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

- It all started here

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.

Swift.org

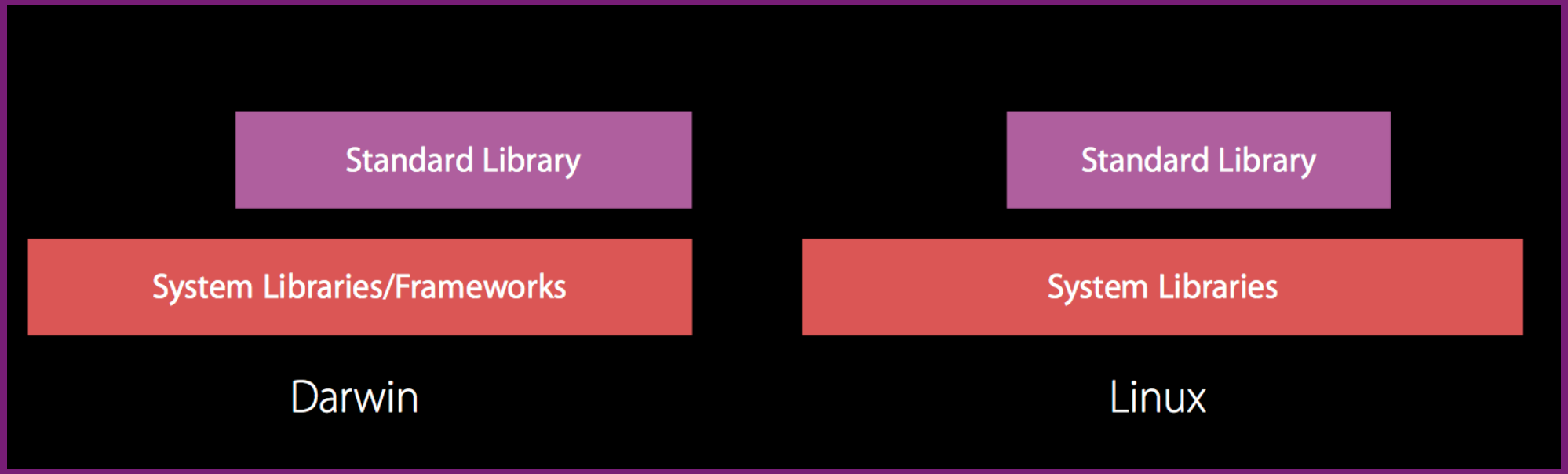
Swift.org is an entirely new site dedicated to open source Swift. This site hosts resources for the community of developers that want to help evolve Swift, contribute fixes, and most importantly, interact with each other. Swift.org hosts:

- 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

OPEN SOURCE SWIFT

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

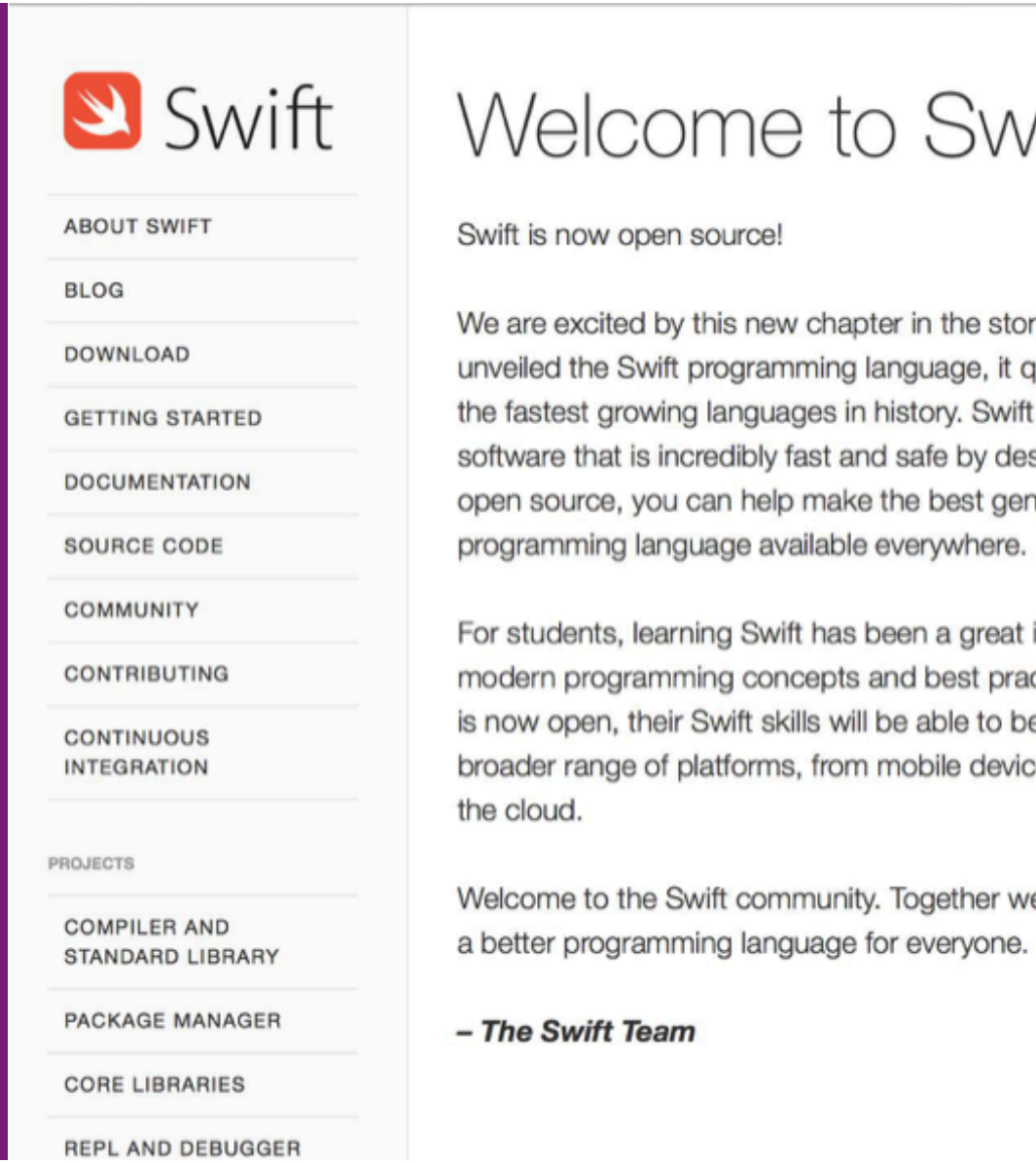
OPEN SOURCE SWIFT



- Something is missing...

OPEN SOURCE SWIFT

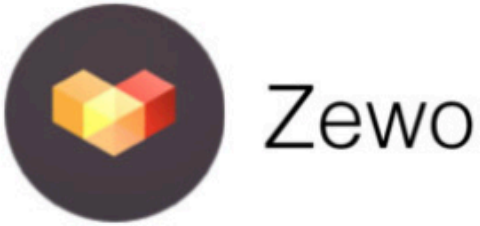
- Linux and Mac Platform support
- Standard Library Foundation, Dispatch, and XCTest Compiler
- Command Line Tools



STATE OF SWIFT ON THE SERVER

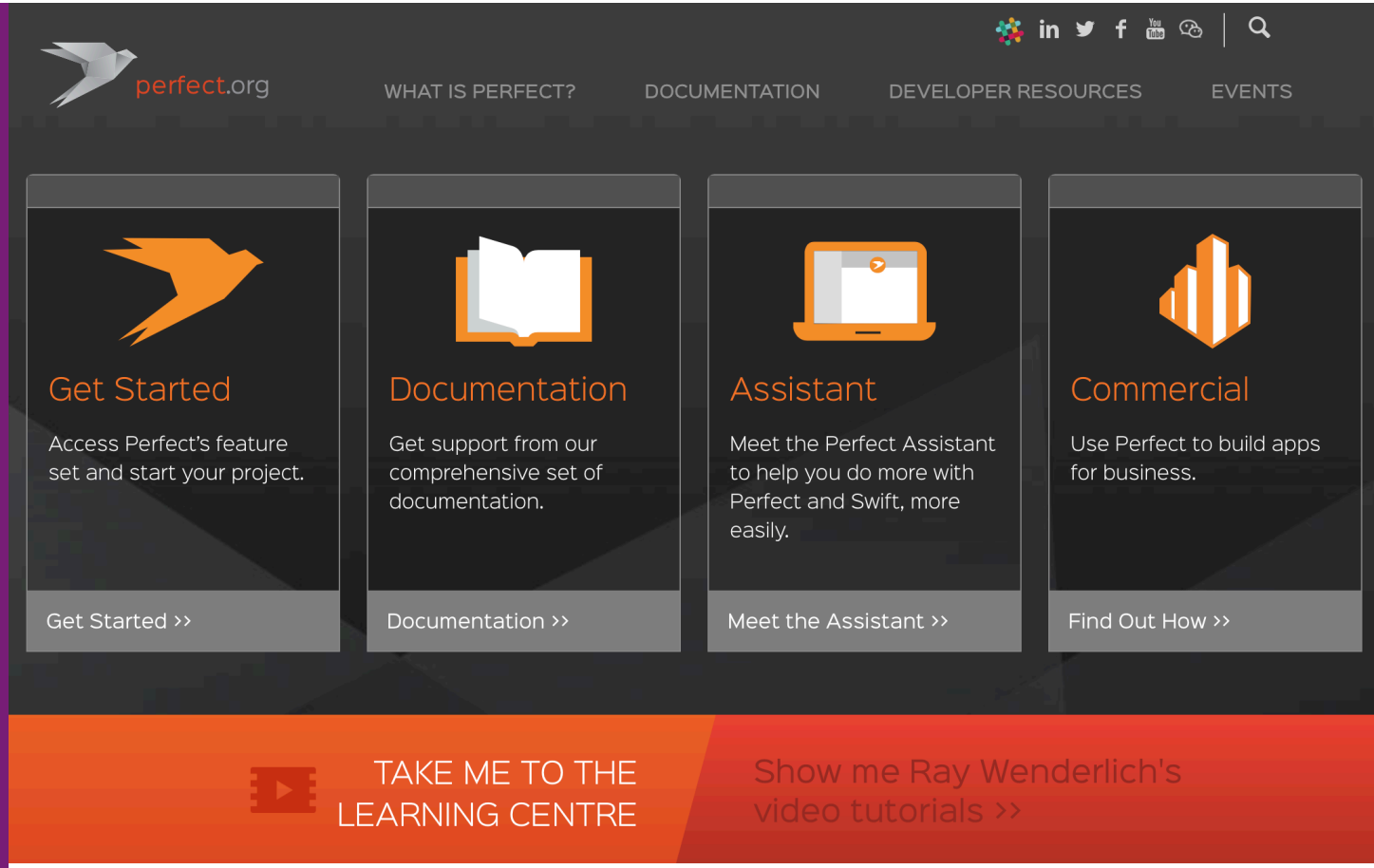
STATE OF SWIFT ON THE SERVER

- 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



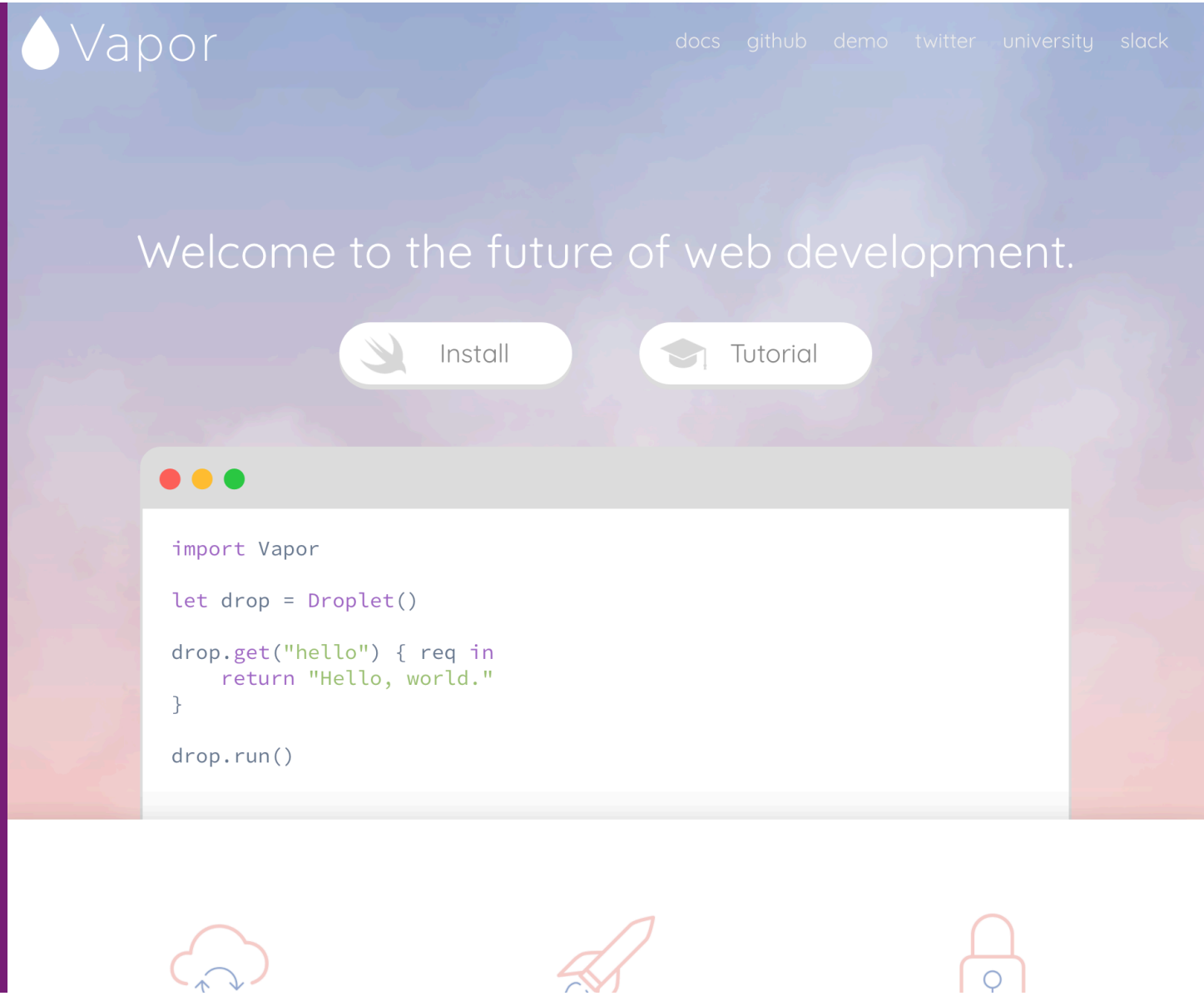
STATE OF SWIFT ON THE SERVER

- Perfect
  - A complete framework
  - Runs it 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



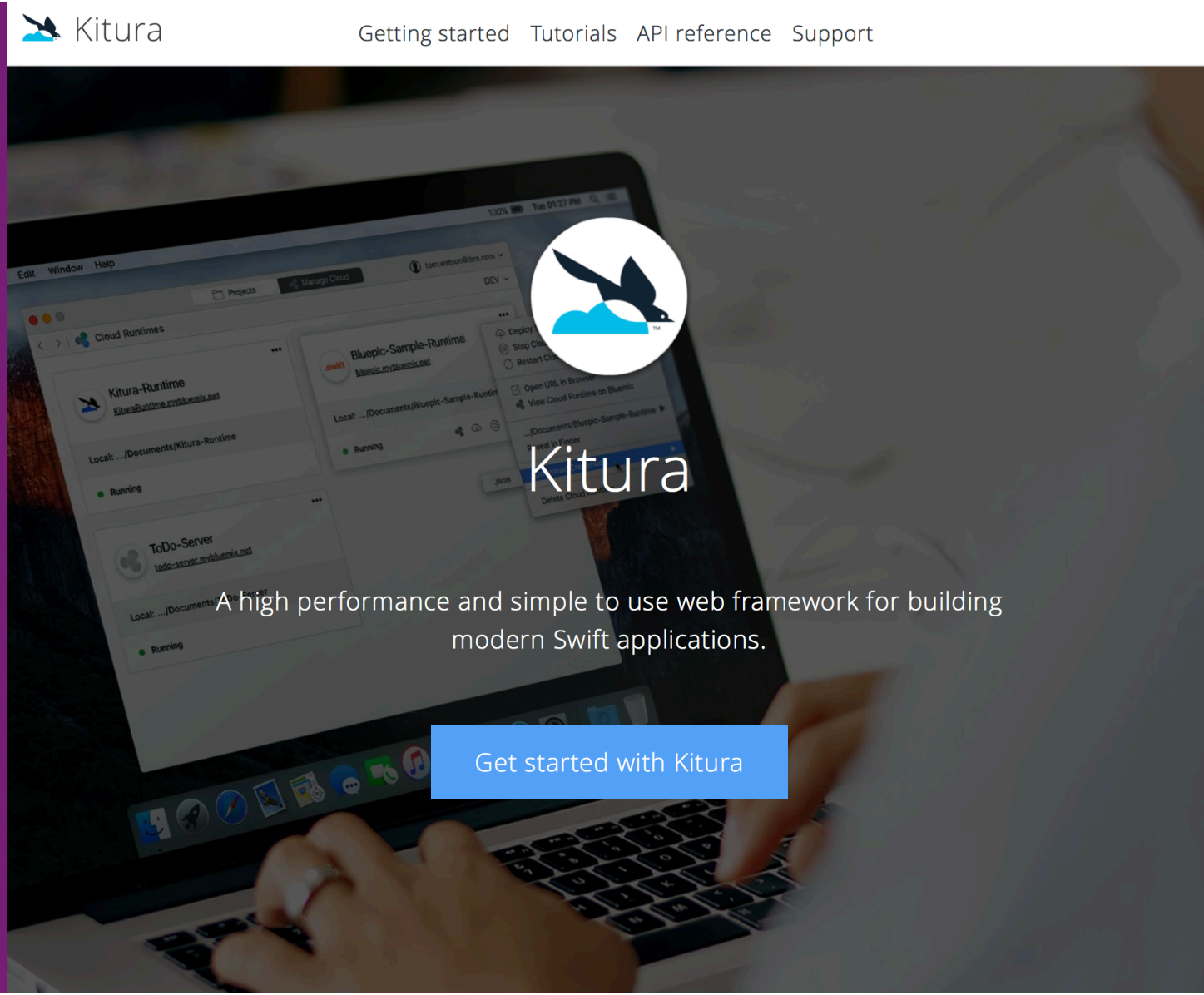
STATE OF SWIFT ON THE SERVER

- Perfect
  - A complete framework
  - Runs it 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



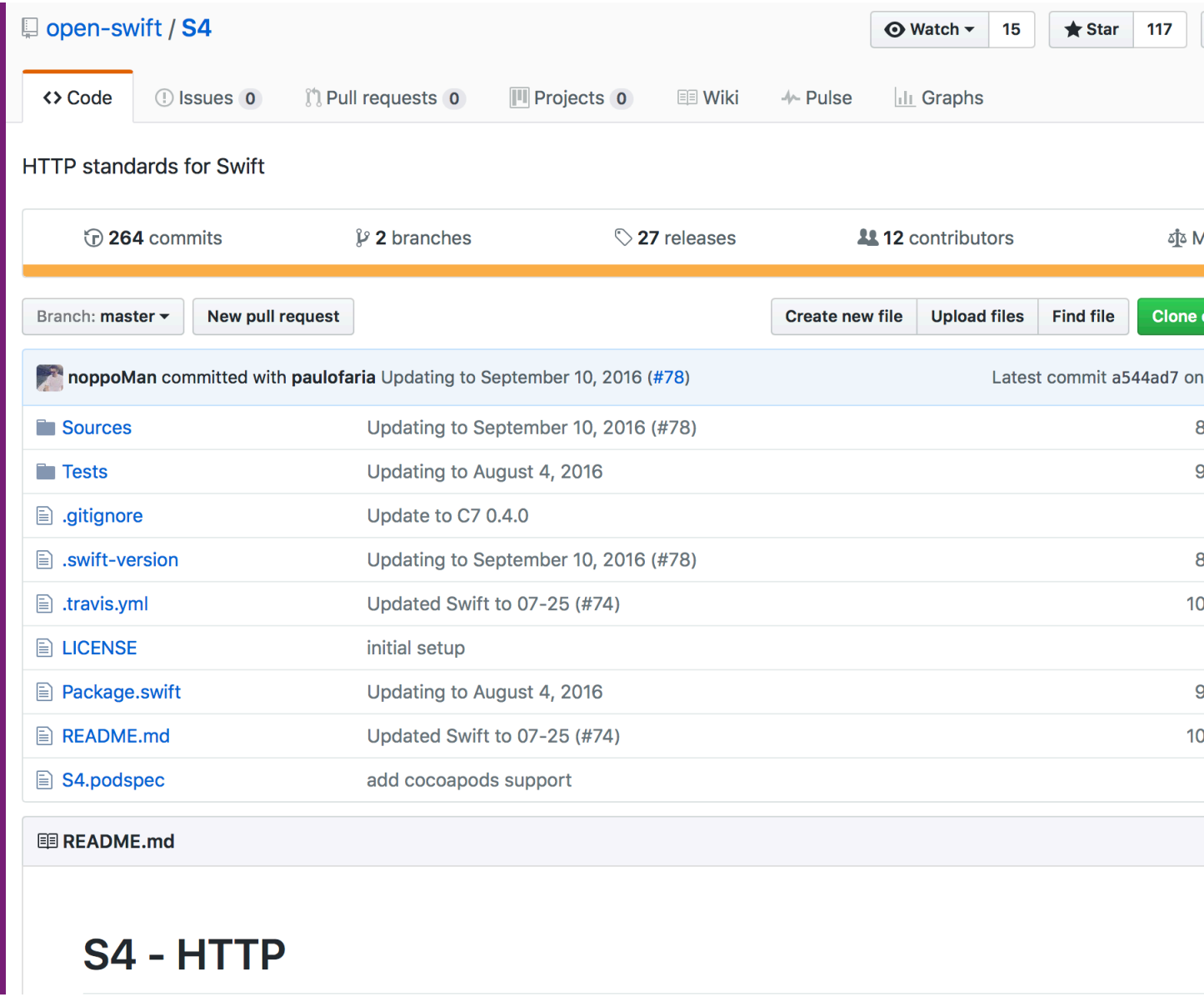
STATE OF SWIFT ON THE SERVER

- 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



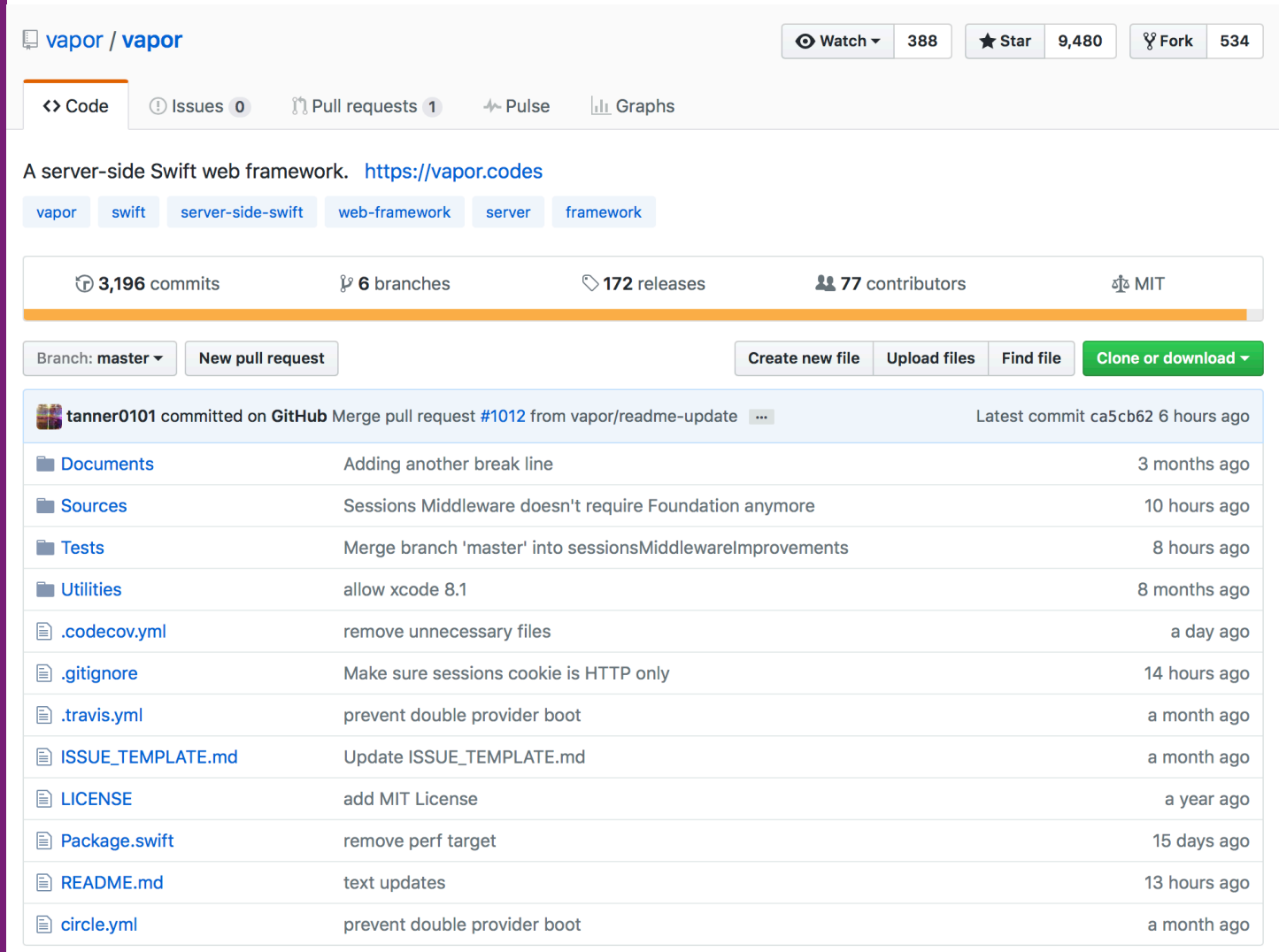
STATE OF SWIFT ON THE SERVER

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

VAPOR

VAPOR

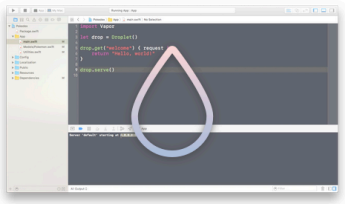
- Web framework and server for Swift
- Works on macOS and Ubuntu



