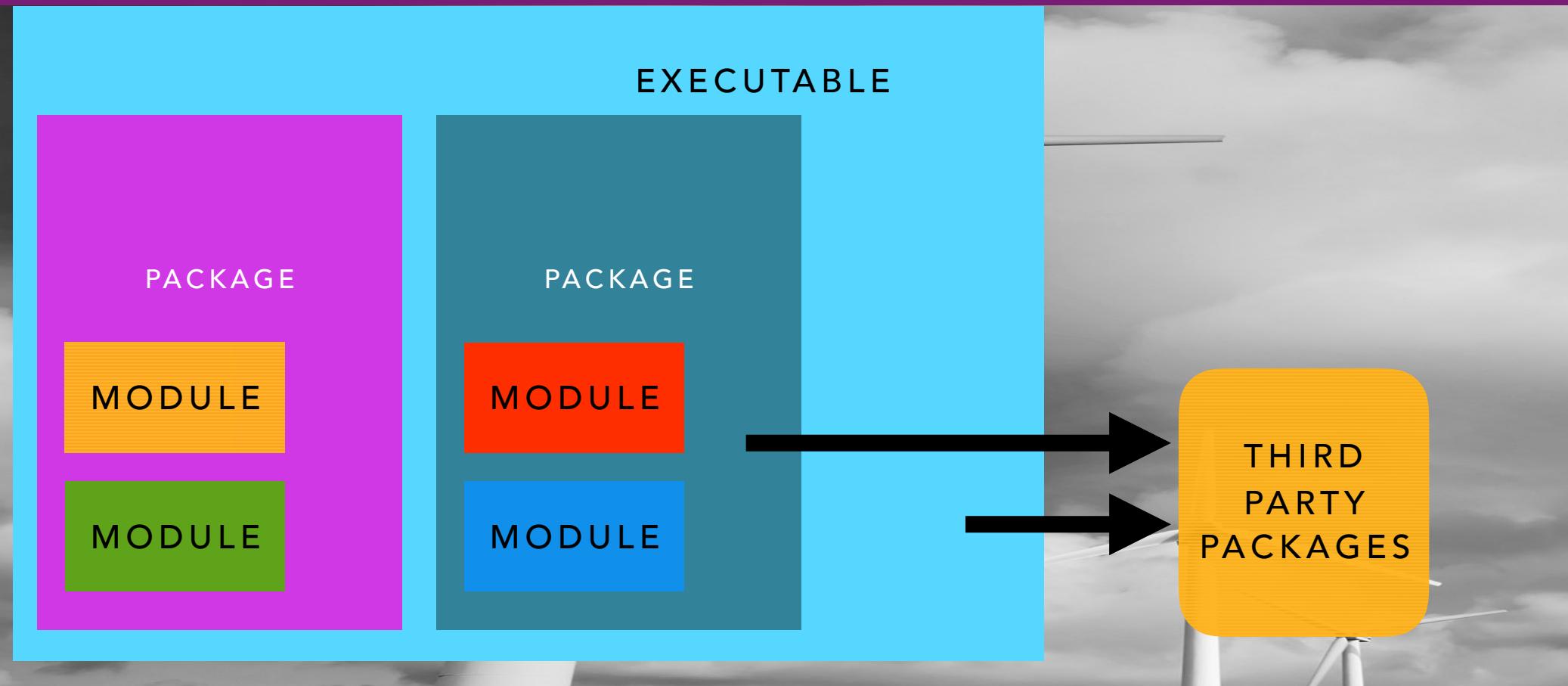
EXECUTABLE



SWIFT PACKAGE MANAGER



SWIFT PACKAGE MANAGER



**CREATE A
PACKAGE**

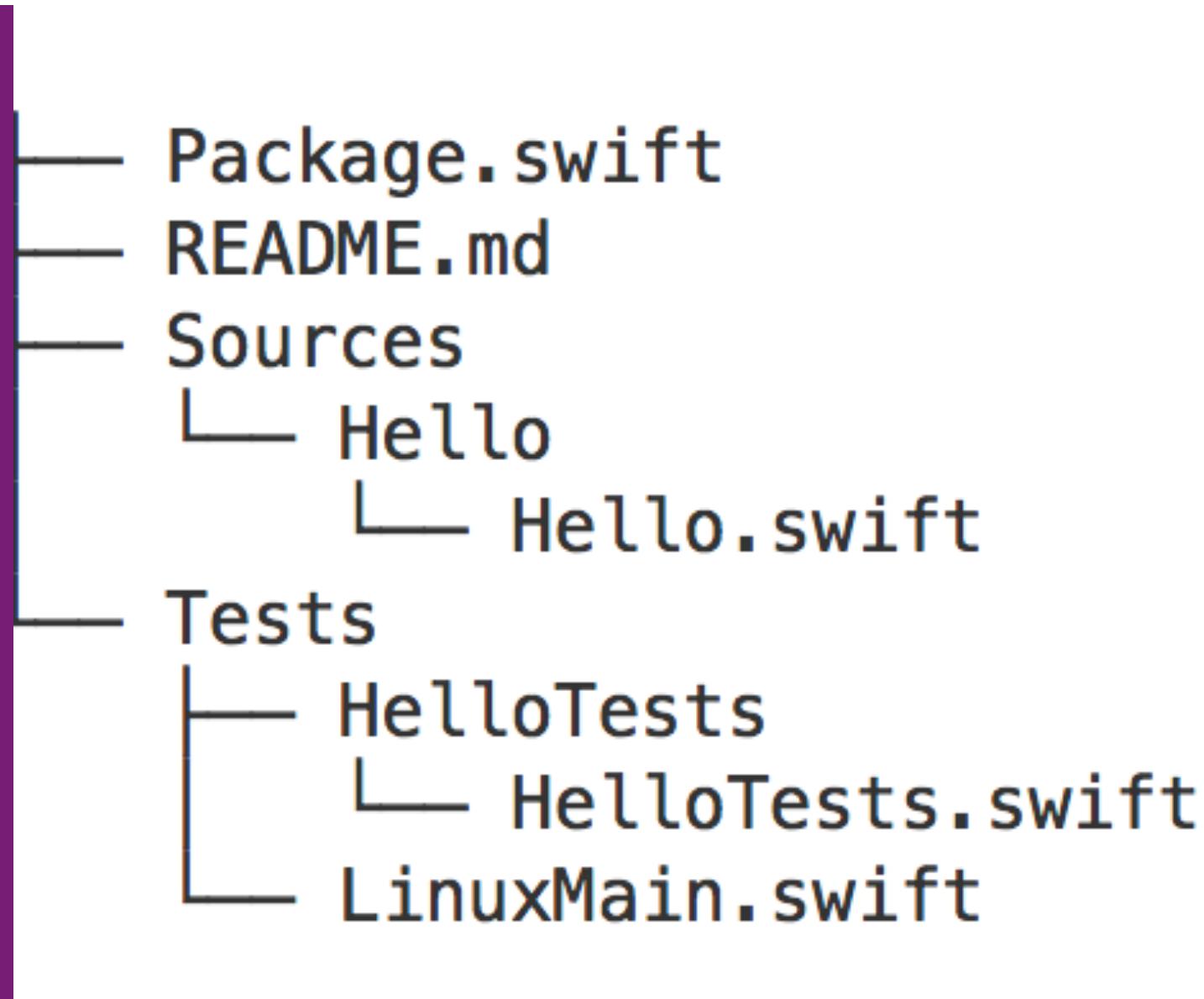
CREATE A PACKAGE

- Create a directory
- `cd` to directory
- Run `swift package init`

```
mkdir Hello  
cd Hello
```