VAPOR

- Great documentation and resources

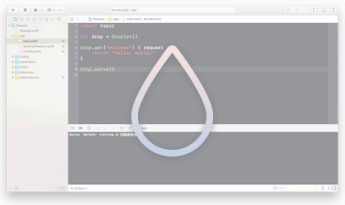
vapor.university



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

VAPOR

VAPOR Docs

GETTING STARTED

Install Swift 3: macOS
Install Swift 3: Ubuntu
Install Toolbox
Hello, World
Manual
Xcode

GUIDE

Droplet
Folder Structure
JSON
Config
Views
Leaf

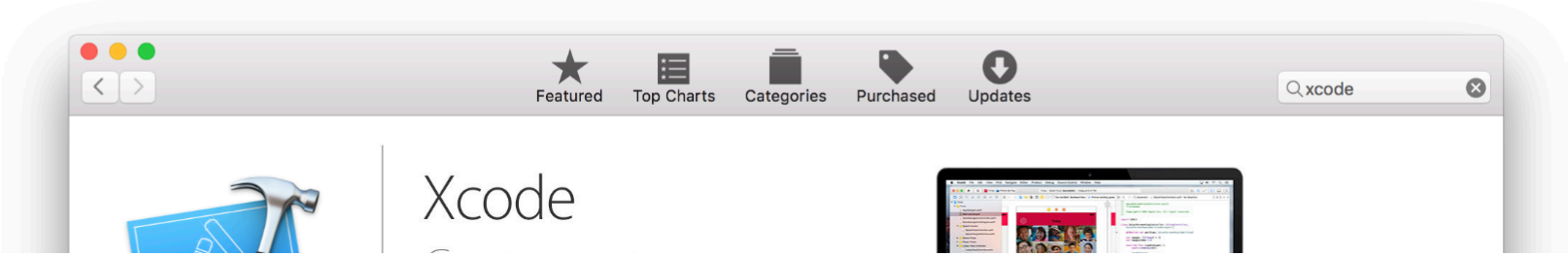
✎ Edit on GitHub

Install Swift 3: macOS

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

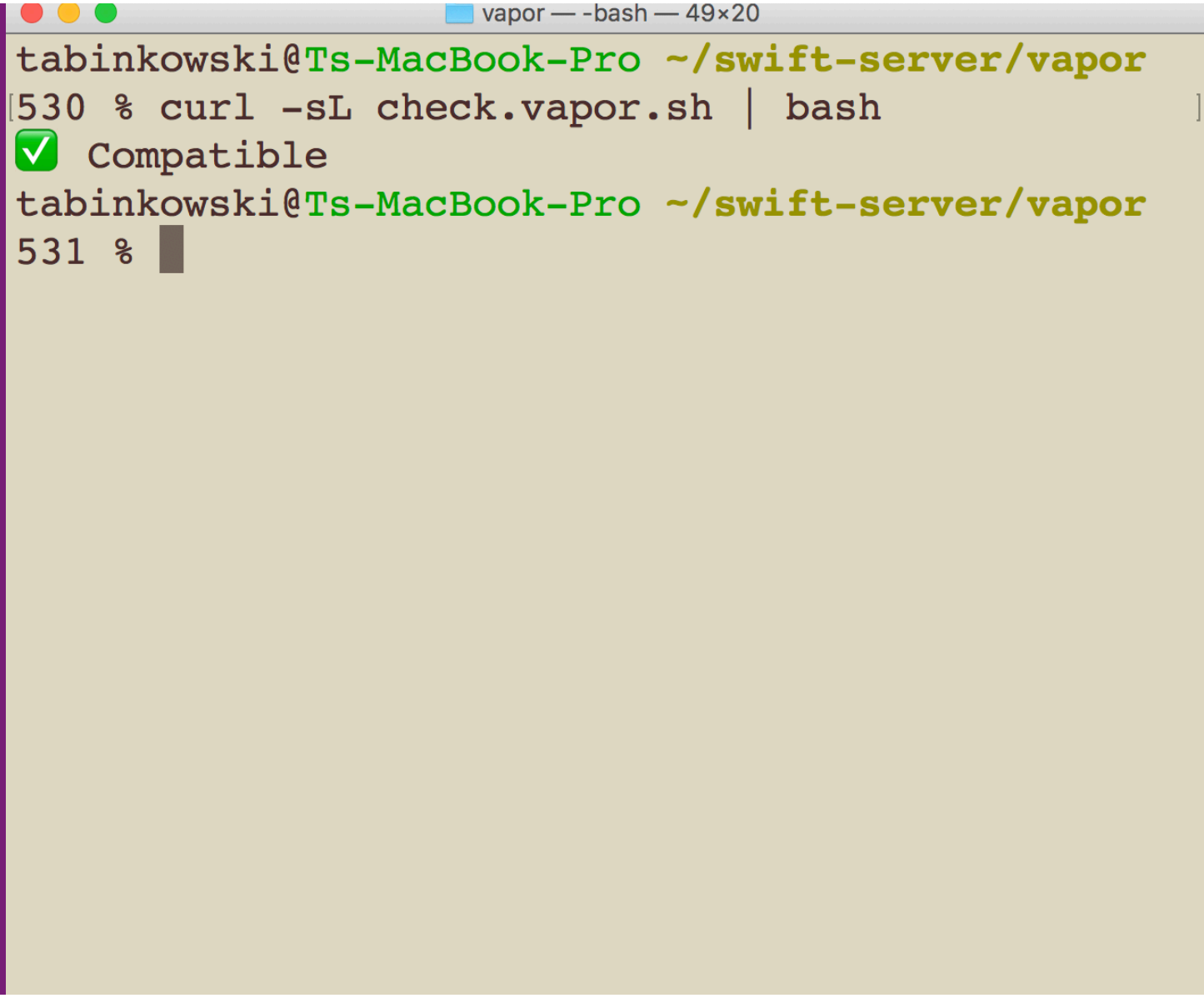
Install Xcode

Install Xcode 8 from the Mac App Store.



VAPOR

- Install Xcode
- Check version for vapor

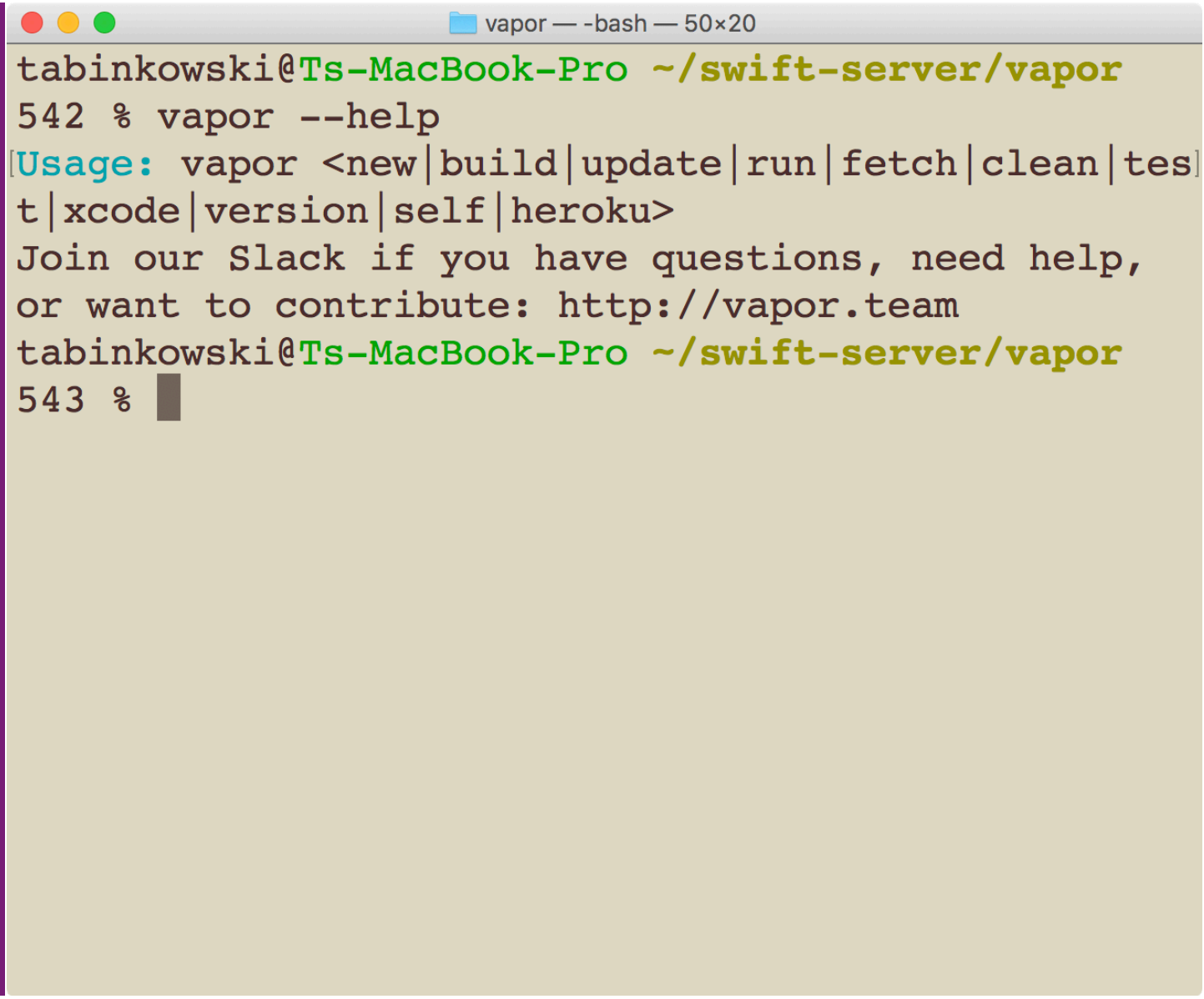


VAPOR

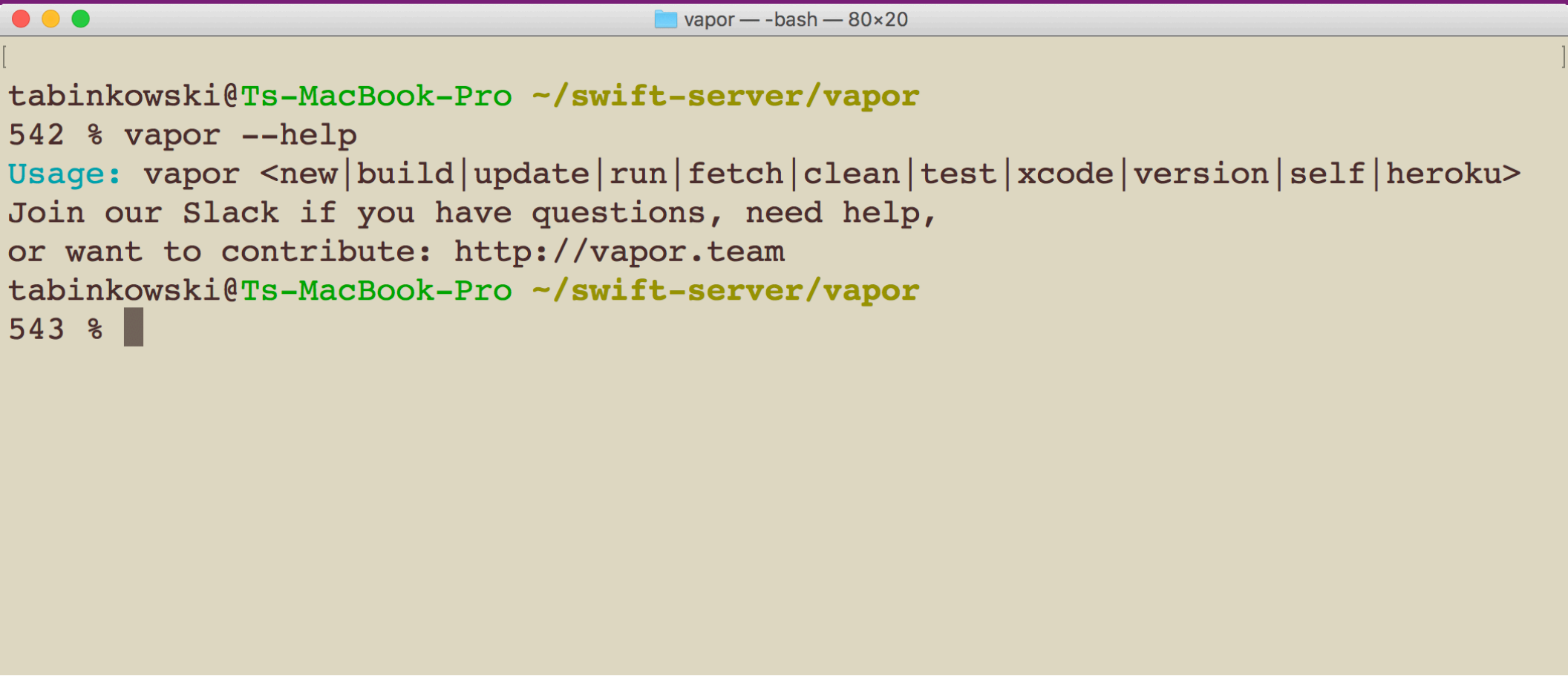
- Toolbox provides command line and shortcuts for common tasks
- Install with homebrew 🍺

VAPOR

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



VAPOR

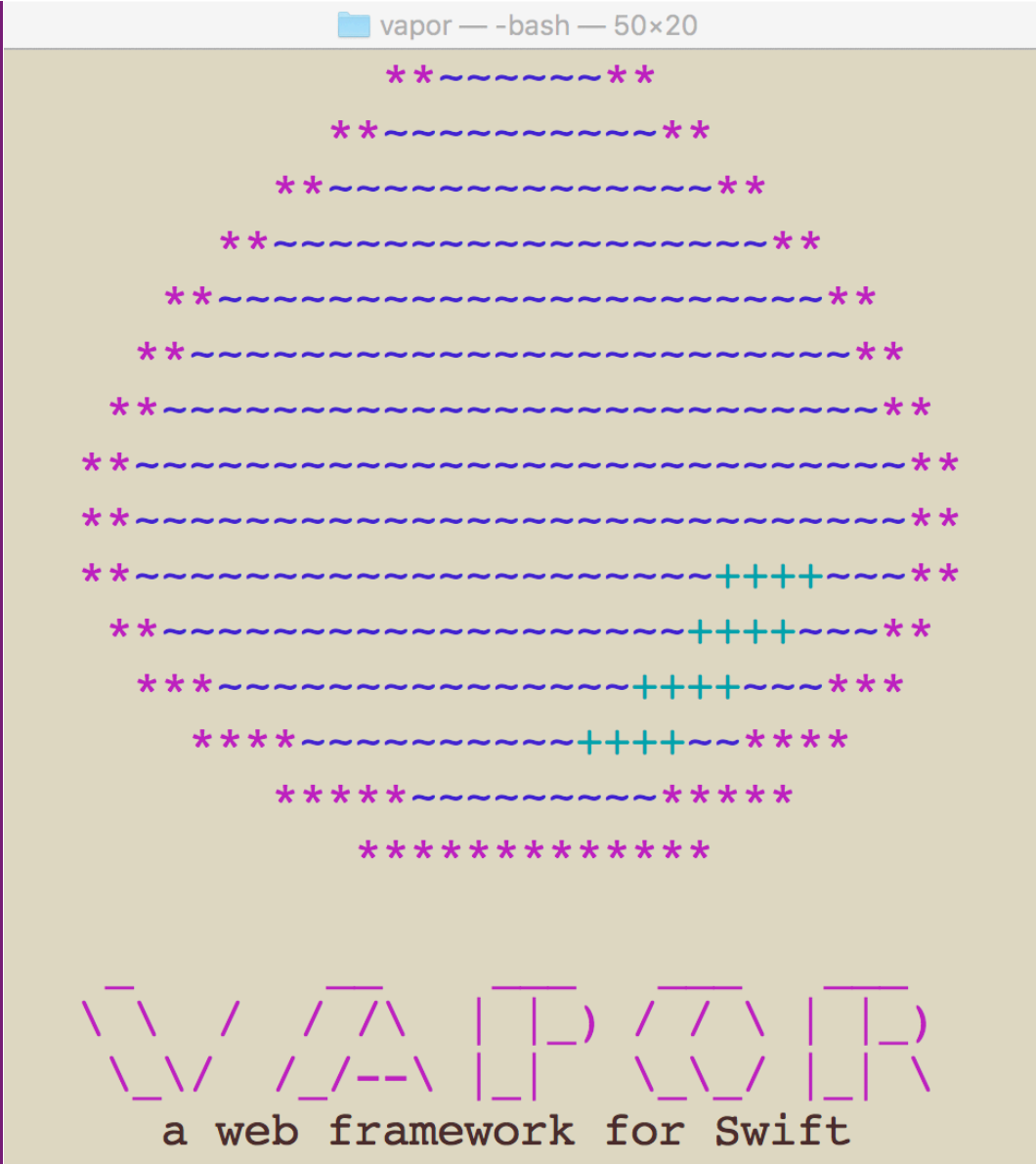

vapor — -bash — 80×20
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
542 % vapor --help
Usage: vapor <new|build|update|run|fetch|clean|test|xcode|version|self|heroku>
Join our Slack if you have questions, need help,
or want to contribute: http://vapor.team
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor
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>]
```

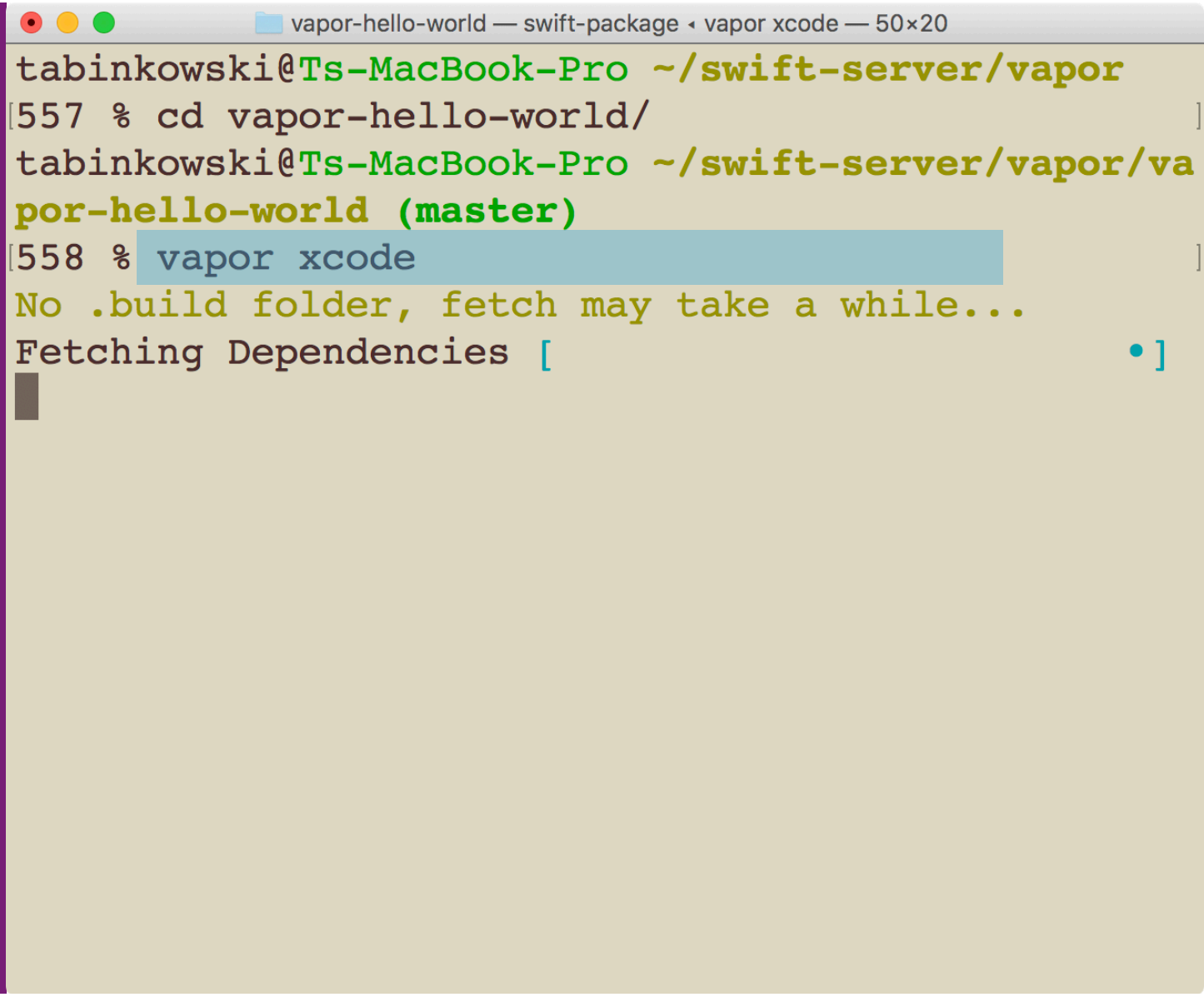
VAPOR

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



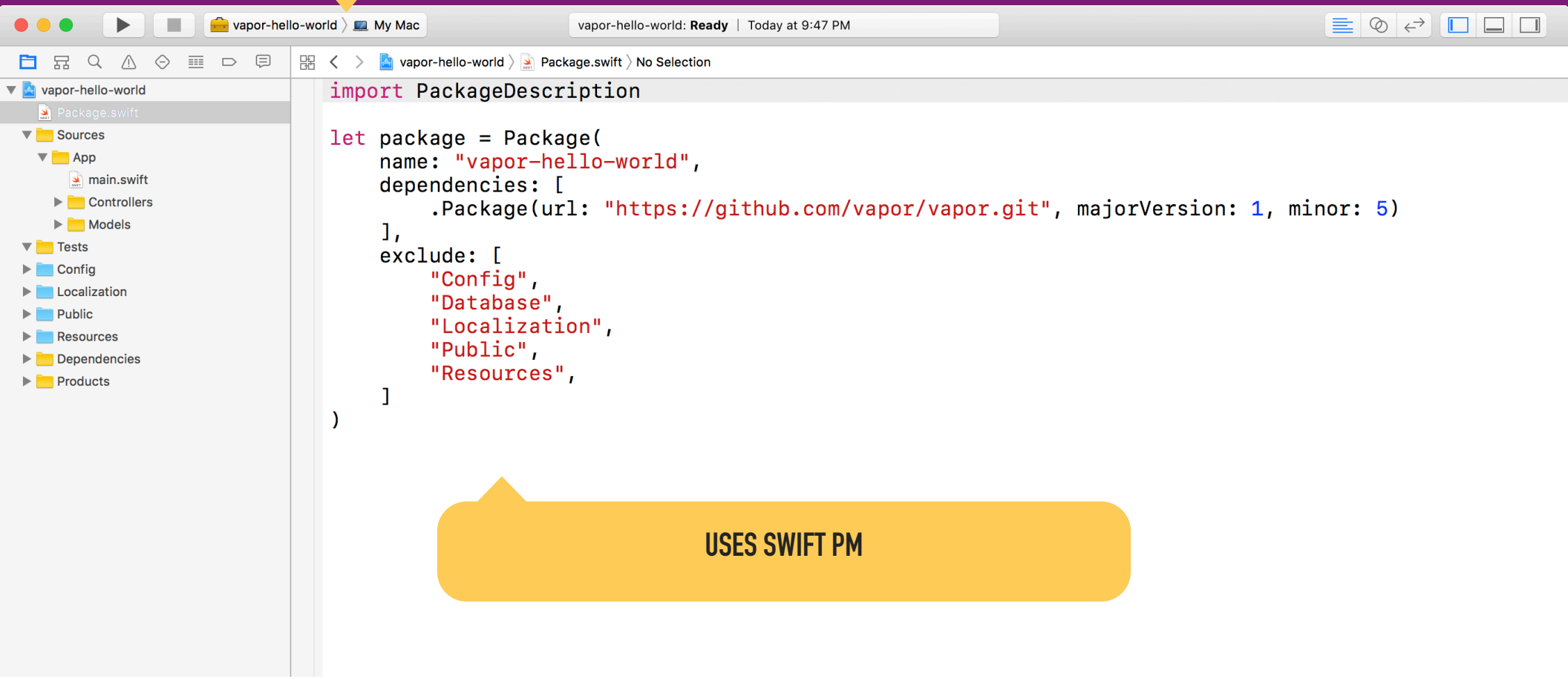
VAPOR

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



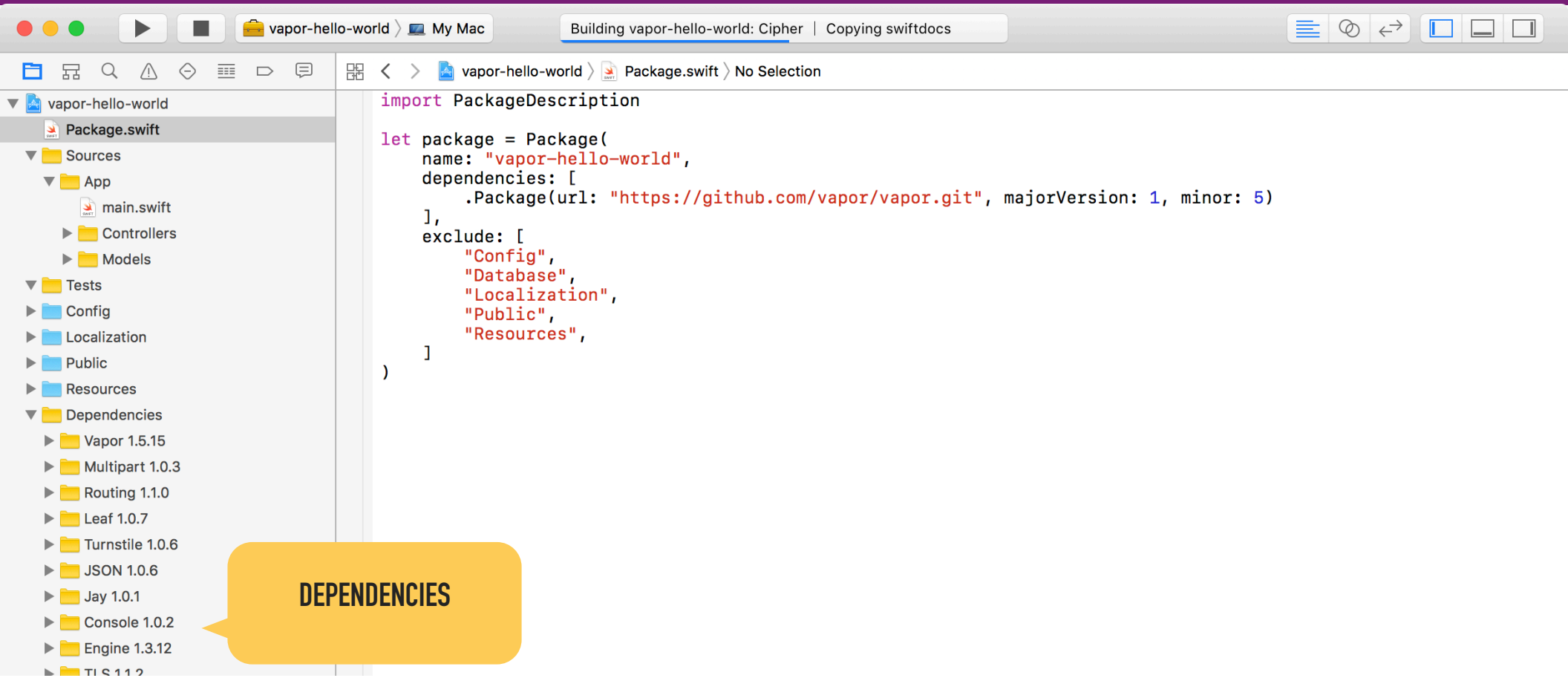
VAPOR

BUILDS A FRAMEWORK AND APP



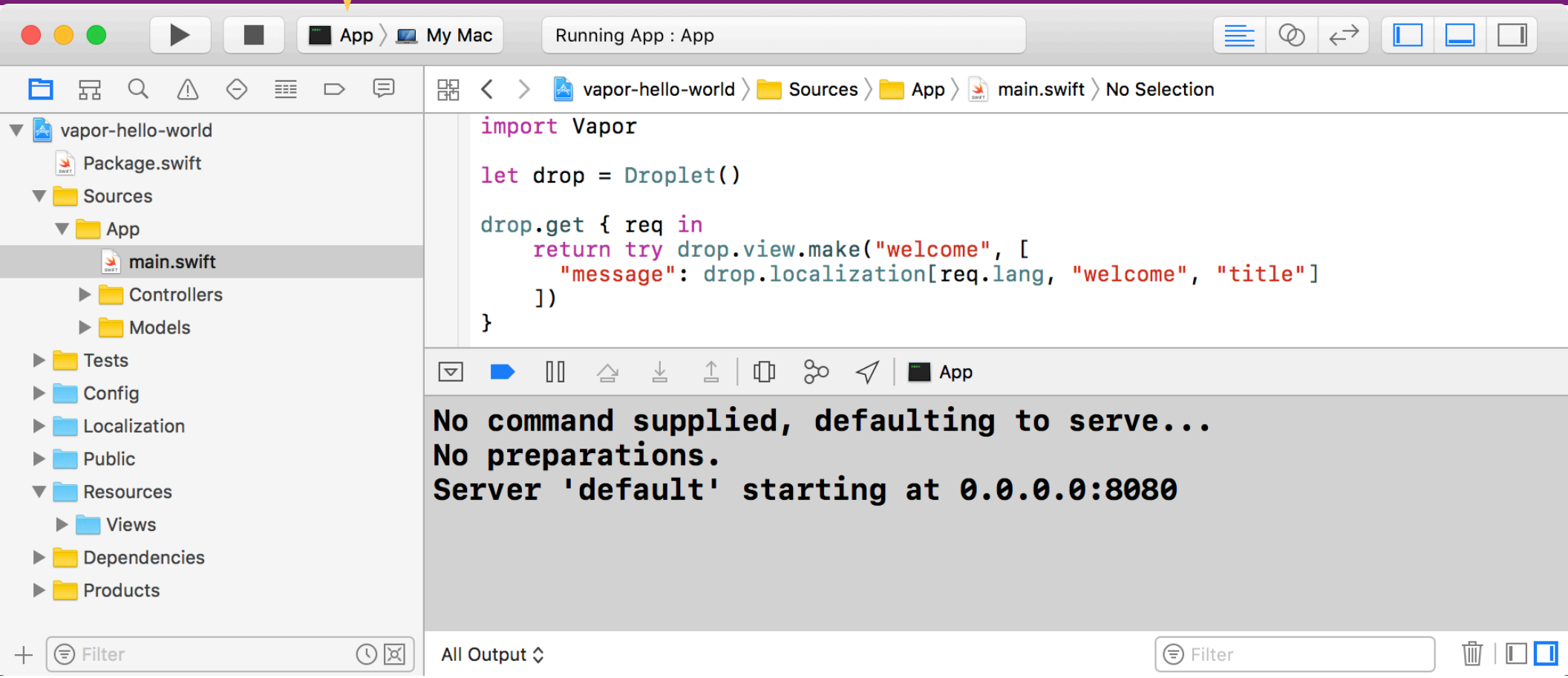
USES SWIFT PM

VAPOR


vapor-hello-world 〉 My Mac
Building vapor-hello-world: Cipher | Copying swiftdocs
vapor-hello-world 〉 Package.swift 〉 No Selection
vapor-hello-world
Package.swift
Sources
App
main.swift
Controllers
Models
Tests
Config
Localization
Public
Resources
Dependencies
Vapor 1.5.15
Multipart 1.0.3
Routing 1.1.0
Leaf 1.0.7
Turnstile 1.0.6
JSON 1.0.6
Jay 1.0.1
Console 1.0.2
Engine 1.3.12
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",
]
)
DEPENDENCIES

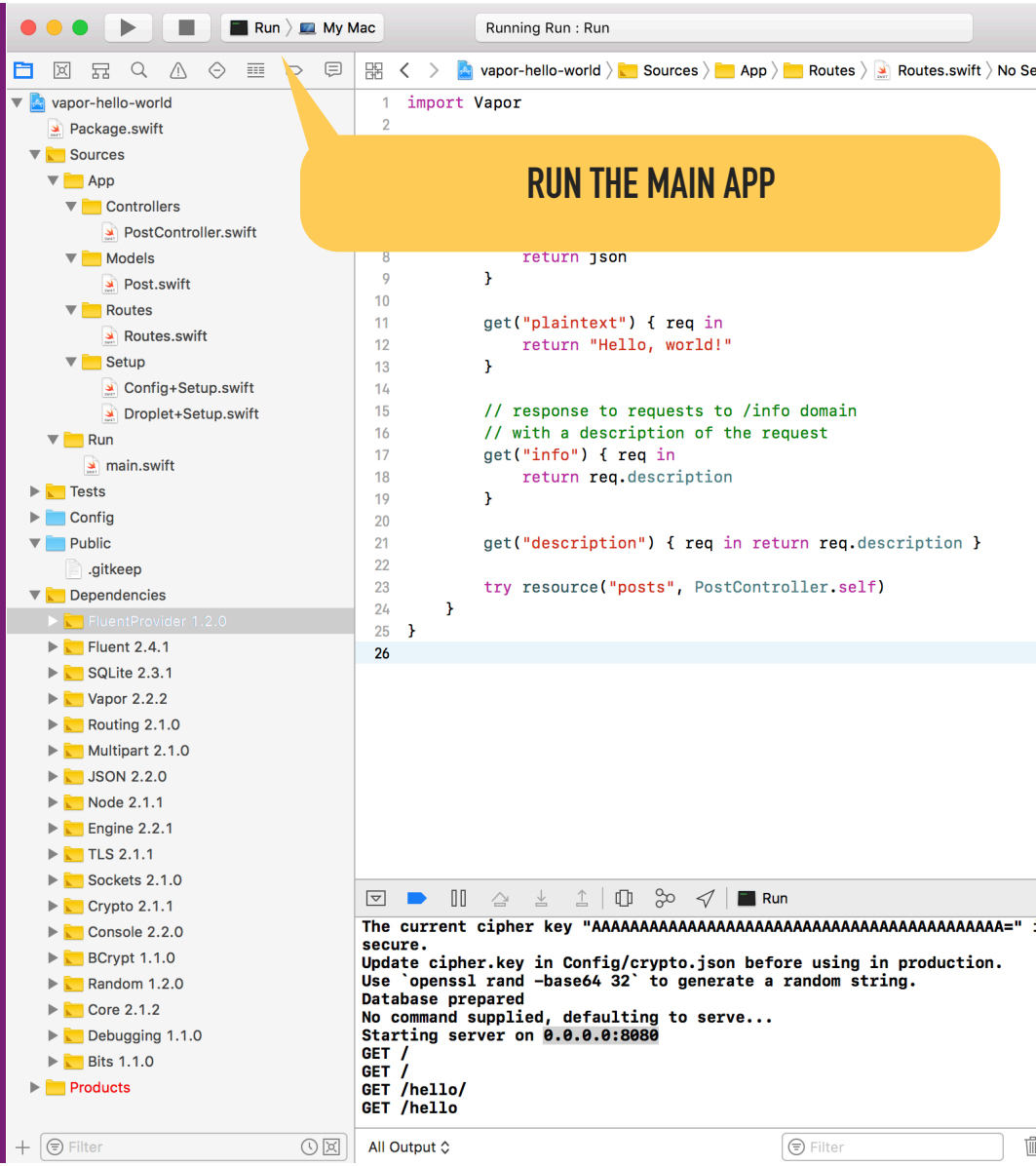
VAPOR

RUN APP



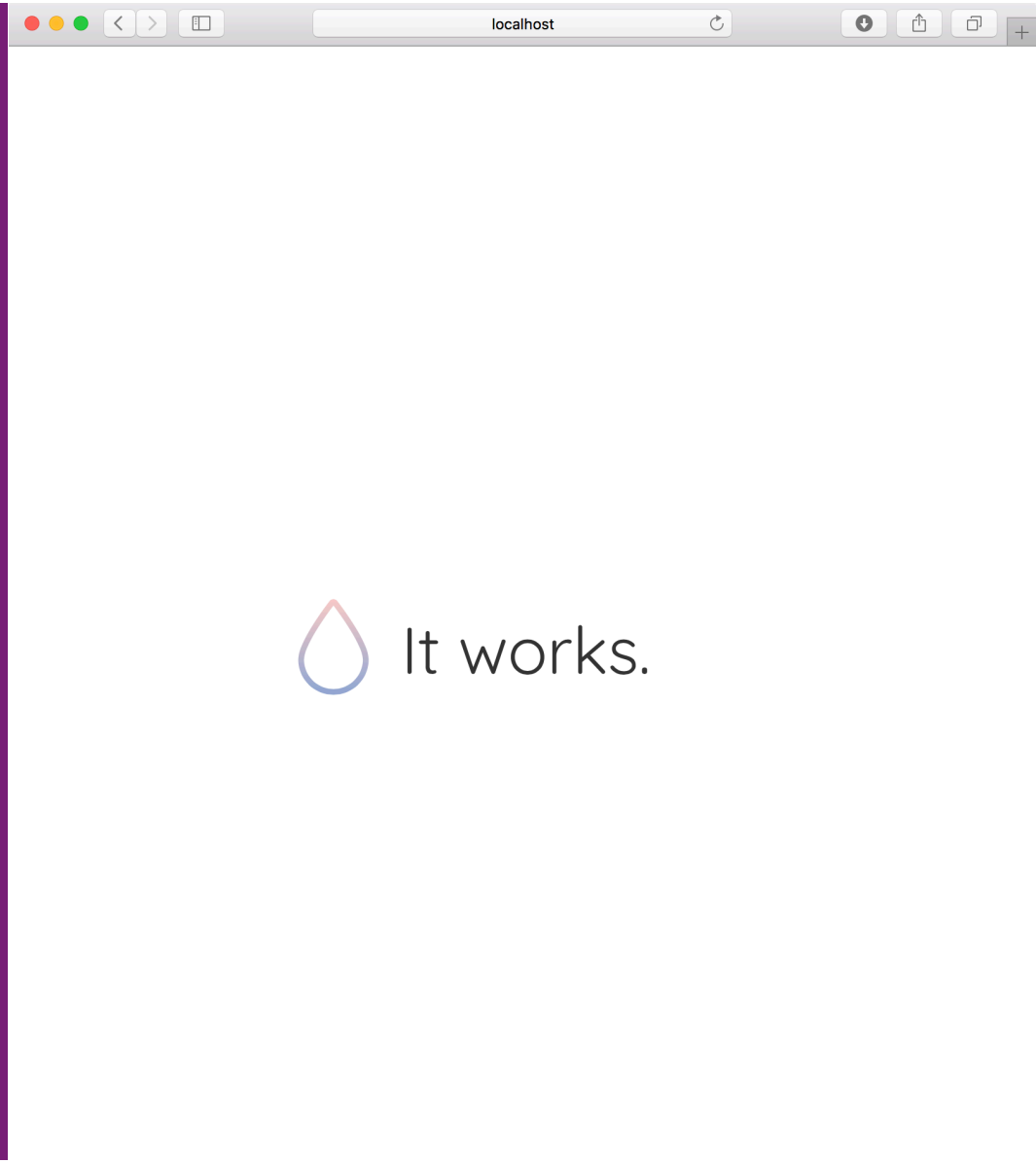
VAPOR

- Builds all the package dependencies and links them



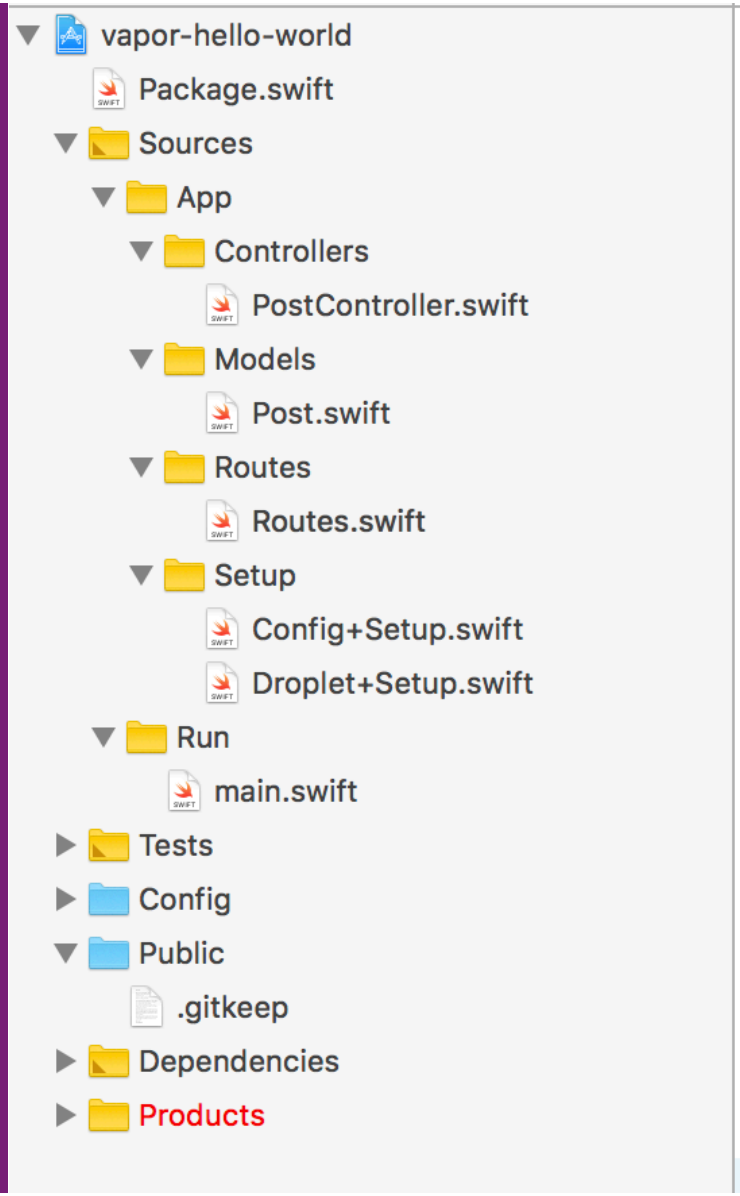
VAPOR

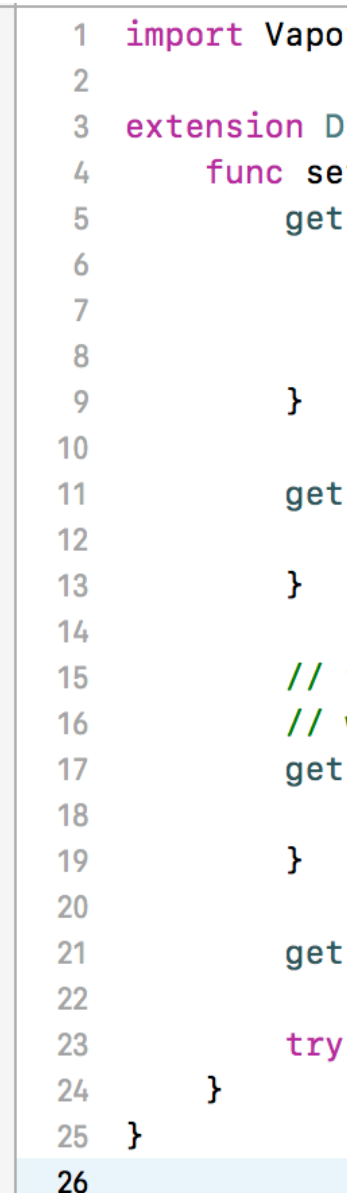
- Server runs on http://localhost:8080



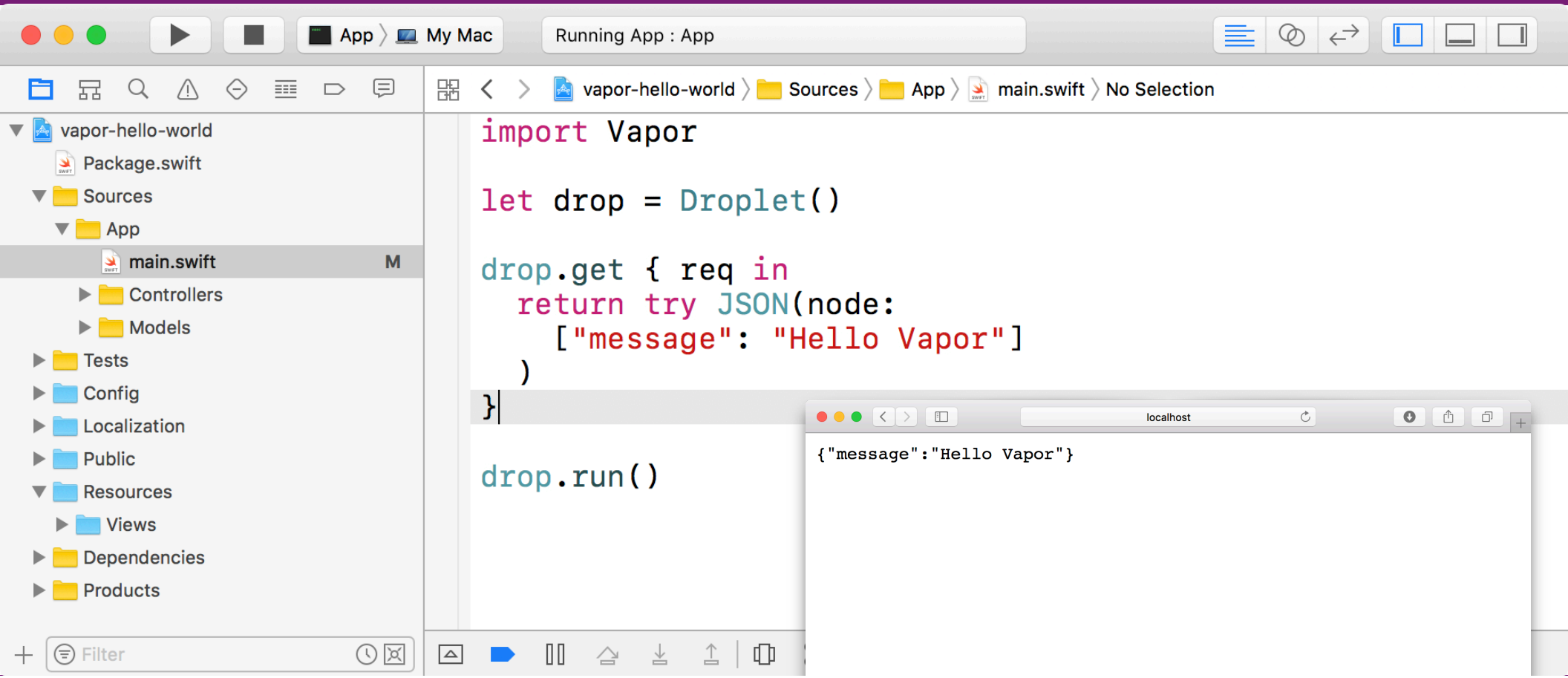
VAPOR

- File structure is designed to make testing easier 🤷‍♀️





VAPOR

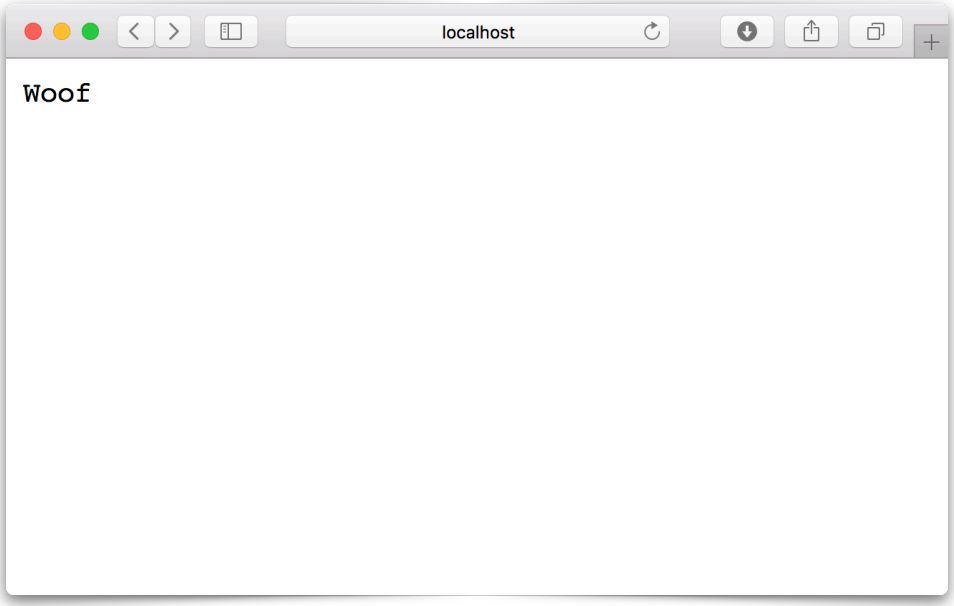

App
My Mac
Running App : App
vapor-hello-world
Sources
App
main.swift
No Selection
vapor-hello-world
Package.swift
Sources
App
main.swift M
Controllers
Models
Tests
Config
Localization
Public
Resources
Views
Dependencies
Products
Filter
import Vapor
let drop = Droplet()
drop.get { req in
return try JSON(node:
["message": "Hello Vapor"]
)
}
drop.run()
localhost
{"message":"Hello Vapor"}

VAPOR

LOCALHOST:8080/DOG

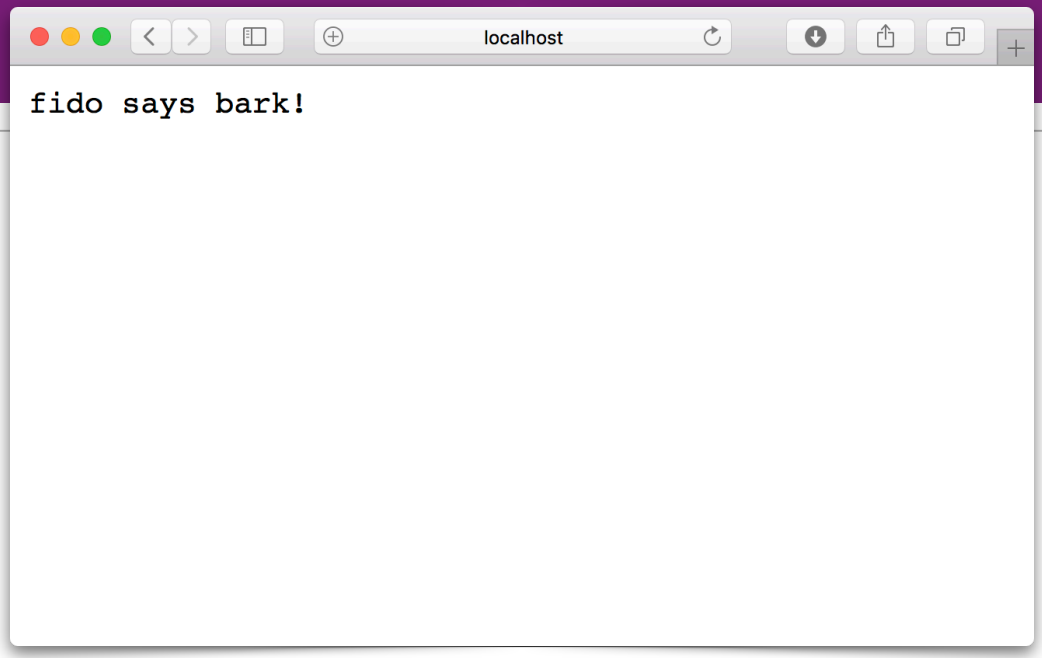
vapor-hello-world 〉 Sources 〉 App 〉 main.swift 〉 No Selection