CREATE A PACKAGE

- Creates a directory/file structure
- Name of folder is name of package



CREATE A PACKAGE

- Creates a directory/file
- IGNORE TESTS FOR NOW
- Name of folder is name of package

```
Package.swift
README.md
Sources
└ Hello
    └ Hello.swift
Tests
└ HelloTests
    └ HelloTests.swift
LinuxMain.swift
```

CREATE A PACKAGE

The screenshot shows a file browser window in Xcode. On the left, the file tree is visible:

- session-5 (selected)
- ↳ Hello
 - ↳ Sources
 - ↳ Hello
 - ↳ Hello.swift (selected)
 - ↳ Tests
 - ↳ Package.swift
 - ↳ README.md

On the right, the code editor displays the contents of `Hello.swift`:

```
1 struct Hello {  
2     var text = "Hello, World!"  
3 }  
4
```

A yellow callout bubble with a black border and a black arrow points from the text "Source code for your package" towards the `Hello.swift` file in the file tree.

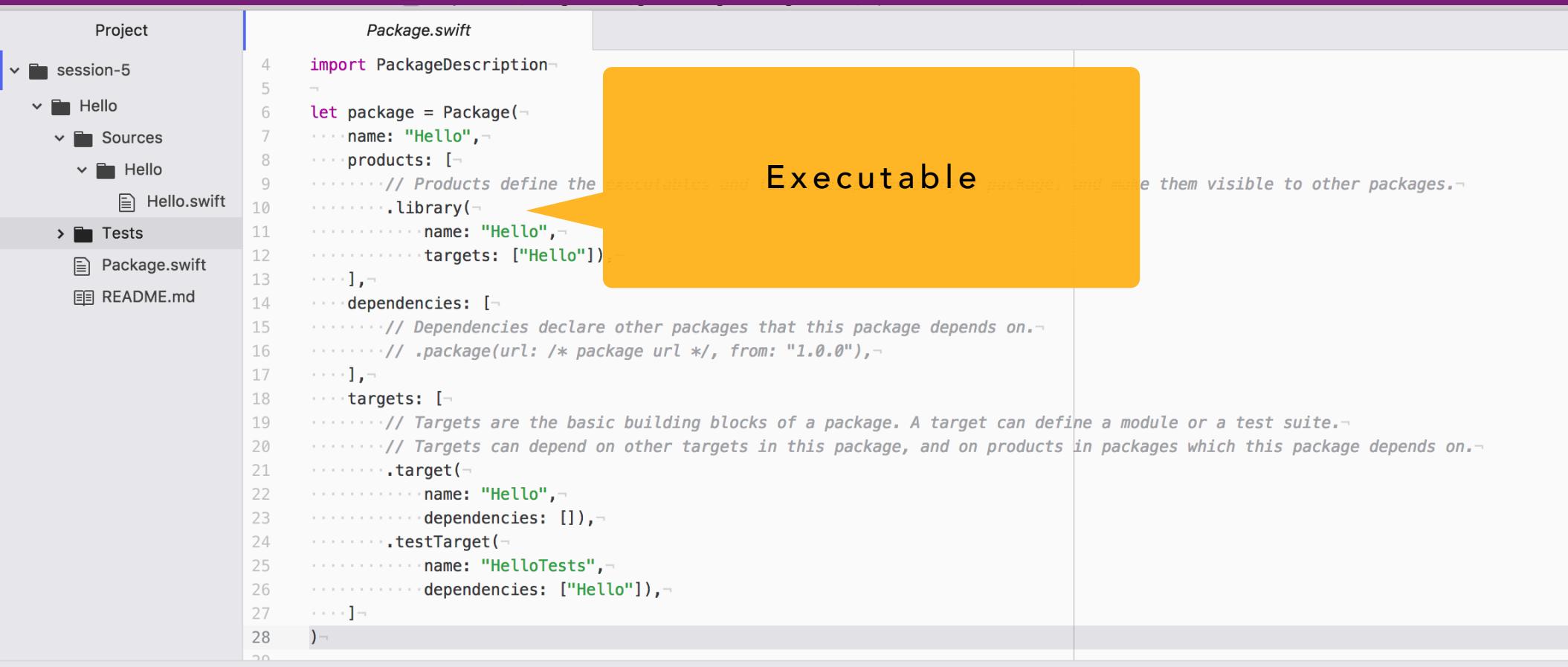
Source code
for your
package

CREATE A PACKAGE

Manifest file
describing the package

```
Project          Package.swift
4 import PackageDescription
5
6 let package = Package(
7   name: "Hello",
8   products: [
9     // Products define the executables and libraries produced by a package, and make them visible to other packages.
10    .library(
11      name: "Hello",
12      targets: ["Hello"]),
13    ],
14   dependencies: [
15     // Dependencies declare other packages that this package depends on.
16     // .package(url: /* package url */, from: "1.0.0"),
17   ],
18   targets: [
19     // Targets are the basic building blocks of a package. A target can define a module or a test suite.
20     // Targets can depend on other targets in this package, and on products in packages which this package depends on.
21     .target(
22       name: "Hello",
23       dependencies: []),
24     .testTarget(
25       name: "HelloTests",
26       dependencies: ["Hello"]),
27   ]
28 )
```