```
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

LOCALHOST:8080/DOG/SPEAK/FIDO/

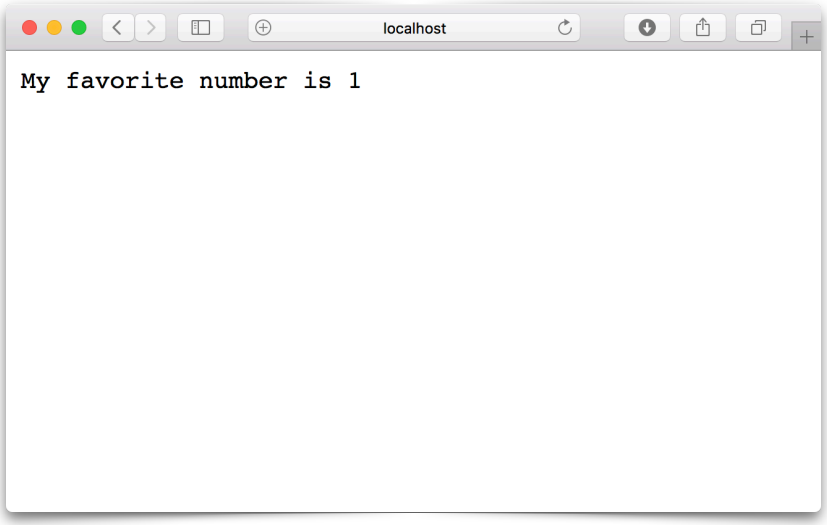


vapor-hello-world / Sources / App / main.swift / No Selection

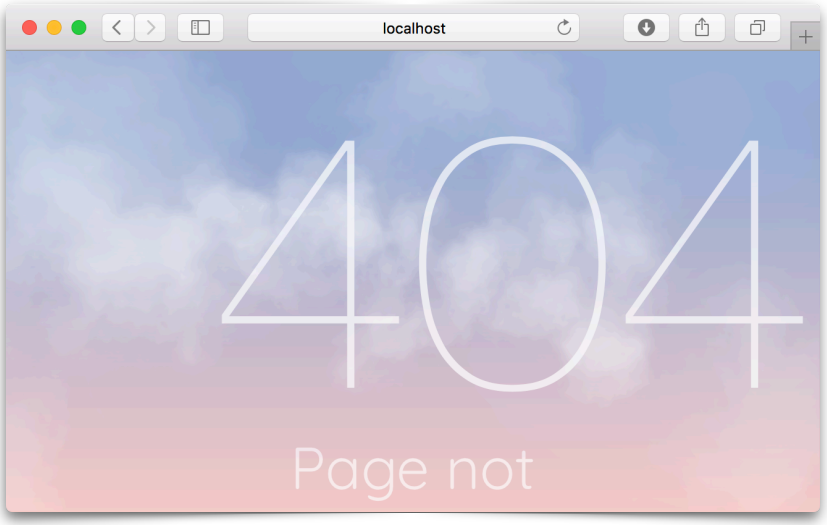
```
drop.get { req in
  return try JSON(node:
    ["message": "Hello Vapor"]
  )
}

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/



LOCALHOST:8080/DOG/SPEAK/FIDO/

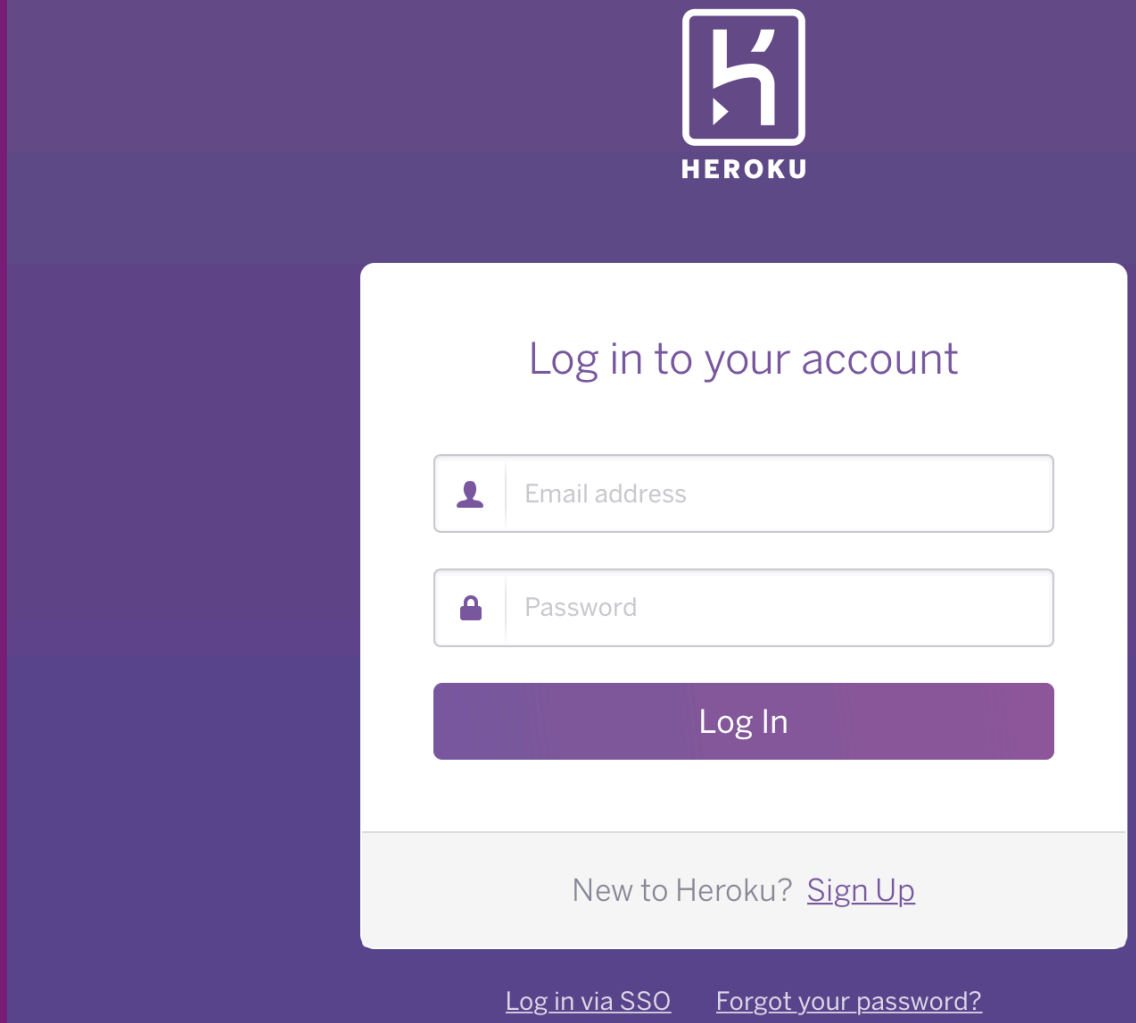



```
drop.get("dog","speak",Int.self) { req, number in
   return "My favorite number is \(number)"
}
```

DEPLOY VAPOR APP TO HEROKU

DEPLOY VAPOR APP TO HEROKU

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



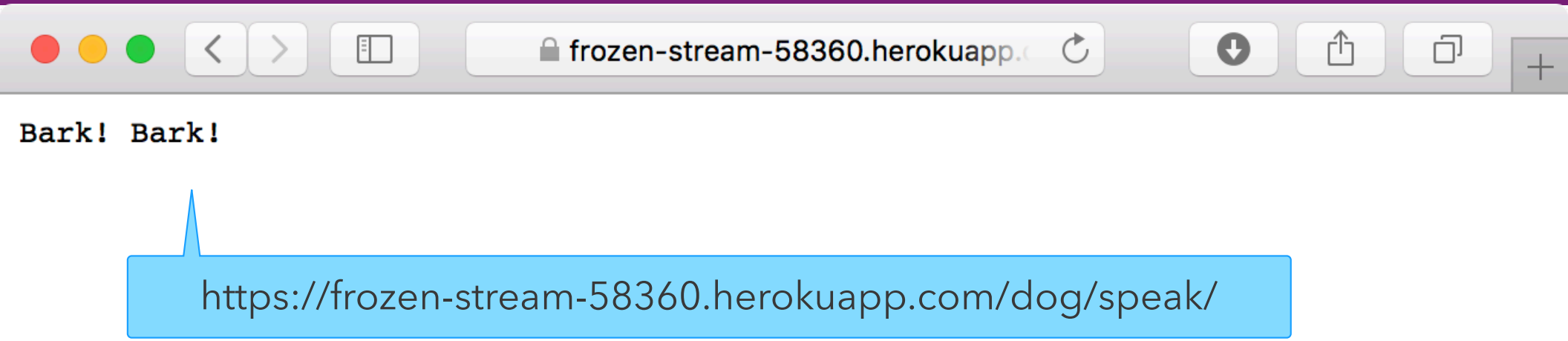
DEPLOY VAPOR APP TO HEROKU