CREATE A PACKAGE



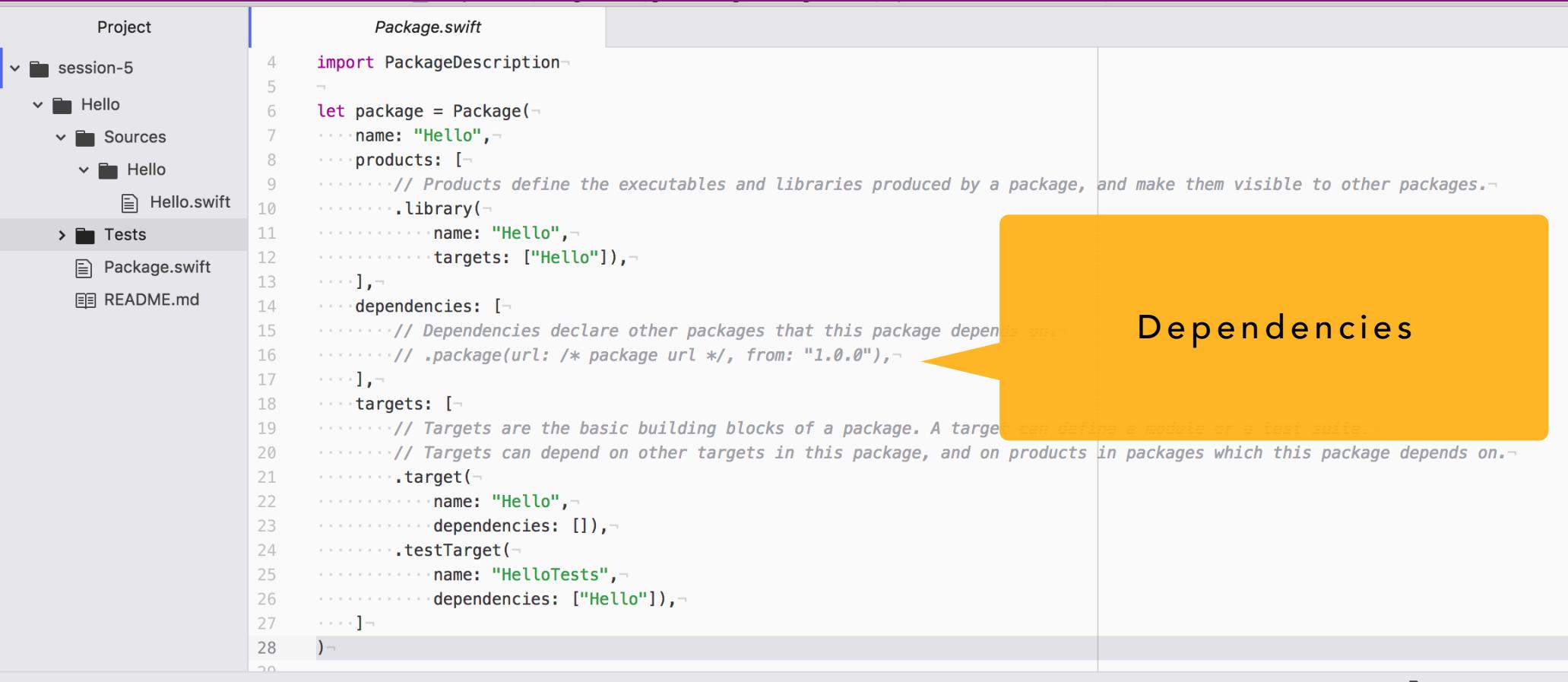
The screenshot shows the Xcode interface with a project named "session-5". The left sidebar shows files like "Hello.swift", "Tests", "Package.swift", and "README.md". The main editor window displays the "Package.swift" file content:

```
Project          Package.swift

4 import PackageDescription
5
6 let package = Package(
7   name: "Hello",
8   products: [
9     // Products define the executables and frameworks and their
10    // dependencies and resources of the package. Make
11    // them visible to other packages.
12    .library(
13      name: "Hello",
14      targets: ["Hello"]),
15  ],
16  dependencies: [
17    // Dependencies declare other packages that this package depends on.
18    // .package(url: /* package url */, from: "1.0.0"),
19  ],
20  targets: [
21    // Targets are the basic building blocks of a package. A target can define a module or a test suite.
22    // Targets can depend on other targets in this package, and on products in packages which this package depends on.
23    .target(
24      name: "Hello",
25      dependencies: []),
26      .testTarget(
27        name: "HelloTests",
28        dependencies: ["Hello"]),
29    ]
30 )
```

A yellow callout bubble points to the word "Executable" in the code, which is part of the documentation for the `.library` product type.

CREATE A PACKAGE

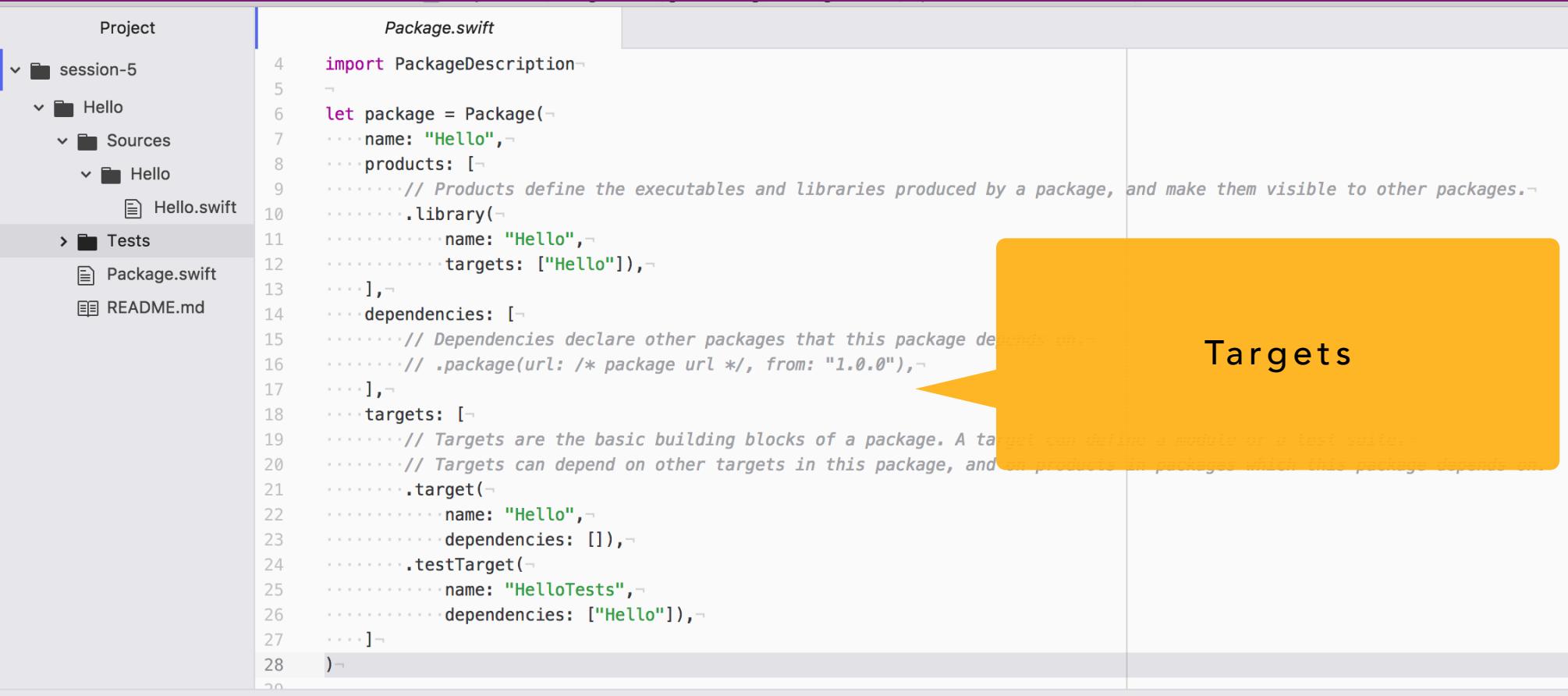


Project Package.swift

```
4 import PackageDescription
5
6 let package = Package(
7   name: "Hello",
8   products: [
9     .library(
10       name: "Hello",
11       targets: ["Hello"]),
12     ],
13     dependencies: [
14       .package(url: /* package url */, from: "1.0.0"),
15     ],
16     targets: [
17       .target(
18         name: "Hello",
19         dependencies: []),
20       .testTarget(
21         name: "HelloTests",
22         dependencies: ["Hello"]),
23     ]
24   )
25
26
27
28 )
```

Dependencies

CREATE A PACKAGE



Project Package.swift

```
4 import PackageDescription
5
6 let package = Package(
7   name: "Hello",
8   products: [
9     .library(
10       name: "Hello",
11       targets: ["Hello"]),
12     ],
13   dependencies: [
14     .package(url: /* package url */, from: "1.0.0"),
15   ],
16   targets: [
17     .target(
18       name: "Hello",
19       dependencies: []),
20     .testTarget(
21       name: "HelloTests",
22       dependencies: ["Hello"]),
23   ]
24 )
25
26
27
28 )
```

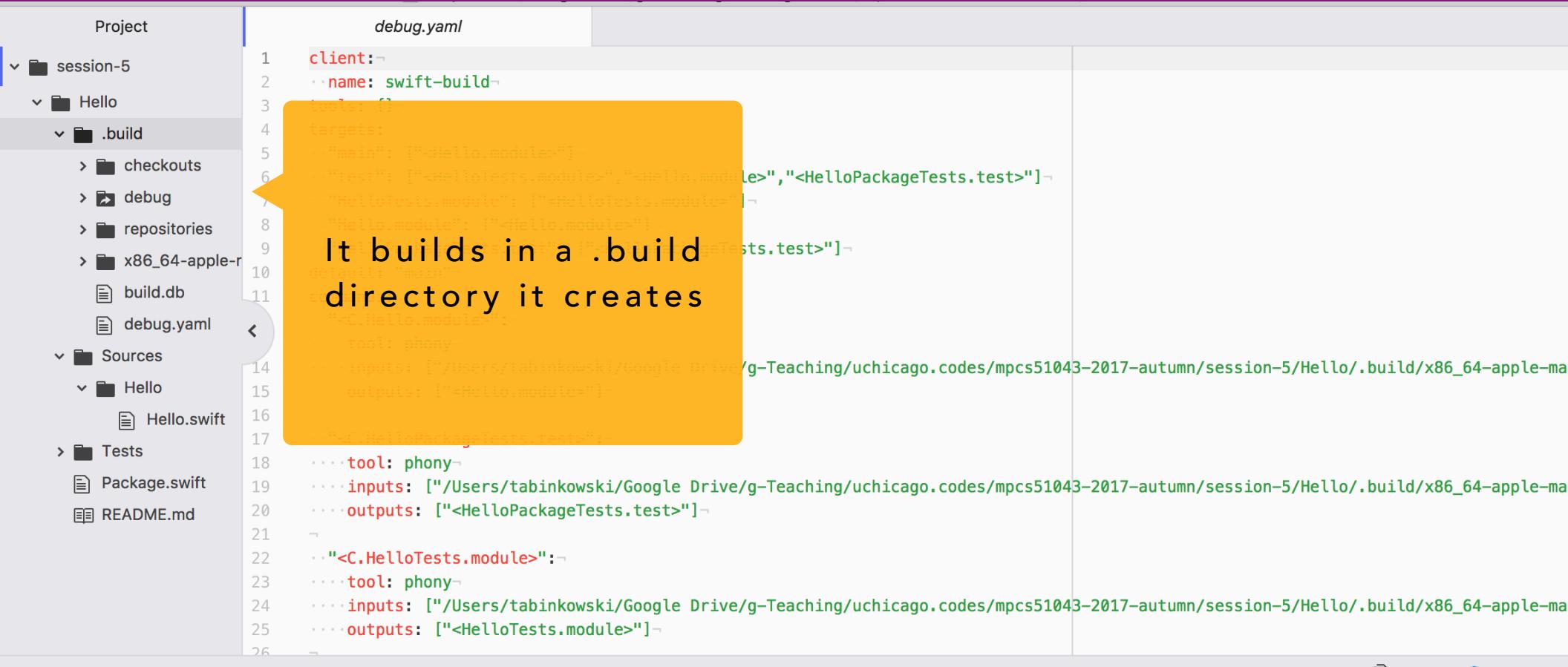
Targets

CREATE A PACKAGE

- Build a package
- Download,
resolve and
compile
dependencies

dule 'He

CREATE A PACKAGE



Project *debug.yaml*

```
1 client:-
2   ..name: swift-build
3   ..targets: []
4   ..targets:
5     ..main: ["<Hello.module>"]
6     ..test: ["<HelloTests.module>","<Hello.module>","<HelloPackageTests.test>"]
7     ..<HelloTests.module>: ["<HelloTests.module>"]
8     ..<Hello.module>: ["<Hello.module>"]
9     ..<HelloPackageTests.test>: ["<HelloPackageTests.test>"]
10    ..outputs: ["<Hello.module>"]
11    ..inputs: []
12    ..outputs: []
13    ..inputs: []
14    ..outputs: []
15    ..inputs: []
16    ..outputs: []
17    ..<HelloPackageTests.test>: []
18    ..tool: phony
19    ..inputs: []
20    ..outputs: []
21    ..<C.HelloTests.module>: []
22    ..tool: phony
23    ..inputs: []
24    ..outputs: []
25    ..inputs: []
26    ..outputs: []
```

It builds in a `.build` directory it creates

CREATE A PACKAGE

- Packages need to be under git control
- Package management will match the tag

```
% git init  
% git add .  
% git commit -m "Initial "  
% git tag 1.0.0
```

CREATE A PACKAGE

- We have a package containing code modules

-

Now what?



**BUILD AN
EXECUTABLE**

BUILD AN EXECUTABLE

- Same structure as package except contains a main.swift file
- Entry point to run program