```
# 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
```

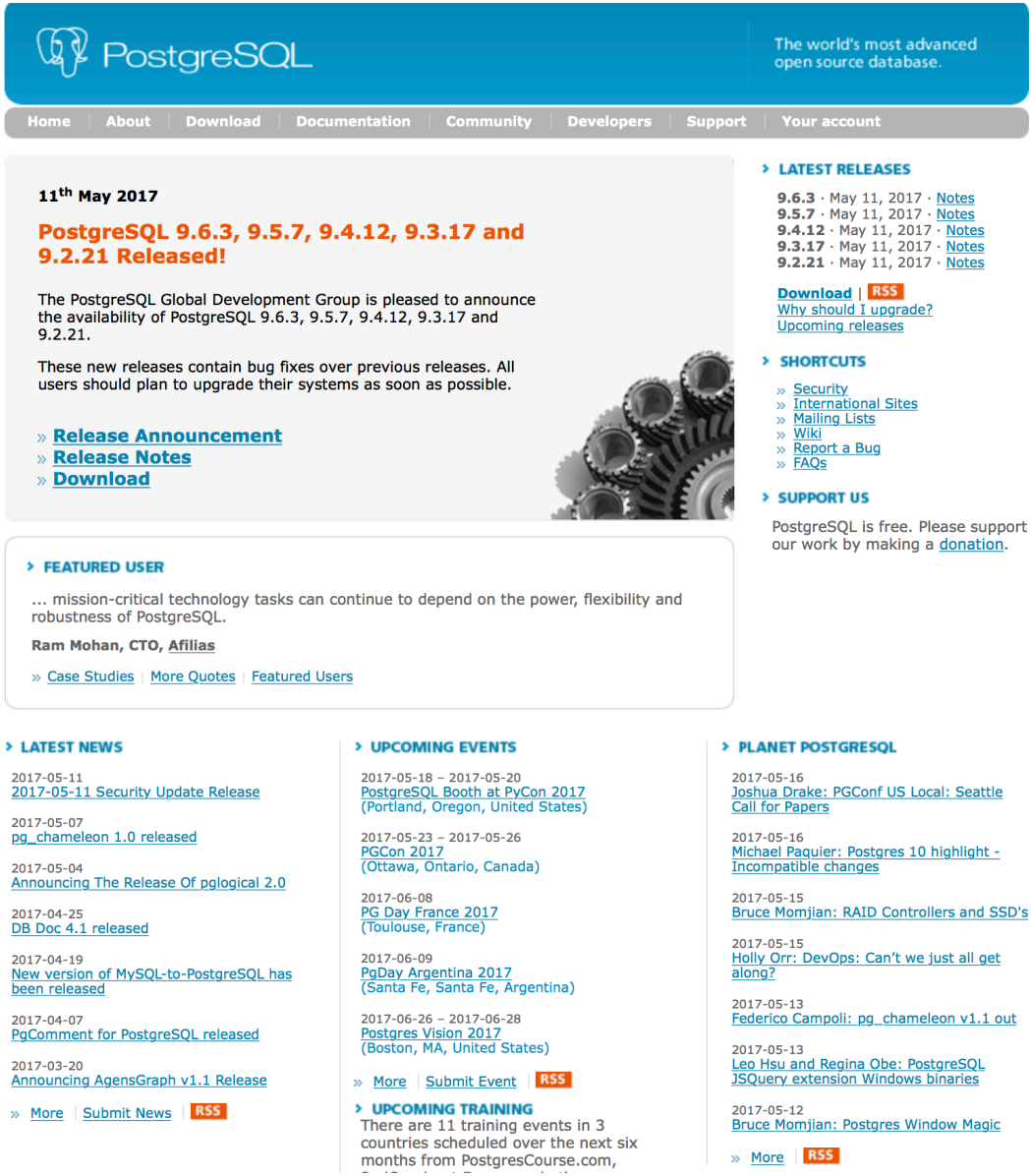
DEPLOY VAPOR APP TO HEROKU



VAPOR WITH POSTGRESQL

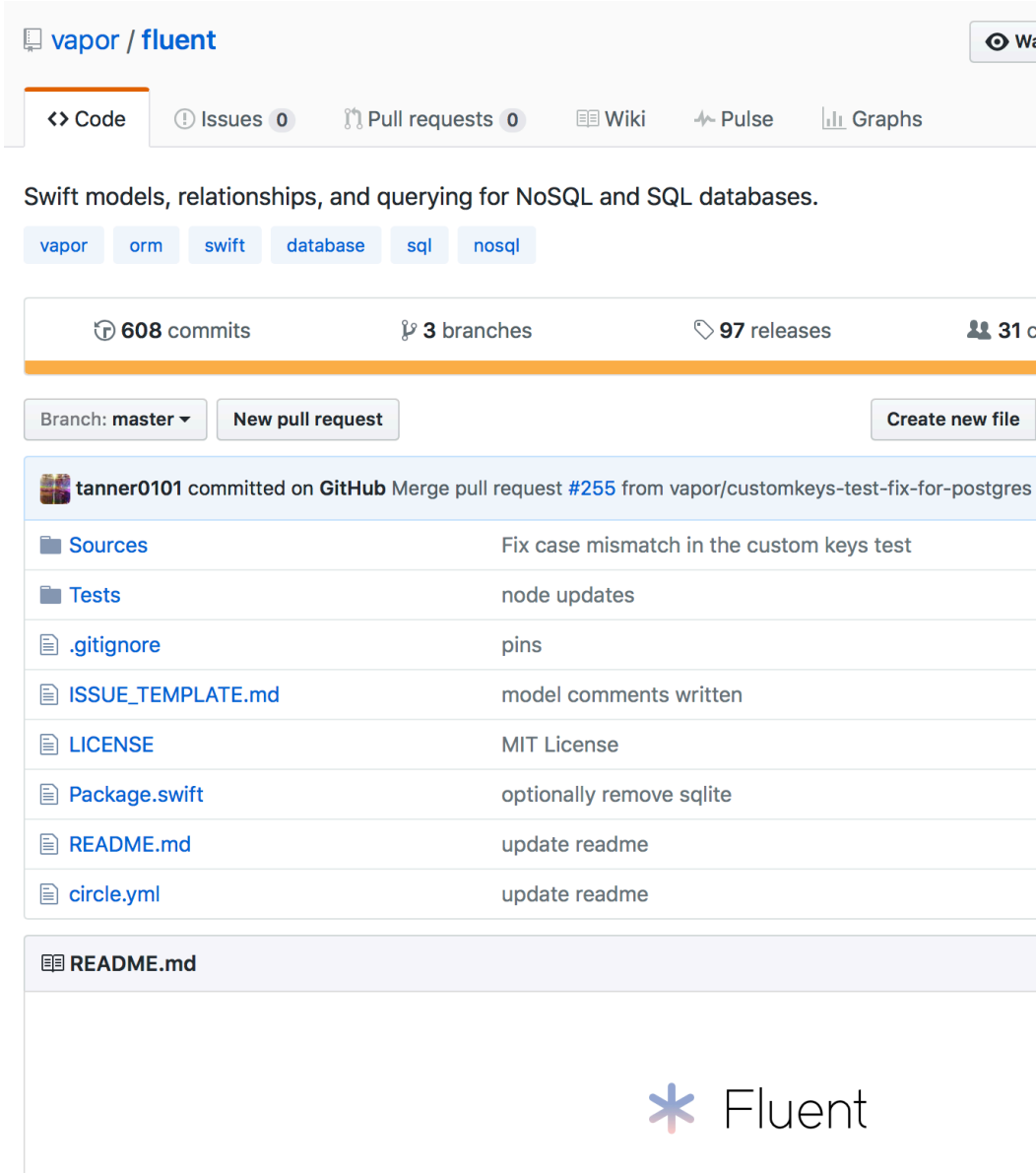
VAPOR WITH POSTGRESQL

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



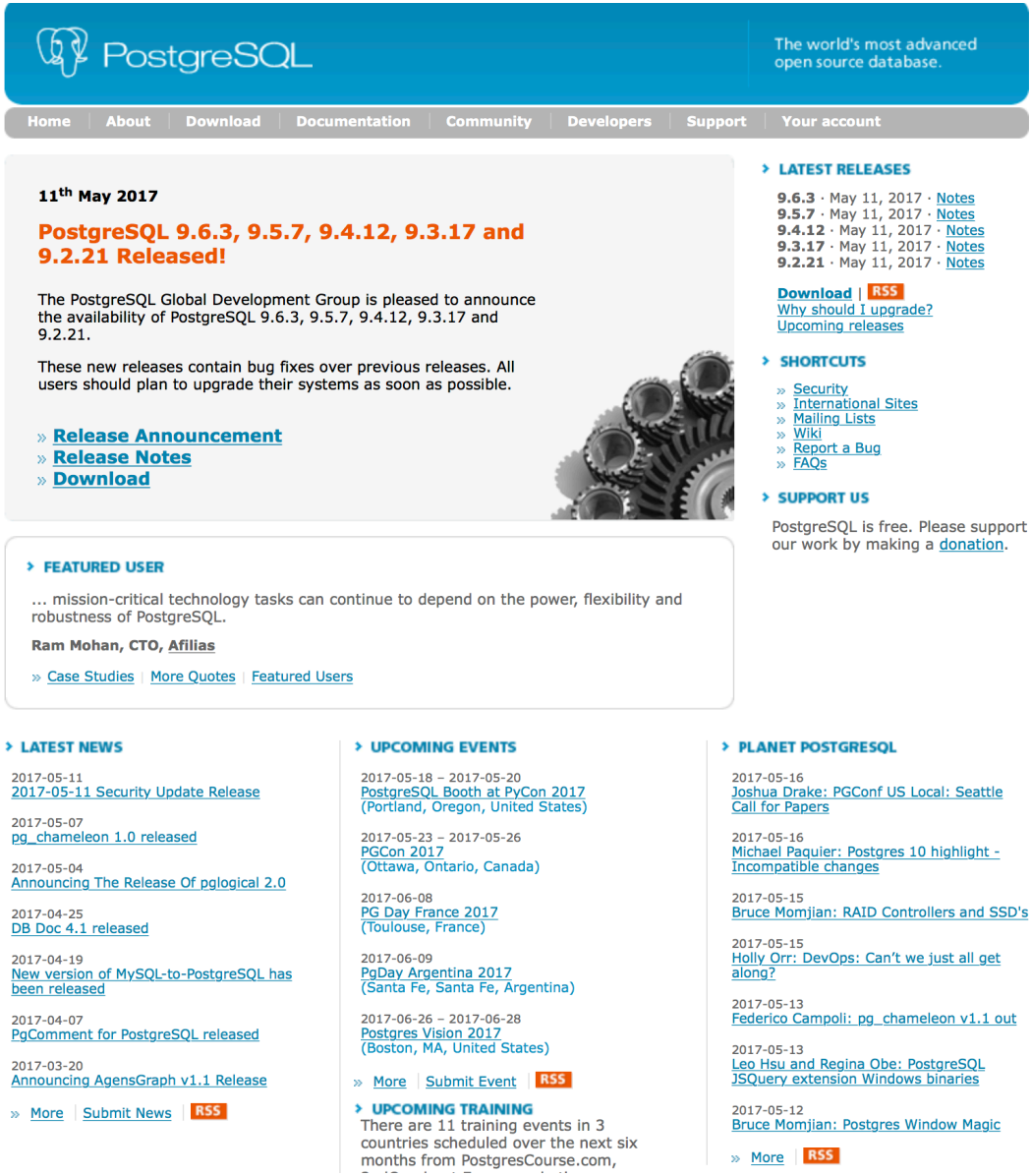
VAPOR WITH POSTGRESQL

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



VAPOR WITH POSTGRESQL

- Import package
- Configure droplet
- Add configuration file



VAPOR WITH POSTGRESQL

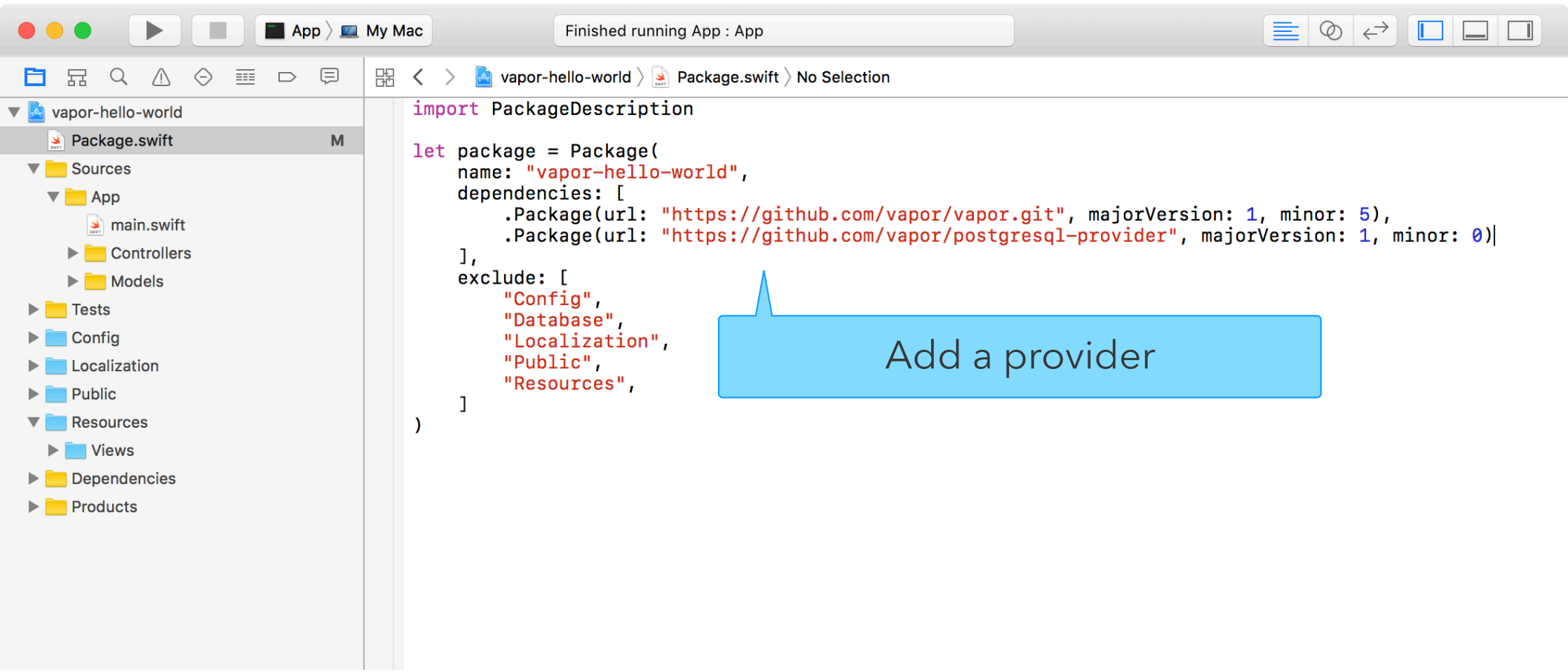
```
# Install postgres
brew install postgres

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

# Create a database
createdb dogs
```

VAPOR WITH POSTGRESQL

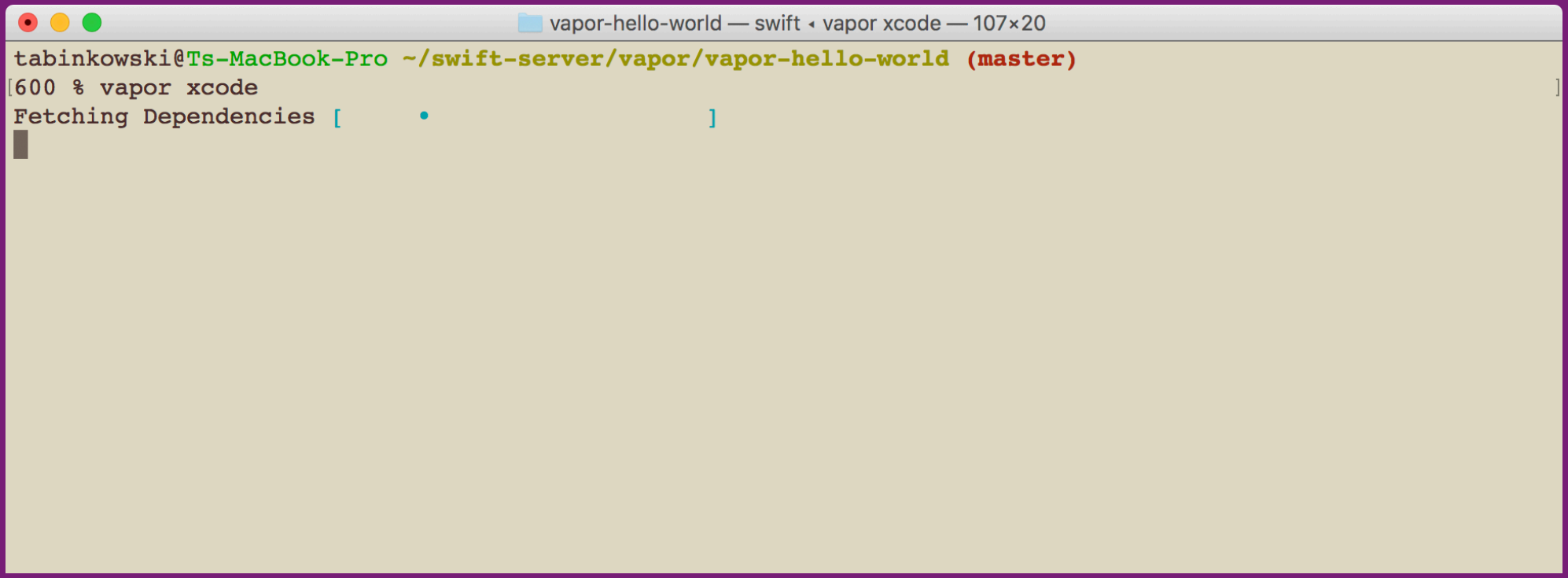
```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[592 % psql
psql (9.6.3)
Type "help" for help.

[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



VAPOR WITH POSTGRESQL



- Update the project and rebuild your framework in Xcode

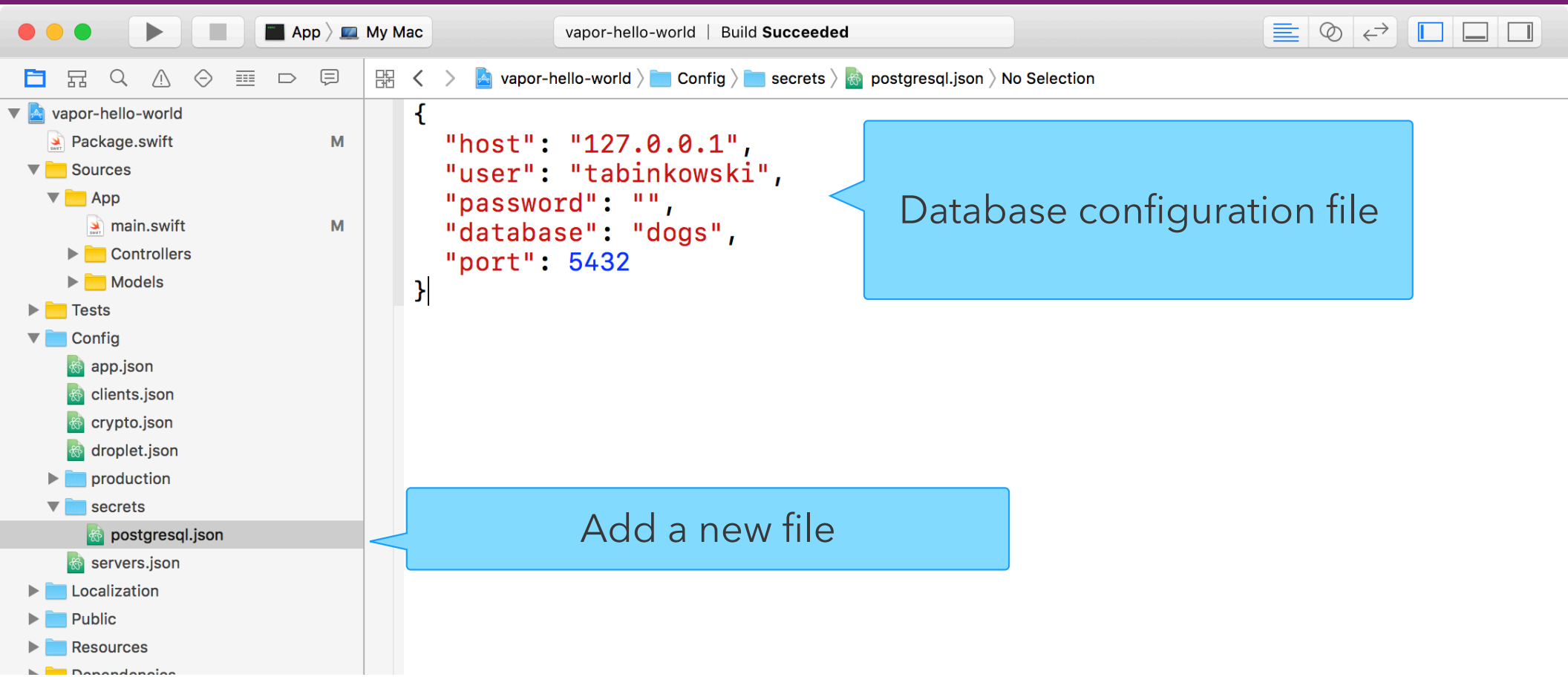
VAPOR WITH POSTGRESQL

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

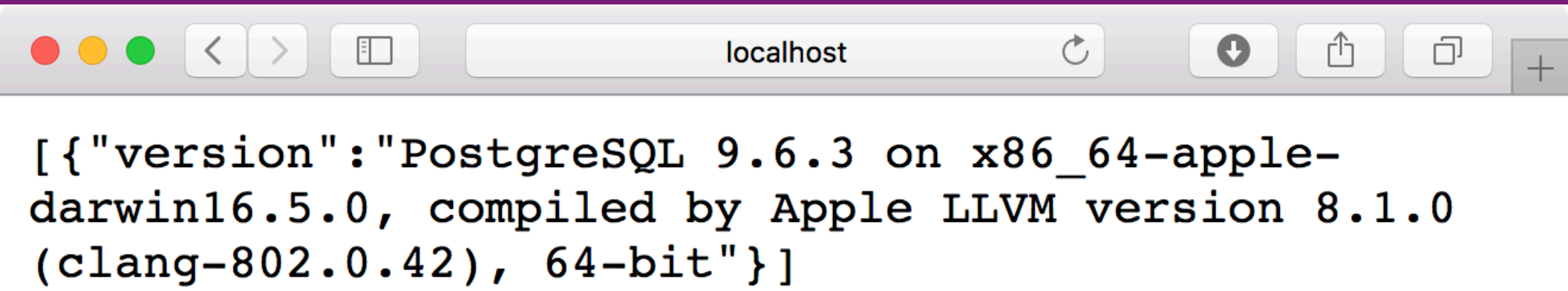


VAPOR WITH POSTGRESQL

```
// 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"
  }
}
```