Package.swift

README.md

Sources

— GoodDayToYou

 └ greetings.swift

 └ main.swift

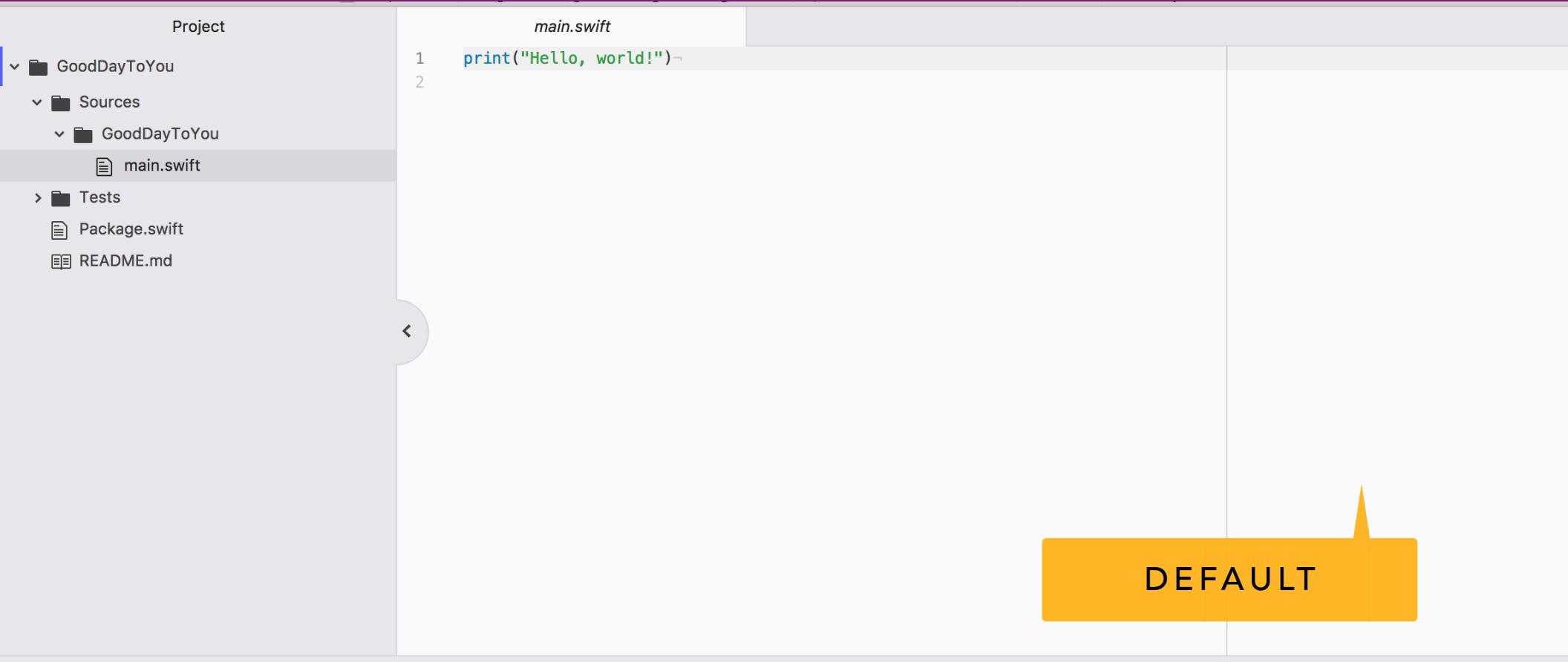
Tests

BUILD AN EXECUTABLE

- Create an executable package

```
% mkdir GoodDayToYou  
% cd Hello  
% swift package init --type executable
```

BUILD AN EXECUTABLE



The screenshot shows the Xcode interface with a project named "GoodDayToYou". The Project Navigator on the left lists the project structure:

- GoodDayToYou (selected)
- Sources
 - GoodDayToYou
 - main.swift
- Tests
- Package.swift
- README.md

The main editor window shows the file "main.swift" with the following content:

```
1 print("Hello, world!")
```

A yellow callout bubble with the word "DEFAULT" points to the "main.swift" file in the Project Navigator.

BUILD AN EXECUTABLE

- Create an executable package

```
# Build it
% swift build
Compile Swift Module 'GoodDayToYou' (1 sources)
Linking ./build/x86_64-apple-macosx10.10/
debug/GoodDayToYou
```

```
# Run it
% swift run
% swift run GoodDayToYou
```

BUILD AN EXECUTABLE

- Runs from .build directory
 - Copy it out to use it

```
# Build it  
% swift build
```

```
Compile Swift Module 'GoodDayToYou' (1 sources)  
Linking ./._.build/x86_64-apple-macosx10.10/  
debug/GoodDayToYou
```

```
# Run it  
% swift run GoodDayToYou
```

This is the location of the executable

CREATE A PACKAGE

```
[580 % ./build/x86_64-apple-macosx10.10/debug/GoodDayToYou
Hello, world!
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2017-autumn/session-5/GoodDayToYou
581 % ]
```

CREATE A PACKAGE

The screenshot shows the Xcode interface with a project named "GoodDayToYou". The Project Navigator on the left lists files and folders: .build, Sources (containing GoodDayToYou with greetings.swift and main.swift), Tests, Package.swift, and README.md. The "greetings.swift" file is open in the editor, showing the following code:

```
1 // Send a customized greeting
2 /// - parameter name: A `String` representing the name
3 func greet(name: String) {
4     print("Good day to you \(name) 😊")
5 }
```

A yellow callout bubble with a black border and a white background is positioned at the bottom right of the editor area. It contains the text: "Add another file to executable package".

CREATE A PACKAGE

The screenshot shows the Xcode interface with a project named "session-5". The left sidebar displays the project structure:

- session-5
 - Candy
 - GoodDayToYou
 - .build
 - Sources
 - GoodDayToYou
 - greetings.swift
 - main.swift
 - Tests
 - Package.swift
 - README.md
 - Hello
 - TrickOrTreat

```
1 //print("Hello, world!")  
2  
3 if CommandLine.arguments.count != 2 {  
4     print("Usage: hello NAME")  
5 } else {  
6     let name = CommandLine.arguments[1]  
7     greet(name: name)  
8 }
```

main.swift — TrickOrTreat/Sourc...

main.swift — GoodDayToYou/Sourc...

CREATE A PACKAGE

No need to import anything because they are from the same module

The screenshot shows a Xcode project window. On the left, the Project Navigator displays a package named "GoodDayToYou". It contains a ".build" folder, a "Sources" folder which includes a "GoodDayToYou" module with "greetings.swift" and "main.swift" files, a "Tests" folder, and files "Package.swift" and "README.md". The main editor area shows two files: "greetings.swift" and "main.swift". A yellow callout bubble points from the text in the top right towards the "greetings.swift" file.

```
greetings.swift
1 //print("Hello, world!")
2
3 if CommandLine.arguments.count != 2 {
4     print("Usage: hello NAME")
5 } else {
6     let name = CommandLine.arguments[1]
7     sayHello(name: name)
8 }
9
```

```
main.swift
```

CREATE A PACKAGE

```
017-autumn/session-5/GoodDayToYou
```

```
585 % swift run
```

```
Usage: hello NAME
```

```
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2
```

```
017-autumn/session-5/GoodDayToYou
```

```
[586 % swift run
```

```
Usage: hello NAME
```

```
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2
```

```
017-autumn/session-5/GoodDayToYou
```

```
587 %
```



Needed commandline
argument

CREATE A PACKAGE

```
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2
017-autumn/session-5/GoodDayToYou
[589 % swift run GoodDayToYou Andrew
Good day to you Andrew 😊.
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2
017-autumn/session-5/GoodDayToYou
590 % ]
```

EXECUTABLES WITH PACKAGES

EXECUTABLES WITH PACKAGES

- Create an executable package that uses a dependent package
- Typical behavior

```
# Build it  
% swift build
```

```
Compile Swift Module 'GoodDayToYou' (1 sources)  
Linking ./build/x86_64-apple-macosx10.10/  
debug/GoodDayToYou
```

```
# Run it  
% swift run  
% swift run GoodDayToYou
```

EXECUTABLES WITH PACKAGES

The screenshot shows a file browser window in Xcode. On the left, the project structure is visible:

- session-5 (selected)
- Candy
 - .build
 - Sources
 - Candy (selected)
 - Candy.swift
 - Tests
 - Package.swift
 - README.md- GoodDayToYou
- Hello
- TrickOrTreat

On the right, the code editor displays the contents of Candy.swift:

```
1  public struct Candy {  
2      public let name: String  
3      public let calories: Int  
4      public var eat: Bool?  
5  
6      public init (name: String, calories: Int) {  
7          self.name = name  
8          self.calories = calories  
9      }  
10 }
```

A yellow callout bubble with a black border and a white arrow points from the text "Candy package" towards the Candy.swift file in the code editor.

Candy package

EXECUTABLES WITH PACKAGES

Trick Or Treat Executable

```
session-3
> Candy
> GoodDayToYou
> Hello
> TrickOrTreat
> .build
> Sources
> TrickOrTreat
> main.swift
> Tests
> Package.resolved
> Package.swift
> README.md
```

```
1 // swift-tools-version:4.0
2 // The swift-tools-version declares the minimum version of Swift required to build this package.
3
4 import PackageDescription
5
6 let package = Package(
7     name: "TrickOrTreat",
8     dependencies: [
9         // Dependencies declare other packages that this package depends on.
10        .package(url: "../Candy", from: "1.0.1"),
11    ],
12    targets: [
13        // Targets are the basic building blocks of a package. A target can define
14        // Targets can depend on other targets in this package, and on products
15        .target(
16            name: "TrickOrTreat",
17            dependencies: ["Candy"]),
18        ]
19    )
20
```

EXECUTABLES WITH PACKAGES

The screenshot shows a file structure on the left and a code editor on the right. The file structure includes a folder named 'session-5' containing subfolders 'Candy', 'GoodDayToYou', 'Hello', and 'TrickOrTreat'. 'TrickOrTreat' contains '.build', 'Sources' (with 'TrickOrTreat' and 'main.swift'), 'Tests', 'Package.resolved', 'Package.swift', and 'README.md'. The code editor displays 'Package.swift' with the following content:

```
1 // This file is machine generated - do not edit directly
2 // To edit this file, run the following command instead
3 //   swift package edit --root .
4 import PackageDescription
5
6 let package = Package(
7     name: "TrickOrTreat",
8     dependencies: [
9         // Dependencies declare other packages that this package depends on.
10        .package(url: "../Candy", from: "1.0.1"),
11    ],
12    targets: [
13        // Targets are the basic building blocks of a package. A target can define
14        // Targets can depend on other targets in this package, and on products
15        .target(
16            name: "TrickOrTreat",
17            dependencies: ["Candy"]),
18        ]
19    )
20
```

Two yellow callout boxes highlight the line ".package(url: \"../Candy\", from: \"1.0.1\")" and the line ".target(name: \"TrickOrTreat\", dependencies: [\"Candy\"])" with the text "Use the Candy package".

EXECUTABLES WITH PACKAGES

```
session-5
  > Candy
  > GoodDayToYou
  > Hello
  < TrickOrTreat
    > .build
    < Sources
      < TrickOrTreat
        main.swift
    > Tests
    Package.resolved
    Package.swift
  README.md
```

```
1 import Candy
2
3 let kitkat = Candy(name: "Kit Kat", calories: 200)
4
5 print("🎃 Trick or Treat!")
6 print("Have a \(kitkat.name)")
7 print("😊")
```



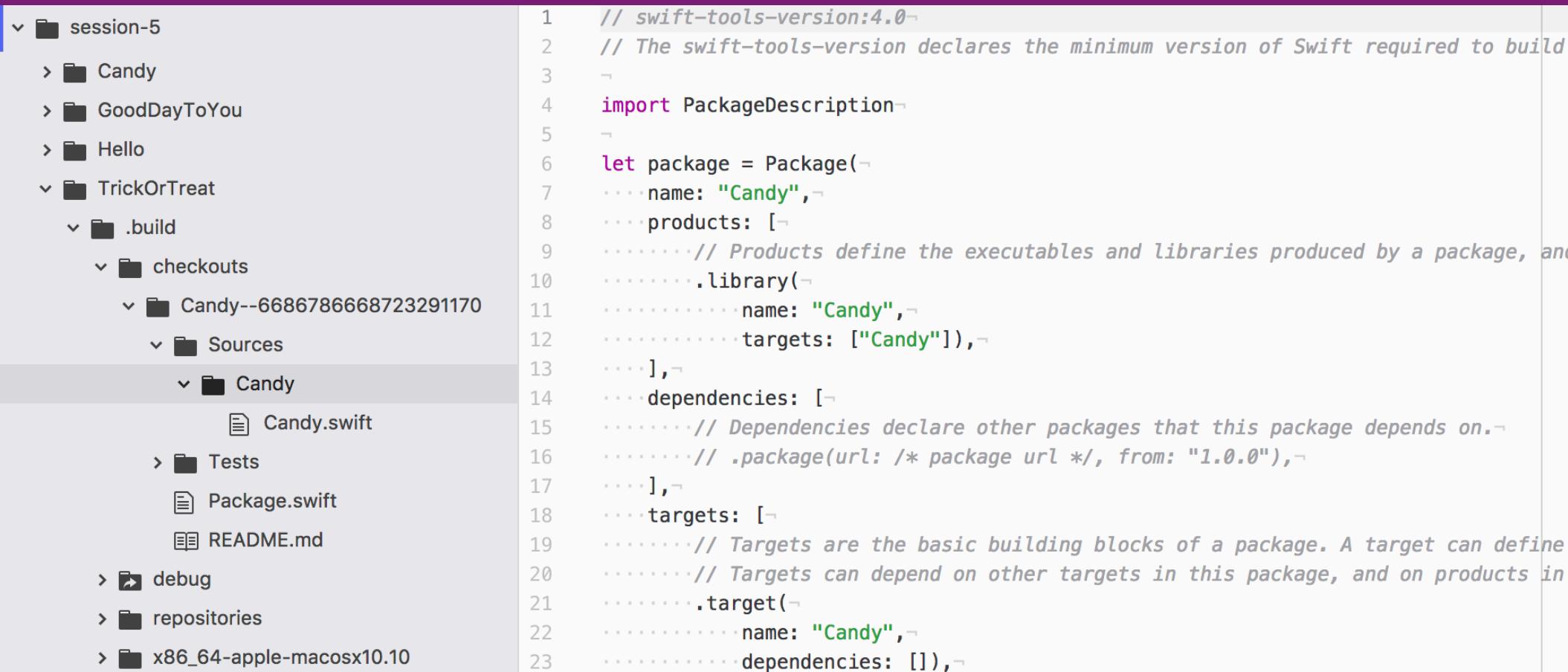
Import Candy

EXECUTABLES WITH PACKAGES

Build the executable

```
[511 % swift build
Fetching /Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/mp
cs51043-2017-autumn/session-5/Candy
Cloning /Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/mpc
s51043-2017-autumn/session-5/Candy
Resolving /Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/m
pcs51043-2017-autumn/session-5/Candy at 1.0.1
Compile Swift Module 'Candy' (1 sources)
Compile Swift Module 'TrickOrTreat' (1 sources)
Linking ./build/x86_64-apple-macosx10.10/debug/TrickOrTreat
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/m
pcs51043-2017-autumn/session-5/TrickOrTreat
512 %
```