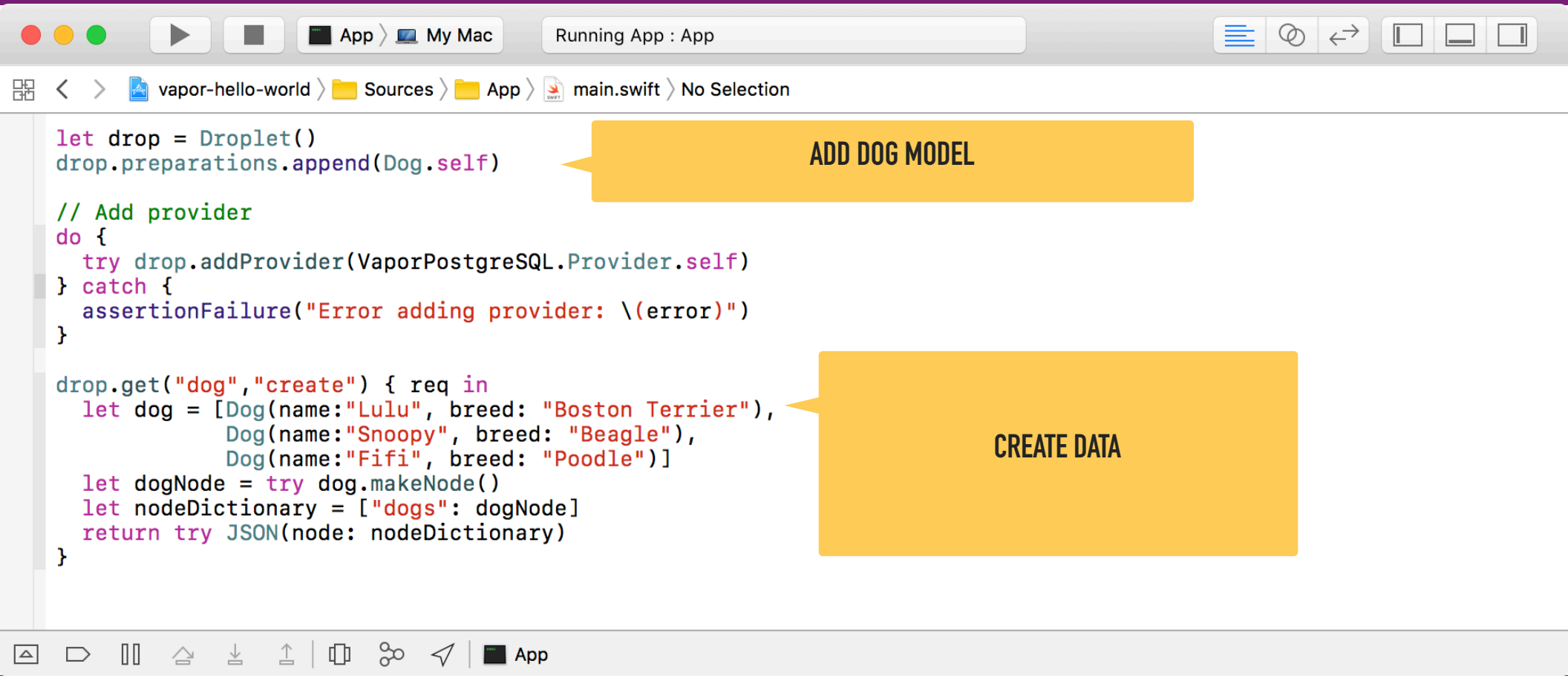
Print out database version

VAPOR WITH POSTGRESQL

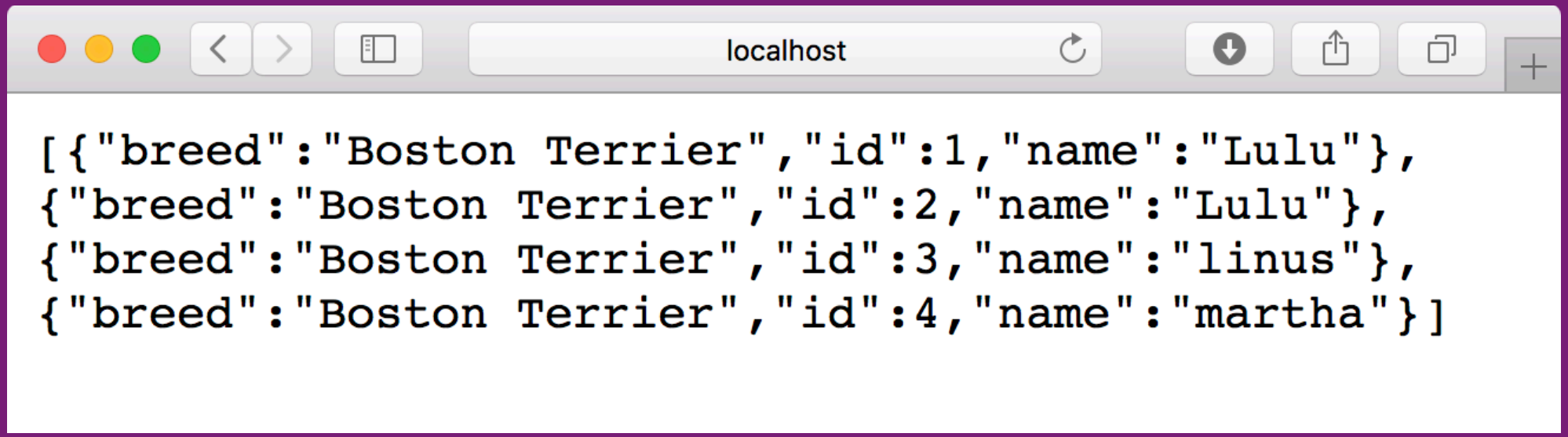


IT'S BORING, BUT IT WORKS

VAPOR WITH POSTGRESQL



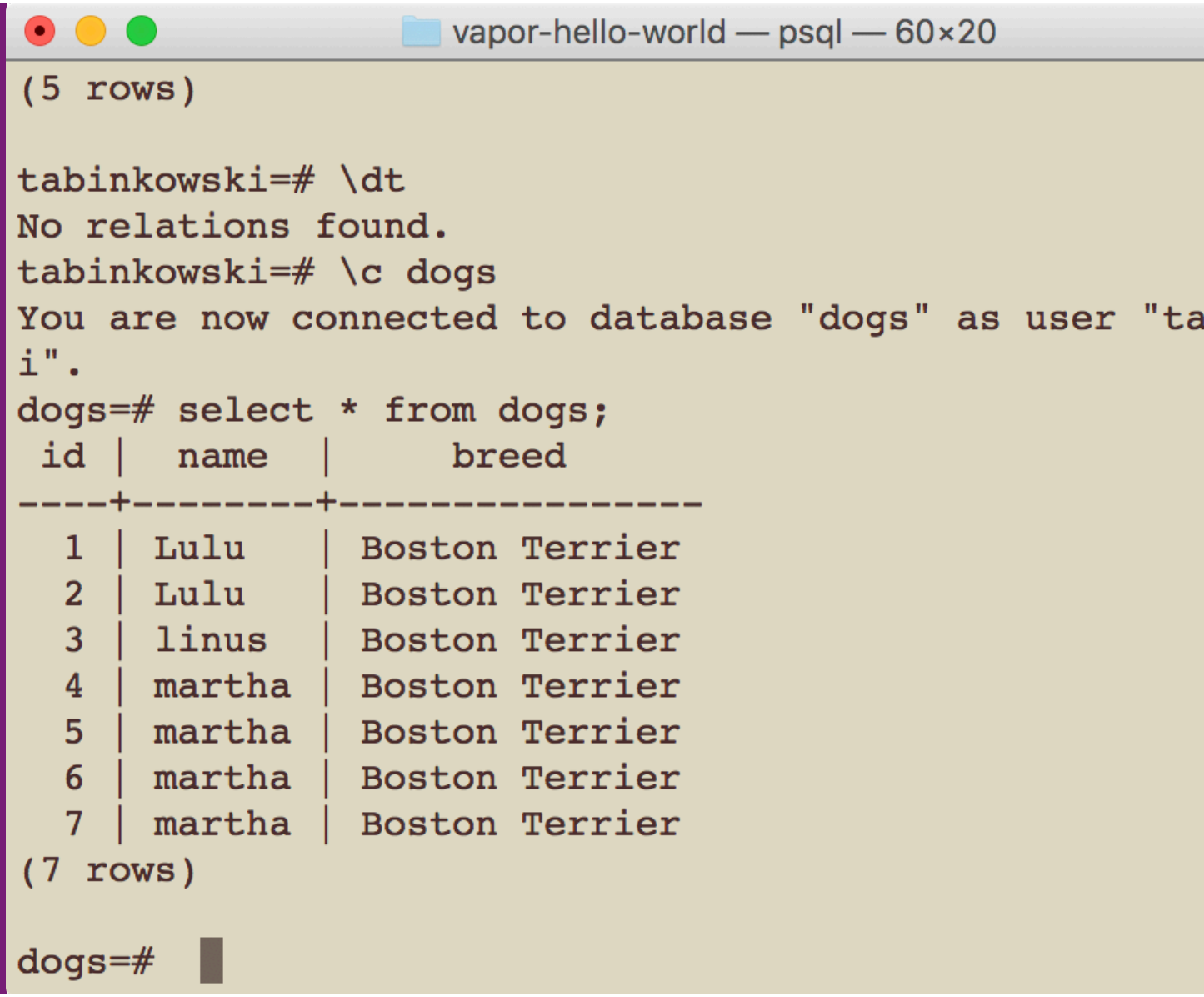
VAPOR WITH POSTGRESQL




```
// 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 dis
  - \c - switch db
  - \d - list tables



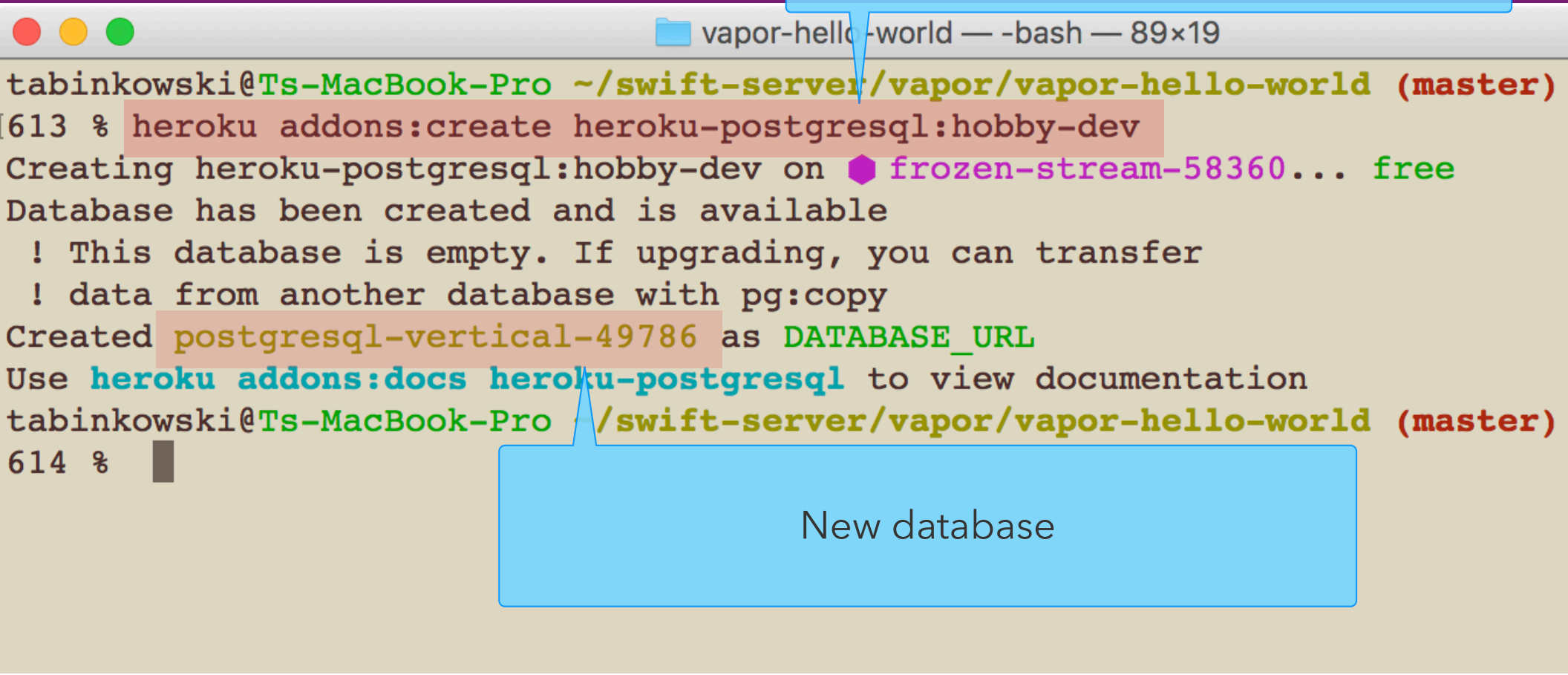
DEPLOY TO HEROKU WITH POSTGRESQL

DEPLOY TO HEROKU WITH POSTGRESQL

- Set up postgresql on heroku
  - Built-in, just need to add it
- Tell Postgresql where the database is locates
  - 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

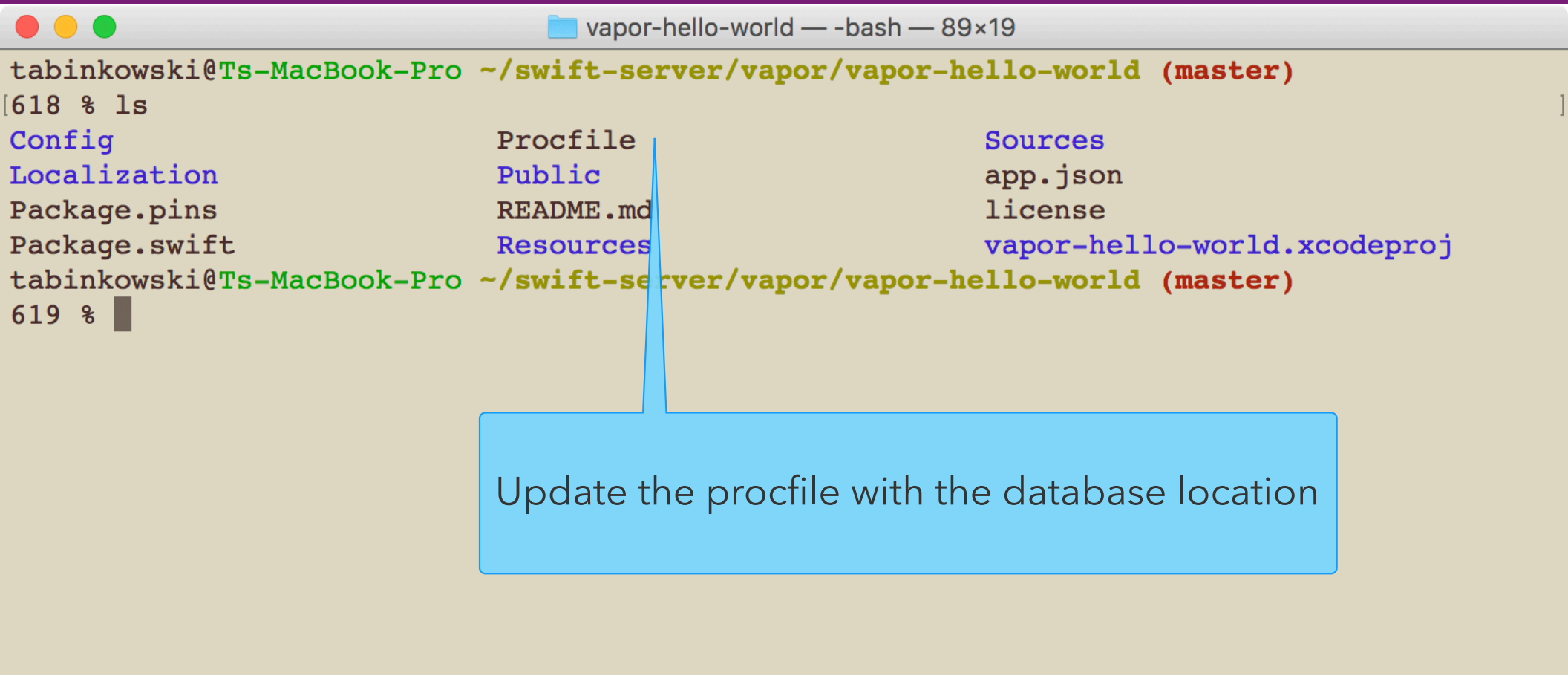


DEPLOY TO HEROKU WITH POSTGRESQL

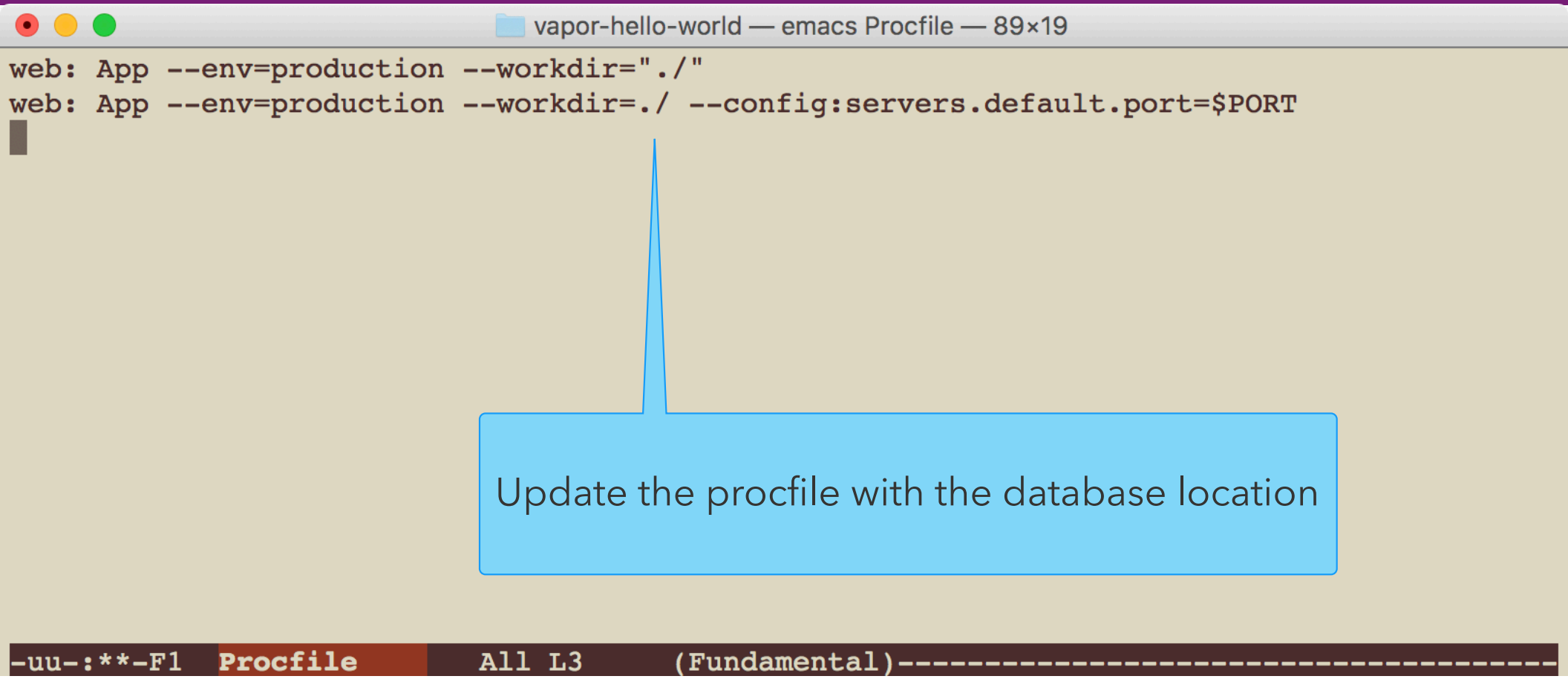
```
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
[613 % heroku addons:create heroku-postgresql:hobby-dev
Creating heroku-postgresql:hobby-dev on ⬢ frozen-stream-58360... free
Database has been created and is available
 ! This database is empty. If upgrading, you can transfer
 ! 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/d9hapt1pu1fgrf
tabinkowski@Ts-MacBook-Pro ~/swift-server/vapor/vapor-hello-world (master)
615 %
```

Database location

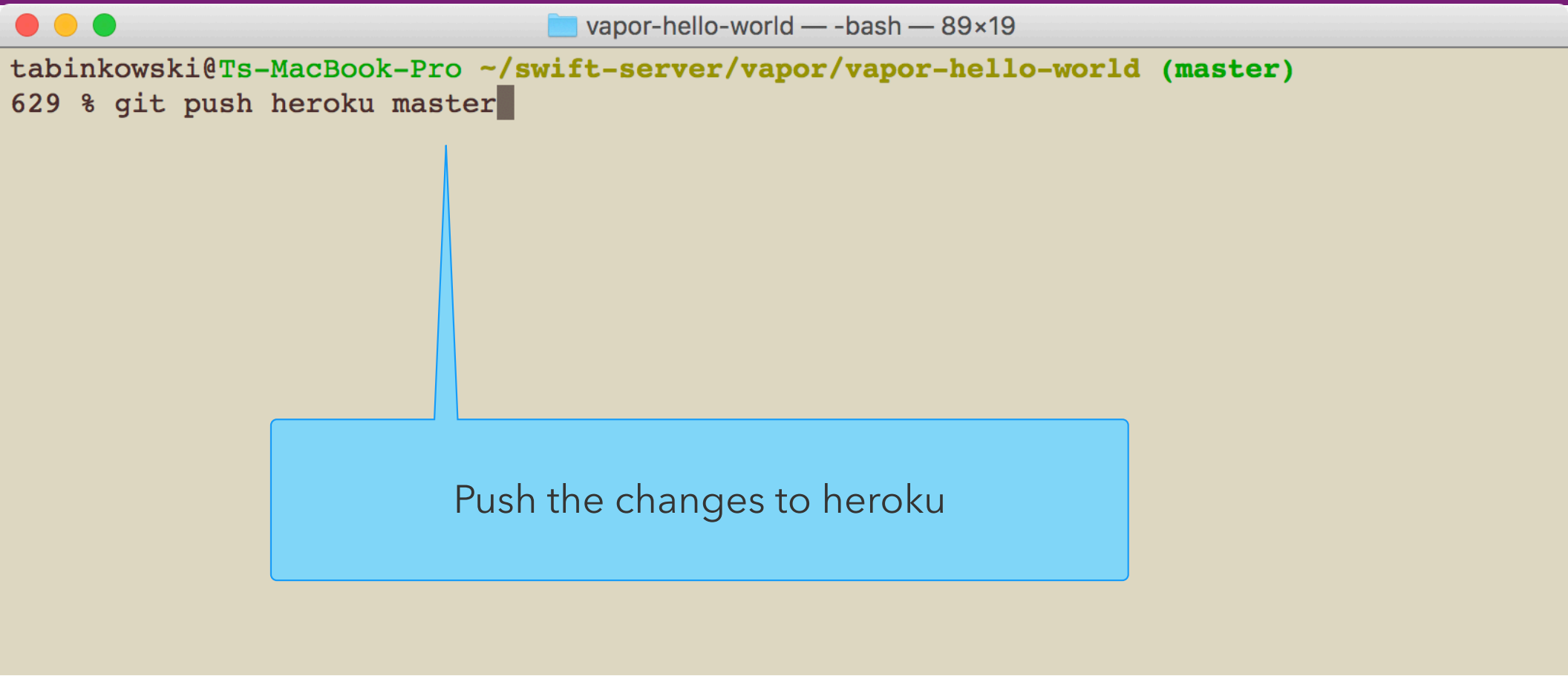
DEPLOY TO HEROKU WITH POSTGRESQL



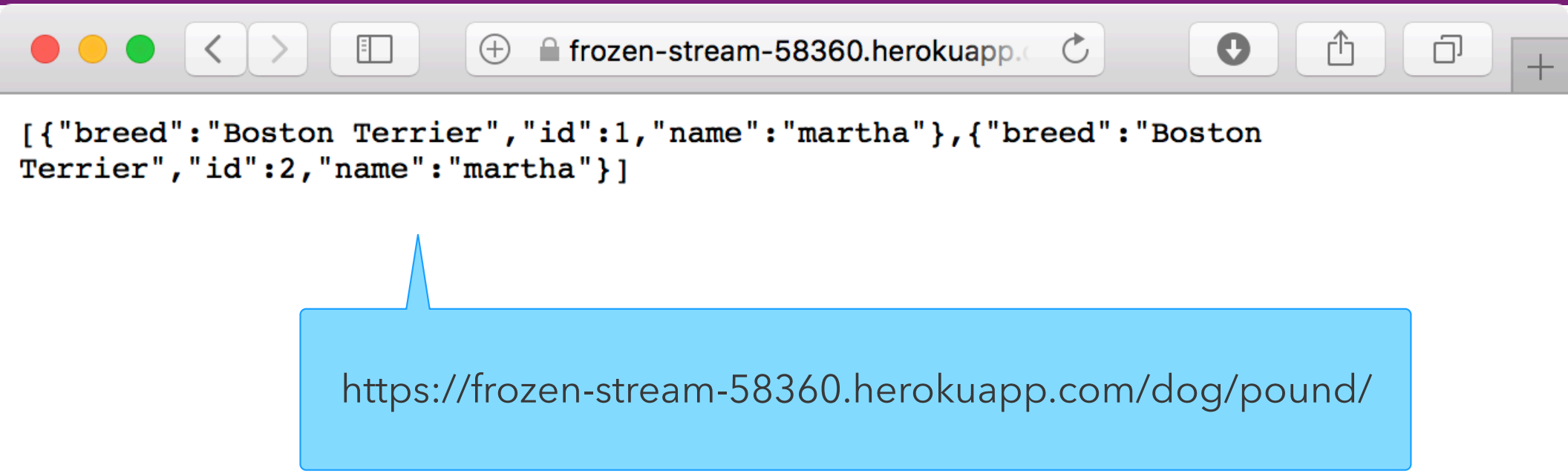
DEPLOY TO HEROKU WITH POSTGRESQL



DEPLOY TO HEROKU WITH POSTGRESQL



DEPLOY TO HEROKU WITH POSTGRESQL



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

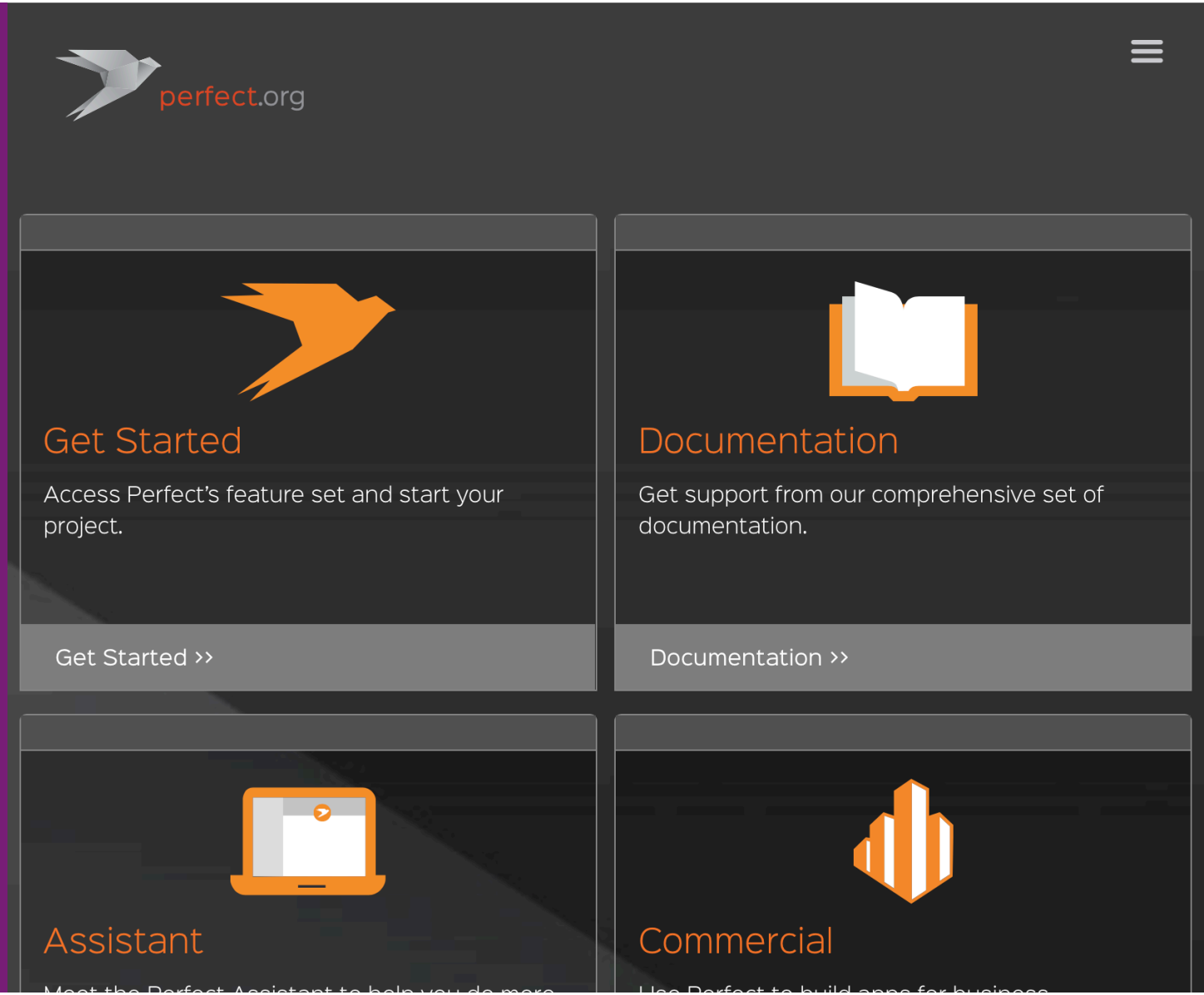
Change values

Save

PERFECT

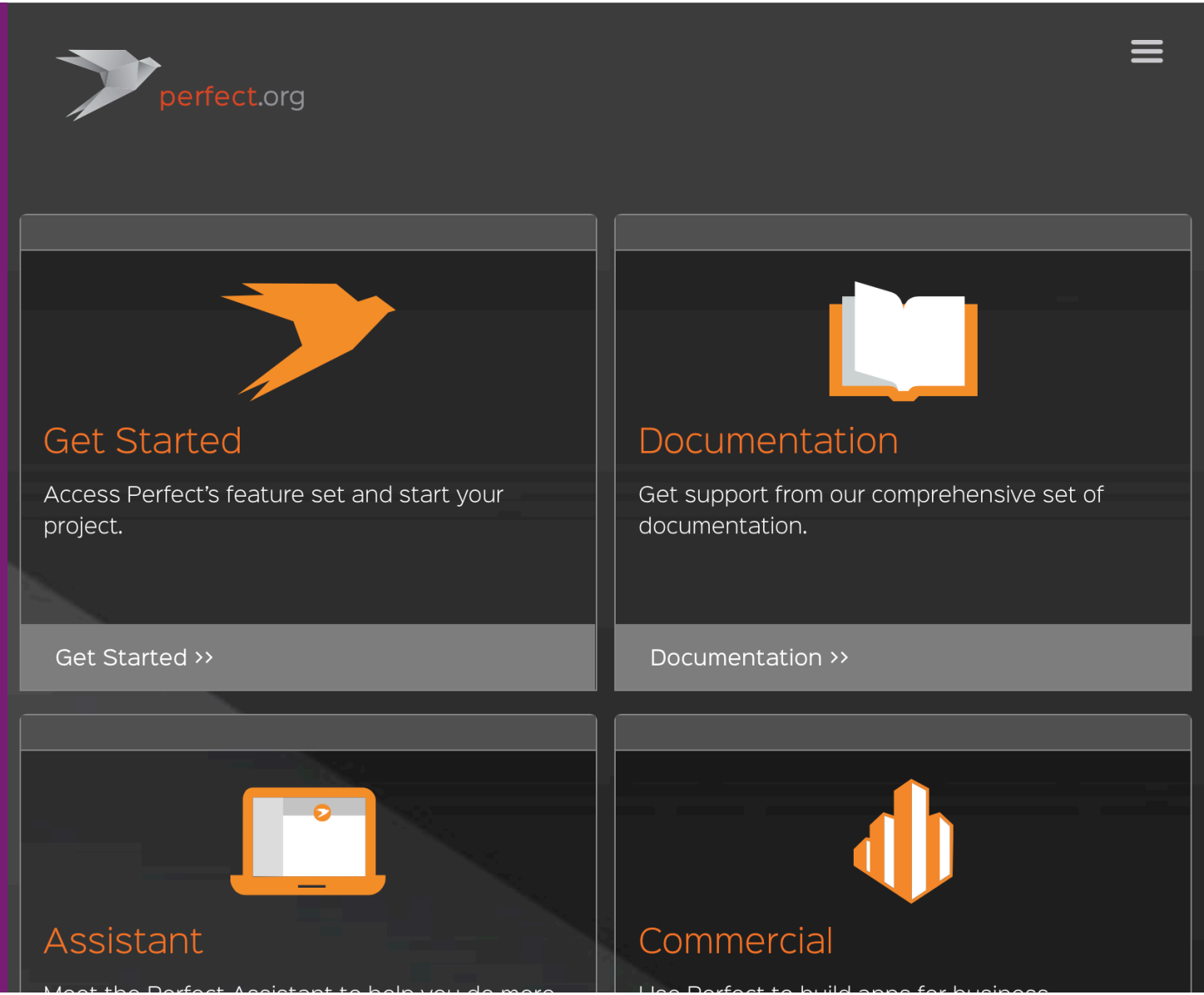
PERFECT

- Web framework and server for Swift server applications



PERFECT

- Requirements
  - Xcode 9 Mac
  - Linux toolchain

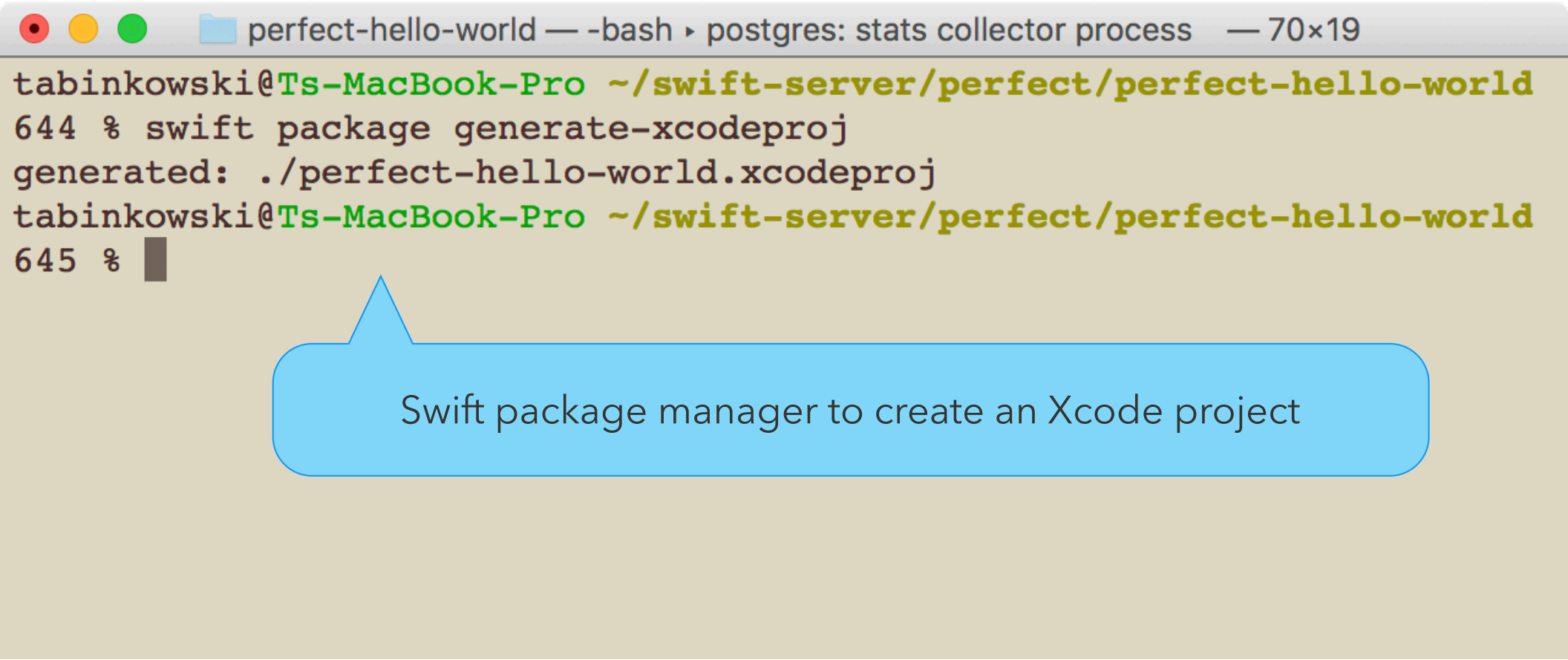


PERFECT

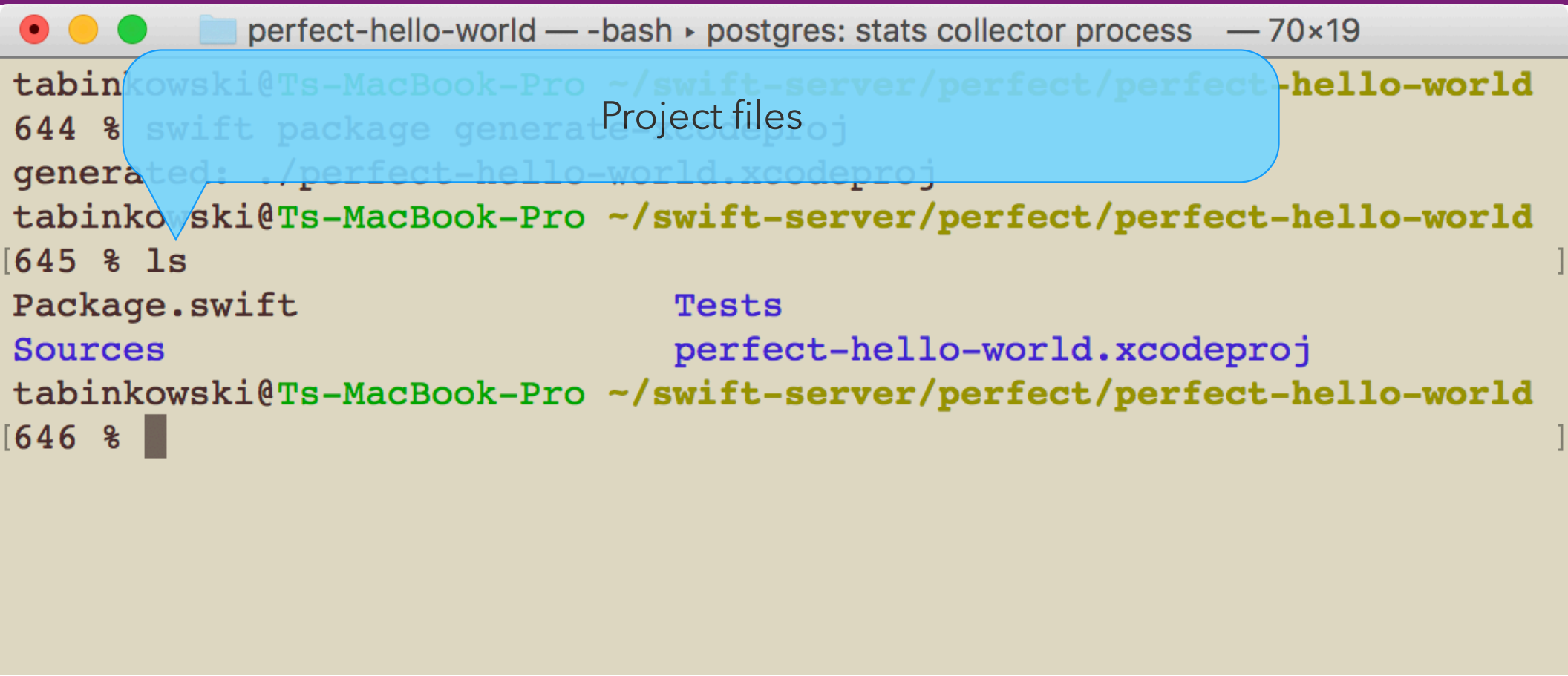
```
[637 % cd perfect/
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect
[638 % ls
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect
[639 % mkdir perfect-hello-world
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect
[640 % cd perfect-hello-world/
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
[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 %
```

Swift package manager to create an executable package

PERFECT

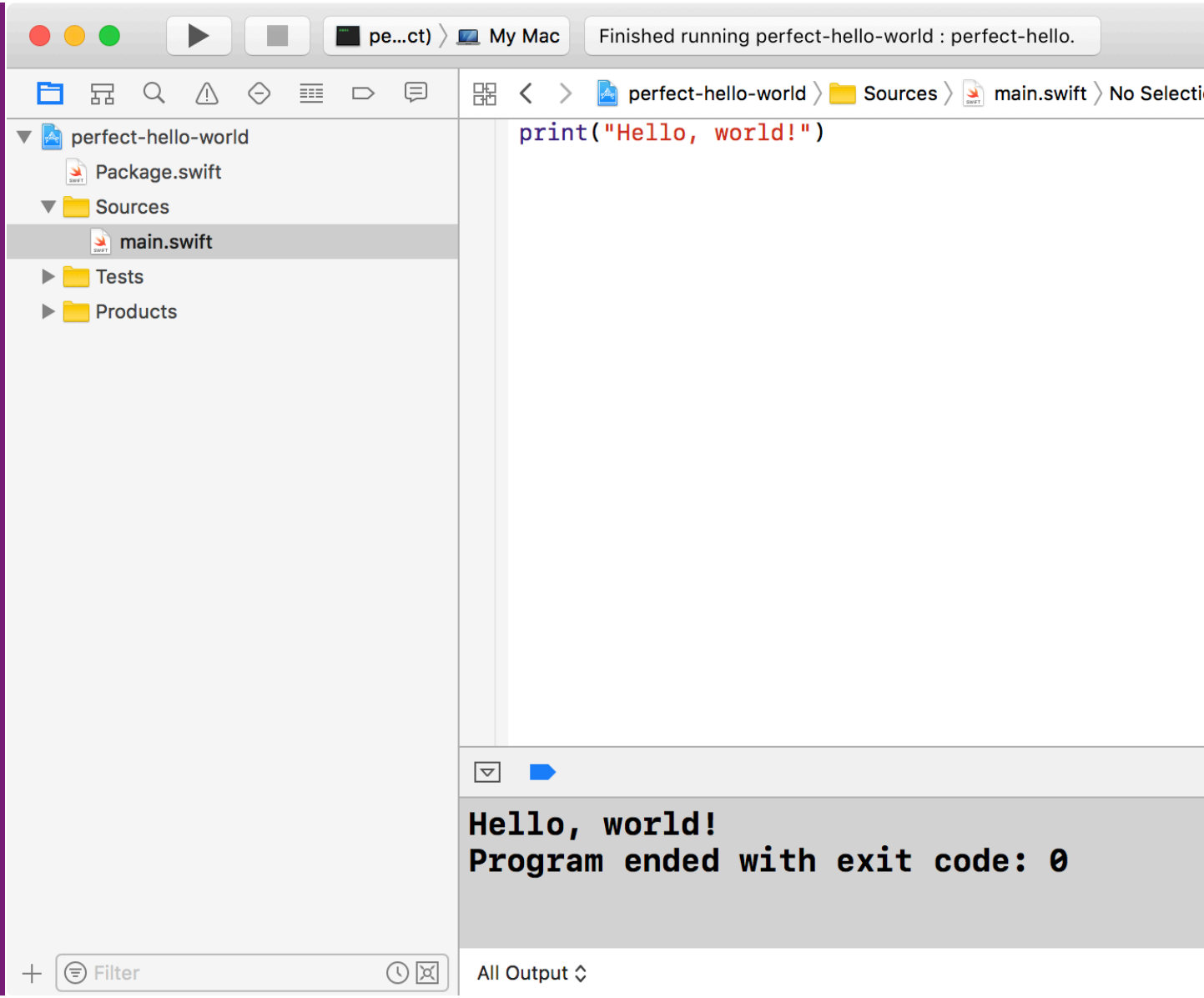

perfect-hello-world — -bash ▸ postgres: stats collector process — 70×19
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
644 % swift package generate-xcodeproj
generated: ./perfect-hello-world.xcodeproj
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
645 %
Swift package manager to create an Xcode project

PERFECT

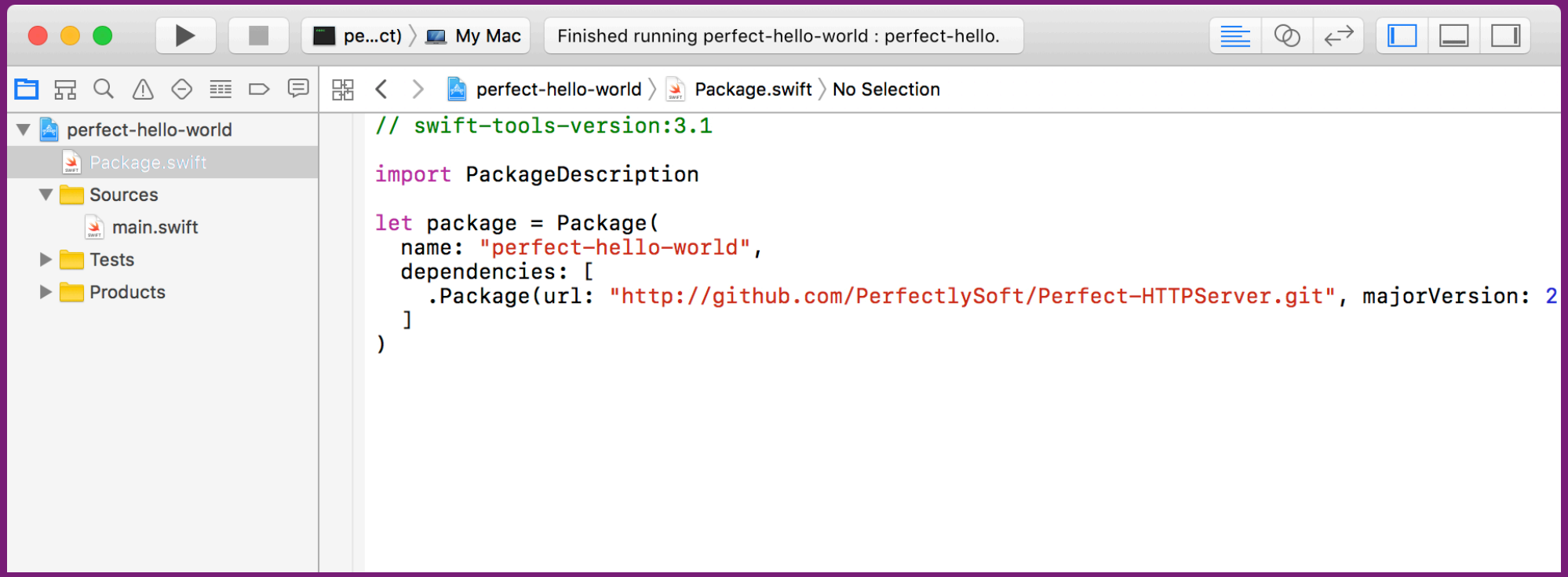

perfect-hello-world — -bash ▸ postgres: stats collector process — 70×19
Project files
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
644 % swift package generate-xcodeproj
generated: ./perfect-hello-world.xcodeproj
tabinkowski@Ts-MacBook-Pro ~/swift-server/perfect/perfect-hello-world
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

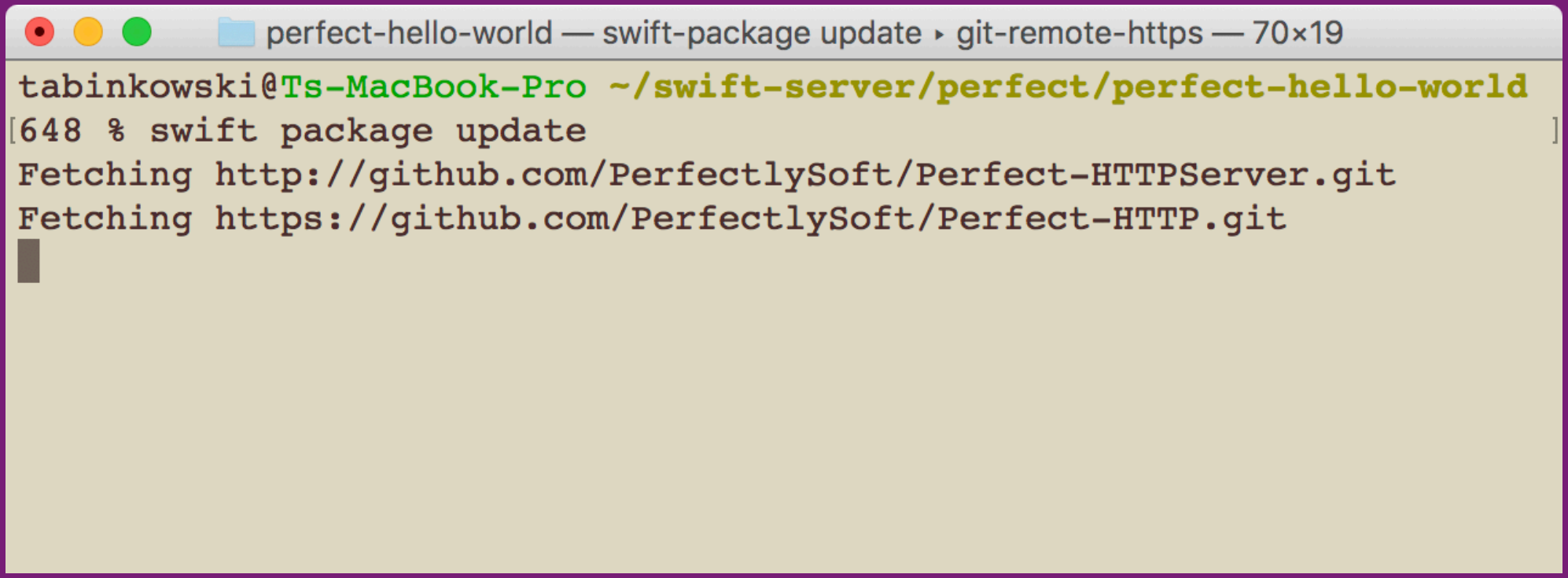


PERFECT



- Add Perfect as a project dependency

PERFECT

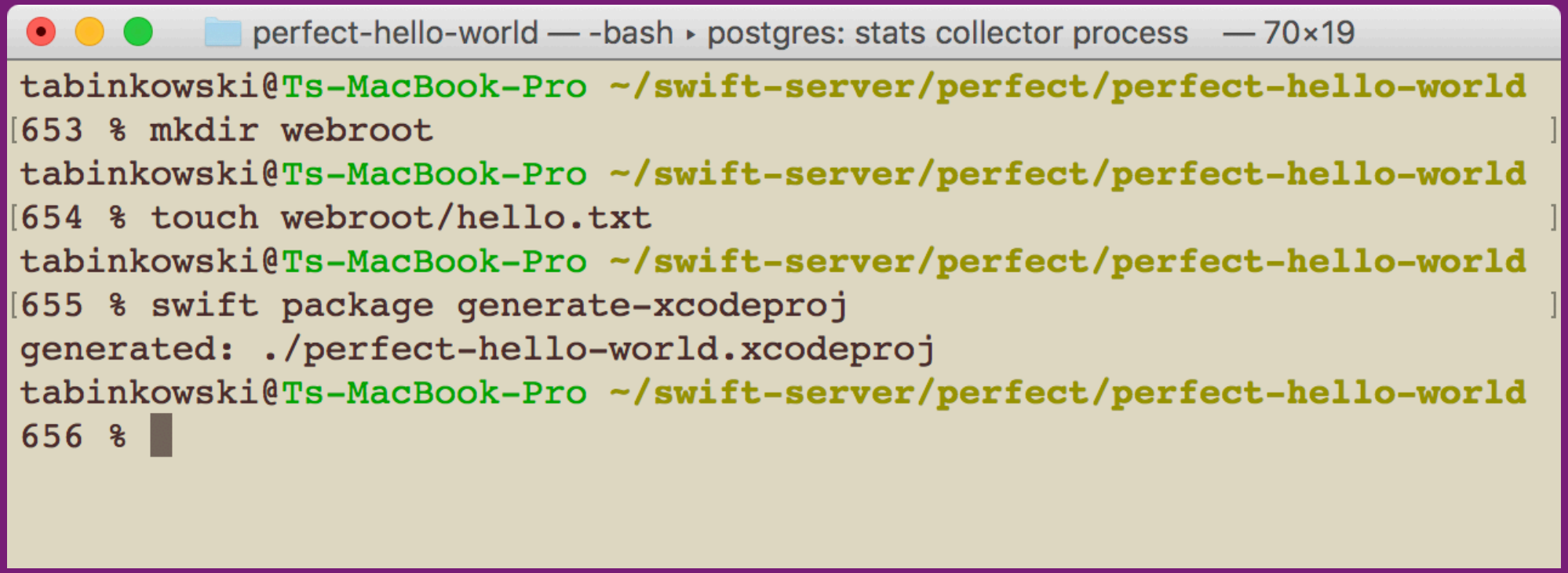


- Update the package manager

PERFECT

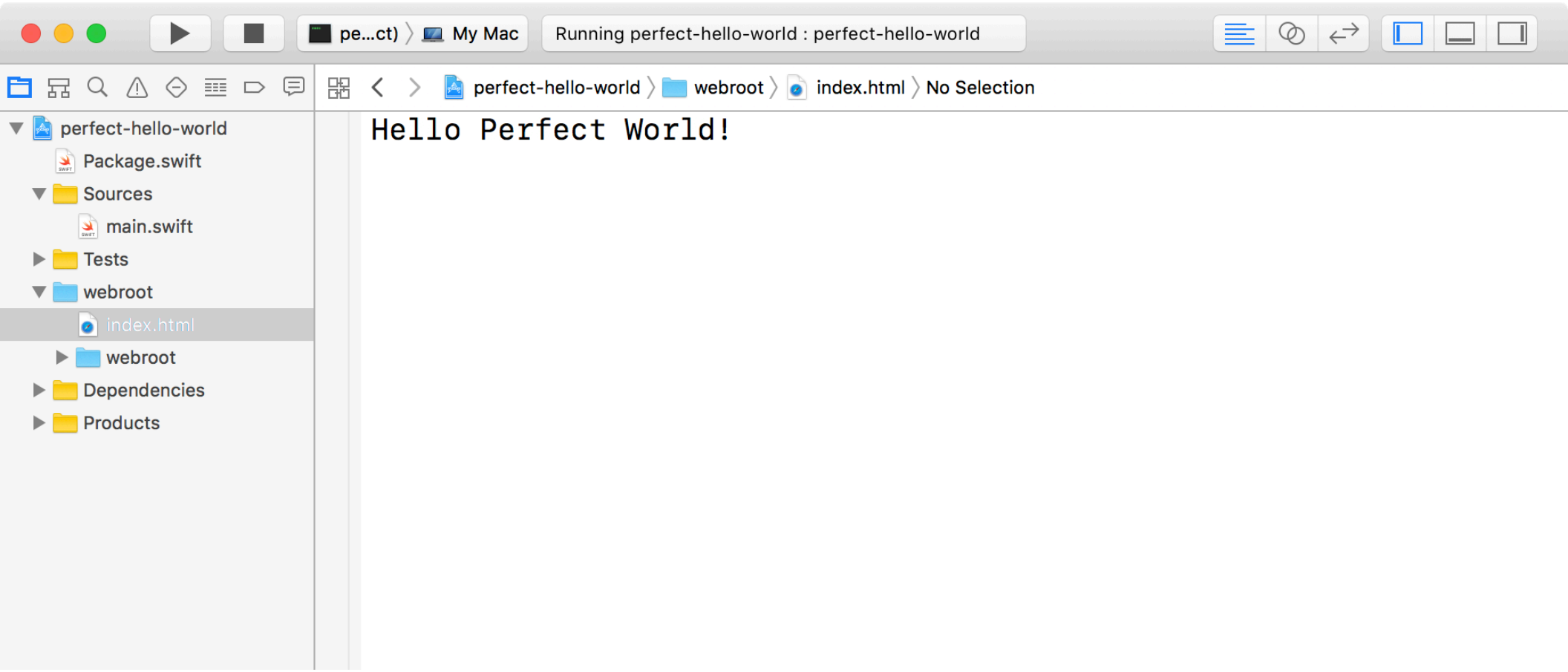
- Update the Xcode project

PERFECT



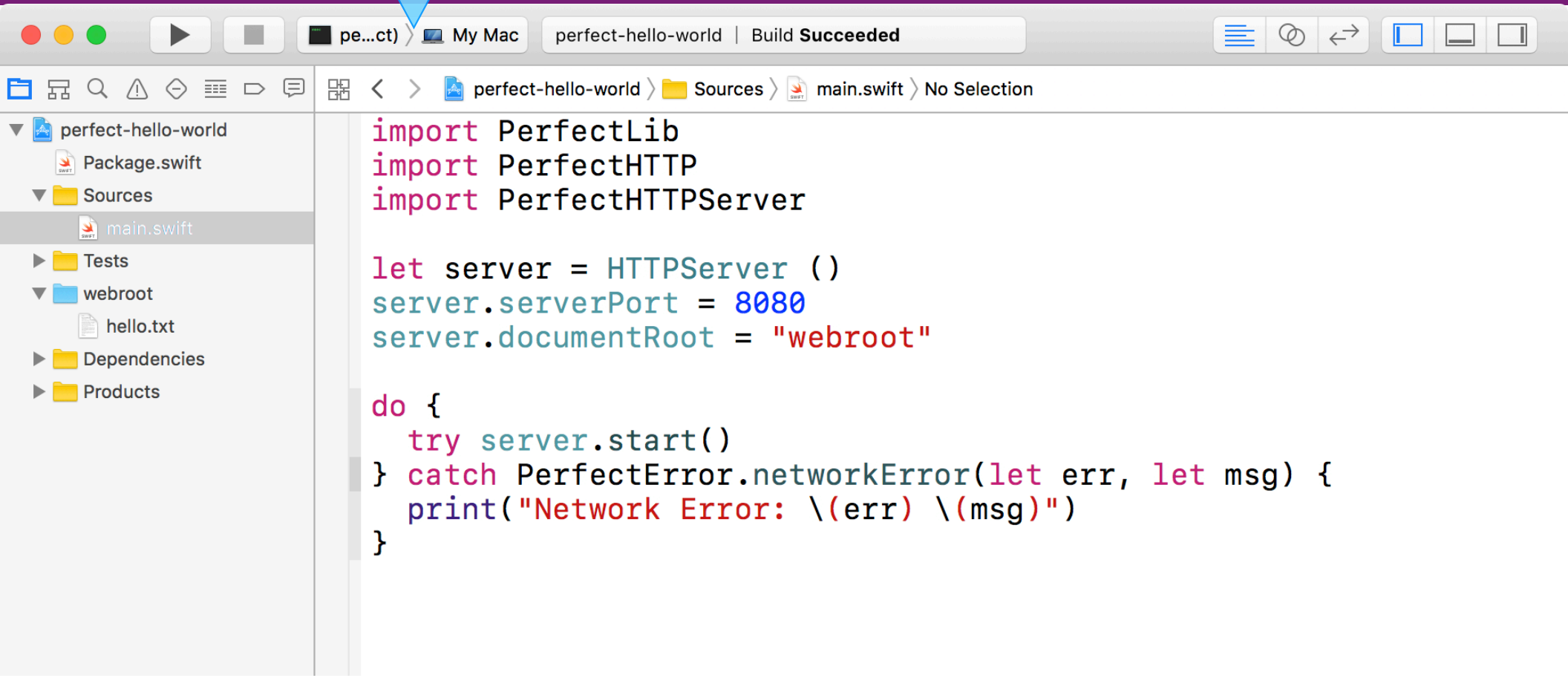
- Create a root directory for the web server

PERFECT

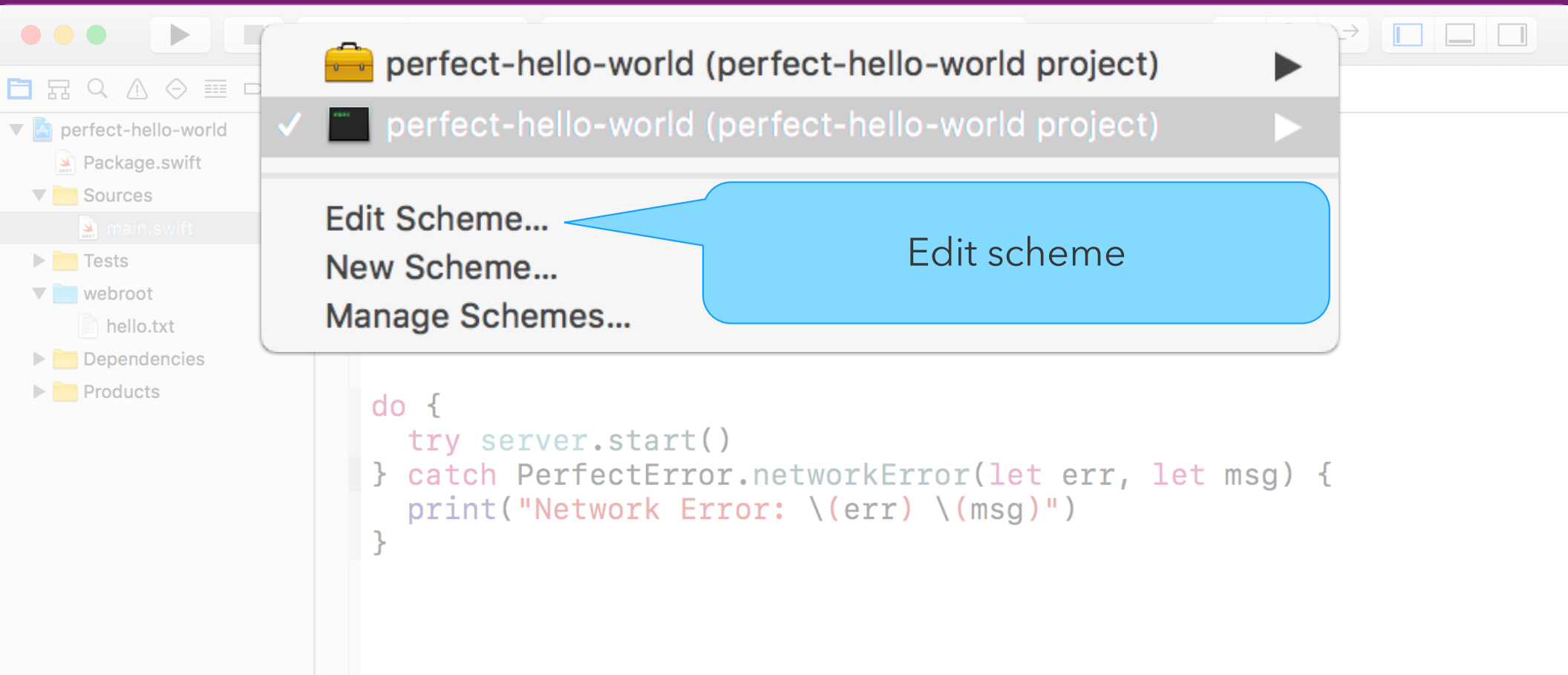

pe...ct) 〉 My Mac
Running perfect-hello-world : perfect-hello-world
perfect-hello-world 〉 webroot 〉 index.html 〉 No Selection