EXECUTABLES WITH PACKAGES



The image shows a screenshot of Xcode's interface. On the left, there is a file browser window titled "session-5". It lists several projects and files:

- Candy
- GoodDayToYou
- Hello
- TrickOrTreat
- .build
- checkouts
- Candy--6686786668723291170
- Sources
- Candy
- Candy.swift
- Tests
- Package.swift
- README.md
- debug
- repositories
- x86_64-apple-macosx10.10

On the right, the content area displays the contents of the "Package.swift" file. The code is color-coded and numbered from 1 to 23:

```
1 // swift-tools-version:4.0
2 // The swift-tools-version declares the minimum version of Swift required to build
3
4 import PackageDescription
5
6 let package = Package(
7   name: "Candy",
8   products: [
9     // Products define the executables and libraries produced by a package, and
10    .library(
11      name: "Candy",
12      targets: ["Candy"]),
13    ],
14   dependencies: [
15     // Dependencies declare other packages that this package depends on.
16     // .package(url: /* package url */, from: "1.0.0"),
17   ],
18   targets: [
19     // Targets are the basic building blocks of a package. A target can define
20     // Targets can depend on other targets in this package, and on products in
21     .target(
22       name: "Candy",
23       dependencies: []),
```

WITH XCODE

WITH XCODE

```
Costumes — -bash — 93x14
es.firebaseio/functions — -bash ...1033-2017-autumn-playground — -bash ...mn/session-5/GoodDayToYou — -bash ...-autumn/session-5/Costumes
swift package init --type=executable
g executable package: Costumes
g Package.swift
g README.md
g Sources/
g Sources/Costumes/main.swift
g Tests/
wski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2017-aut
Costumes
swift package generate-xcodeproj
ed: ./Costumes.xcodeproj
wski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2017-aut
Costumes
```

WITH XCODE

The screenshot shows the Xcode interface with a Swift package named "Costumes". The left sidebar displays the project structure:

- Costumes (Project root)
- Package.swift (selected)
- Sources
 - Costumes (Folder)
 - main.swift
- Tests
- Products

The main editor window shows the contents of Package.swift:

```
1 // swift-tools-version:4.0
2 // The swift-tools-version declares the minimum version of Swift required to build this package.
3
4 import PackageDescription
5
6 let package = Package(
7     name: "Costumes",
8     dependencies: [
9         // Dependencies declare other packages that this package depends on.
10        // .package(url: /* package url */, from: "1.0.0"),
11    ],
12    targets: [
13        // Targets are the basic building blocks of a package. A target can define a module or a test suite.
14        // Targets can depend on other targets in this package, and on products in packages which this package depends
15        // on.
16        .target(
17            name: "Costumes",
18            dependencies: []),
19    ]
20 )
```

The bottom right corner of the editor shows the output of a build or run command:

```
Hello, world!
Program ended with exit code: 0
```

WITH XCODE

The screenshot shows the Xcode interface with a project named "Costumes". The left sidebar lists files like Package.swift, main.swift, and test cases. The main area is a "Build Settings" tab for the "Costumes" target. A large yellow box covers the bottom-left portion of the screen with the word "ERROR" in black capital letters.

Costumes

General Resource Tags Build Settings Build Phases Build Rules

Basic Customized All Combined Levels + supported

Architectures

Setting Costumes

Supported Platforms macOS

Apple LLVM 9.0 - Code Generation

Setting Costumes

Optimization Level

Debug <Multiple values>
Release None [-O0]
Fastest, Smallest [-Os]

Identity and Type

Name Costumes
Location Absolute
Containing directory
Full Path /Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2017-autumn/session-5/Costumes/Costumes.xcodeproj

Project Document

Project Format Xcode 3.2-compatible
Organization
Class Prefix

Text Settings

Cocoa Touch Class - A Cocoa Touch class

UI Test Case Class - A class implementing a unit test

Unit Test Case Class - A class implementing a unit test

THIRD PARTY PACKAGES

THIRD PARTY PACKAGES

Elegant Networking in Swift

build passing

pod v4.5.1

Carthage compatible

platform ios | osx | tvos | watchos

twitter @AlamofireSF

chat on gitter

Alamofire is an HTTP networking library written in Swift.

- [Features](#)
- [Component Libraries](#)
- [Requirements](#)
- [Migration Guides](#)
- [Communication](#)
- [Installation](#)
- [Usage](#)
 - [Intro - Making a Request, Response Handling, Response Validation, Response Caching](#)
 - [HTTP - HTTP Methods, Parameter Encoding, HTTP Headers, Authentication](#)

THIRD PARTY PACKAGES

```
$ pod install
```

Carthage

[Carthage](#) is a decentralized dependency manager that builds your dependencies and provides you with binary frameworks.

You can install Carthage with [Homebrew](#) using the following command:

```
$ brew update  
$ brew install carthage
```

To integrate Alamofire into your Xcode project using Carthage, specify it in your `Cartfile`:

```
github "Alamofire/Alamofire" ~> 4.5
```

Run `carthage update` to build the framework and drag the built `Alamofire.framework` into your Xcode project.

THIRD PARTY PACKAGES

The screenshot shows a Xcode project window with the following structure:

- Project**: session-5
- session-5**:
 - Candy
 - GoodDayToYou
 - Hello
 - TrickOrTreat**:
 - .build
 - Sources
 - TrickOrTreat
 - main.swift
 - Tests
 - Package.resolved
 - Package.swift
 - README.md

The **Package.swift** file is open in the editor, showing the following code:

```
1 swift-tools-version:4.0
2 The swift-tools-version declares the minimum version of Swift required to build this package.
3
4 import PackageDescription
5
6 package = Package(
7   name: "TrickOrTreat",
8   dependencies: [
9     // Dependencies declare other packages that this package depends on.
10    .package(url: "../Candy", from: "1.0.1"),
11    .package(url: "https://github.com/Alamofire/Alamofire.git", from: "4.5.0"),
12  ],
13   targets: [
14     // Targets are the basic building blocks of a package. A target can define a module or a test suite.
15     // Targets can depend on other targets in this package, and on products in packages which this package depends on.
16     .target(
17       name: "TrickOrTreat",
18       dependencies: ["Candy", "Alamofire"]),
19   ]
20
21
```

THIRD PARTY PACKAGES

The screenshot shows a GitHub repository page for a project named "third-party-packages". The top navigation bar includes links for "Code", "Issues", "Pull requests", "Wiki", "Actions", "Marketplace", and "Settings". Below the navigation, there are tabs for "awesome-lists", "ios", and "linux". Key statistics displayed are 2,591 commits, 2 branches, 0 releases, 387 contributors, and a CC0-1.0 license.

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

File / Commit	Description	Date
READMEbot [auto] [ci skip] Generate README & Database.json		Latest commit edc8e53 a day ago
.github	Update Hi.swift	16 days ago
.gitignore	fix push script, now disabled	4 months ago
.travis.yml	re-ordered scripts	4 months ago
CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md	4 months ago
Dangerfile	Create Dangerfile	a month ago
LICENSE	Initial commit	3 years ago
README.md	[auto] [ci skip] Generate README & Database.json	a day ago
contents.json	Merge pull request #1063 from eneko/feature/add-sourcedocs	a day ago
database.json	[auto] [ci skip] Generate README & Database.json	a day ago
README.md		

THIRD PARTY PACKAGES

A collection of functions for statistical calculation written in Swift.

statistics swift

344 commits 4 branches 26 releases 3 contributors MIT

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

Author	Commit Message	Latest commit
 Evgenii Neumerzhitckii	Update pod version	f4ad79c 28 days ago
 Distrib	Update to Swift 3.1	7 months ago
 Graphics	Add kurtosis	10 months ago
 SigmaSwiftStatistics-Mac	Update to swift 2.0	2 years ago
 SigmaSwiftStatistics-Watch	Add target for Watch	2 years ago
 SigmaSwiftStatistics-tvOS	Add watchOS support	2 years ago
 SigmaSwiftStatistics.xcodeproj	Update to Swift 4.0	5 months ago
 SigmaSwiftStatistics	Update to Swift 3.1	7 months ago
 SigmaSwiftStatisticsTests	Add rank function	9 months ago

OTHER PACKAGE MANAGERS

OTHER PACKAGE MANAGERS

- Package managers for Swift (iOS)
 - CocoaPods
 - Carthage

SEARCH*



* Type here to search by name, version, author, keywords, summary, and dependencies.

WHAT IS COCOAPODS

CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It has over 38 thousand libraries and is used in over 2.7 million apps. CocoaPods can

OTHER PACKAGE MANAGERS

- Cocoa pods is a centralized repository
- Used heavily throughout industry

SEARCH*



* Type here to search by name, version, author, keywords, summary, and dependencies.

WHAT IS COCOAPODS

CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It has over 38 thousand libraries and is used in over 2.7 million apps. CocoaPods can

OTHER PACKAGE MANAGERS

- [https://
www.youtube.com
/watch?
v=iEAjvNRdZa0](https://www.youtube.com/watch?v=iEAjvNRdZa0)



OTHER PACKAGE MANAGERS

- Downloads everything to your project and creates a workspace
- Similar to SPM

SEARCH*



* Type here to search by name, version, author, keywords, summary, and dependencies.

WHAT IS COCOAPODS

CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It has over 38 thousand libraries and is used in over 2.7 million apps. CocoaPods can

OTHER PACKAGE MANAGERS

- Decentralized
 - No point of failure
- Relies on xcode tools



Carthage

license MIT release v0.26.2

Carthage is intended to be the simplest way to add frameworks to your Cocoa application.

The basic [workflow](#) looks something like this:

1. Create a [Cartfile](#) that lists the frameworks you'd like to use in your project.
2. [Run Carthage](#), which fetches and builds each framework you've listed.
3. Drag the built `.framework` binaries into your application's Xcode project.

Carthage builds your dependencies and provides you with binary frameworks, but you retain full control over your project structure and setup. Carthage does not automatically modify your project files or your build settings.

Differences between Carthage and CocoaPods

[CocoaPods](#) is a long-standing dependency manager for Cocoa. So why was Carthage created?

Firstly, CocoaPods (by default) automatically creates and updates an Xcode workspace for your application and all dependencies. Carthage builds framework binaries using `xcodebuild`, but leaves the responsibility of integrating them up to the user. CocoaPods' approach is easier to use, while Carthage's is flexible and unintrusive.

The goal of CocoaPods is listed in its [README](#) as follows:

... to improve discoverability of, and engagement in, third party open-source libraries, by creating a more centralized ecosystem.

By contrast, Carthage has been created as a decentralized dependency manager. There is no central list of projects.

OTHER PACKAGE MANAGERS

- Downloads libraries to local machine, compiles and copies library to your project



Carthage

license MIT release v0.26.2

Carthage is intended to be the simplest way to add frameworks to your Cocoa application.

The basic [workflow](#) looks something like this:

1. Create a [Cartfile](#) that lists the frameworks you'd like to use in your project.
2. [Run Carthage](#), which fetches and builds each framework you've listed.
3. Drag the built `.framework` binaries into your application's Xcode project.

Carthage builds your dependencies and provides you with binary frameworks, but you retain full control over your project structure and setup. Carthage does not automatically modify your project files or your build settings.

Differences between Carthage and CocoaPods

[CocoaPods](#) is a long-standing dependency manager for Cocoa. So why was Carthage created?

Firstly, CocoaPods (by default) automatically creates and updates an Xcode workspace for your application and all dependencies. Carthage builds framework binaries using `xcodebuild`, but leaves the responsibility of integrating them up to the user. CocoaPods' approach is easier to use, while Carthage's is flexible and unintrusive.

The goal of CocoaPods is listed in its [README](#) as follows:

... to improve discoverability of, and engagement in, third party open-source libraries, by creating a more centralized ecosystem.

By contrast, Carthage has been created as a decentralized dependency manager. There is no central list of projects.

OTHER PACKAGE MANAGERS

- Not as popular, but some people prefer their approach



Carthage

license MIT release v0.26.2

Carthage is intended to be the simplest way to add frameworks to your Cocoa application.

The basic [workflow](#) looks something like this:

1. Create a [Cartfile](#) that lists the frameworks you'd like to use in your project.
2. [Run Carthage](#), which fetches and builds each framework you've listed.
3. Drag the built `.framework` binaries into your application's Xcode project.

Carthage builds your dependencies and provides you with binary frameworks, but you retain full control over your project structure and setup. Carthage does not automatically modify your project files or your build settings.

Differences between Carthage and CocoaPods

[CocoaPods](#) is a long-standing dependency manager for Cocoa. So why was Carthage created?

Firstly, CocoaPods (by default) automatically creates and updates an Xcode workspace for your application and all dependencies. Carthage builds framework binaries using `xcodebuild`, but leaves the responsibility of integrating them up to the user. CocoaPods' approach is easier to use, while Carthage's is flexible and unintrusive.

The goal of CocoaPods is listed in its [README](#) as follows:

... to improve discoverability of, and engagement in, third party open-source libraries, by creating a more centralized ecosystem.

By contrast, Carthage has been created as a decentralized dependency manager. There is no central list of projects.

OTHER PACKAGE MANAGERS

- Swift Package Manager will take these over eventually
- Many popular frameworks on iOS use CocoaPods
- The most popular use all three



THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 9

BACKENDS FOR MOBILE APPLICATIONS