perfect-hello-world
Package.swift
Sources
main.swift
Tests
webroot
index.html
webroot
Dependencies
Products
Hello Perfect World!

PERFECT

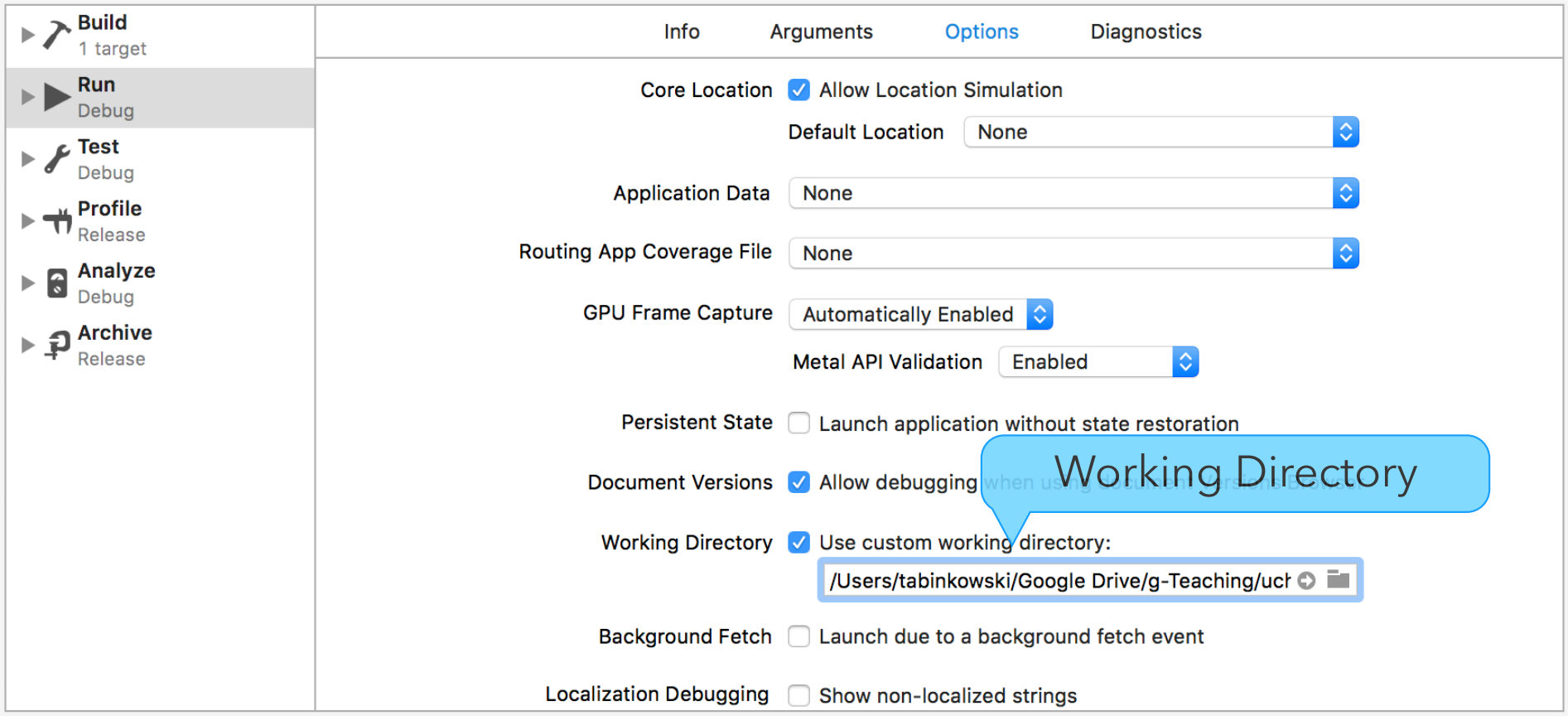
Create a server



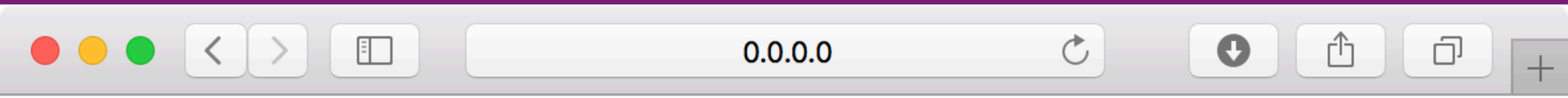
PERFECT


perfect-hello-world (perfect-hello-world project)
perfect-hello-world (perfect-hello-world project)
Edit Scheme...
New Scheme...
Manage Schemes...
Edit scheme
perfect-hello-world
Package.swift
Sources
main.swift
Tests
webroot
hello.txt
Dependencies
Products
do {
try server.start()
} catch PerfectError.networkError(let err, let msg) {
print("Network Error: \(err) \(msg)")
}

PERFECT

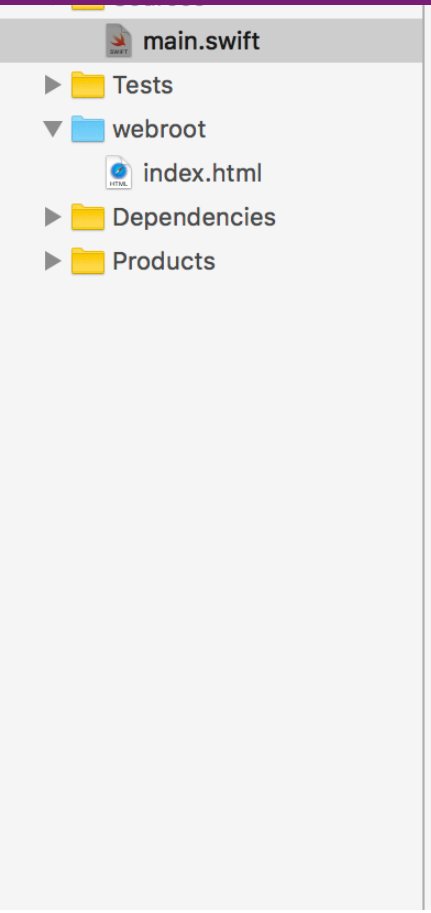

Options
Build
1 target
Run
Debug
Test
Debug
Profile
Release
Analyze
Debug
Archive
Release
Info
Arguments
Options
Diagnostics
Core Location
Allow Location Simulation
Default Location
None
Application Data
None
Routing App Coverage File
None
GPU Frame Capture
Automatically Enabled
Metal API Validation
Enabled
Persistent State
Launch application without state restoration
Document Versions
Allow debugging
Working Directory
Working Directory
Use custom working directory:
/Users/tabinkowski/Google Drive/g-Teaching/uch
Background Fetch
Launch due to a background fetch event
Localization Debugging
Show non-localized strings

PERFECT



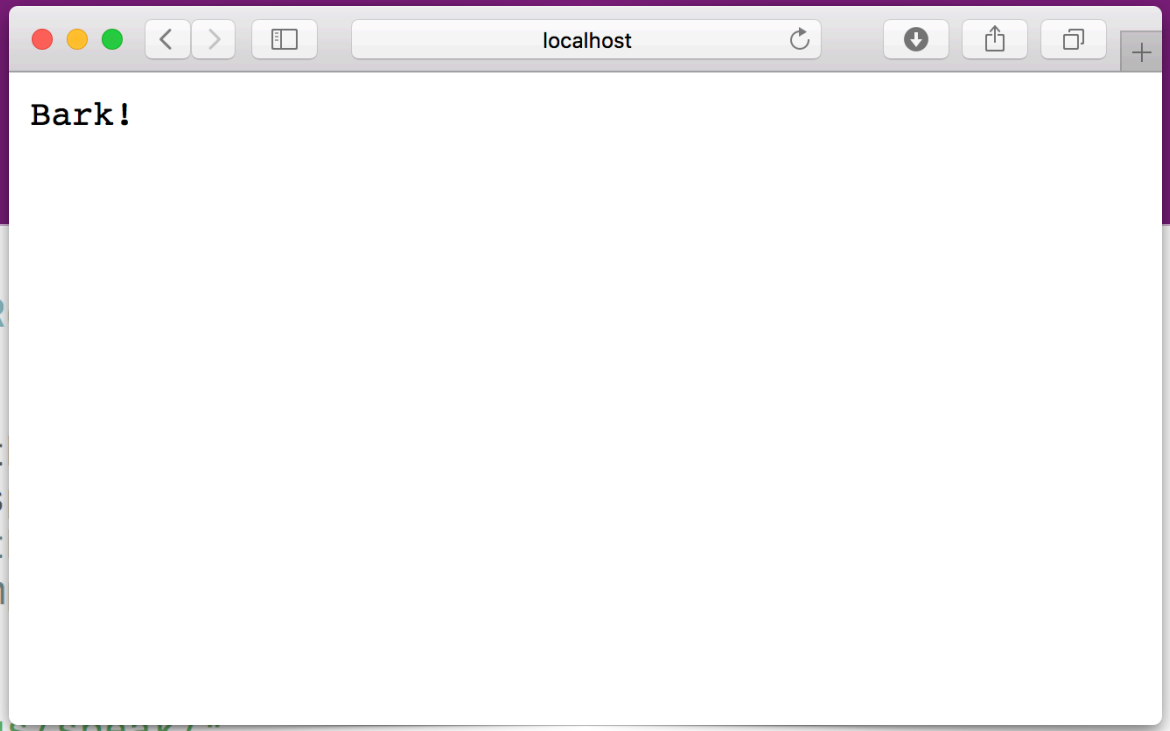
Hello Perfect World!

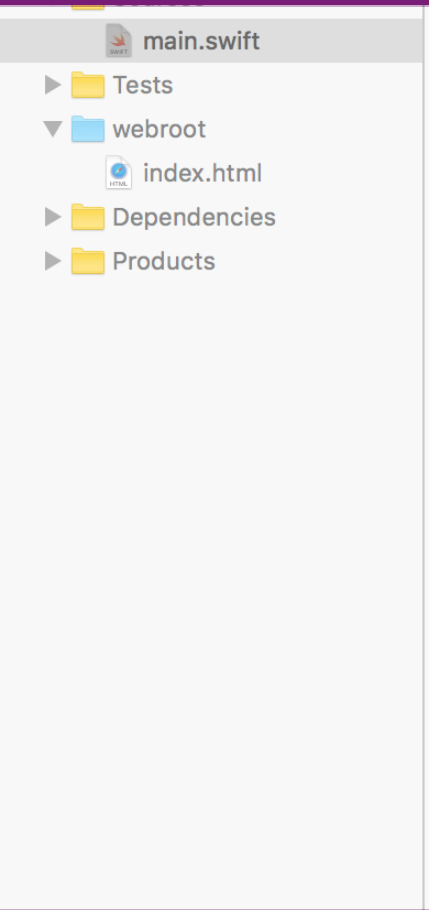
PERFECT




```
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



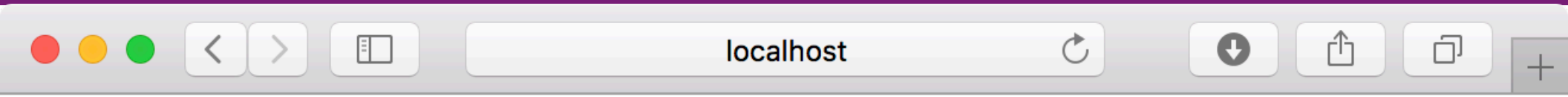



```
var routes = R

// Route "/"
routes.add(met
  request, res
  response.set
  response.com
})

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

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



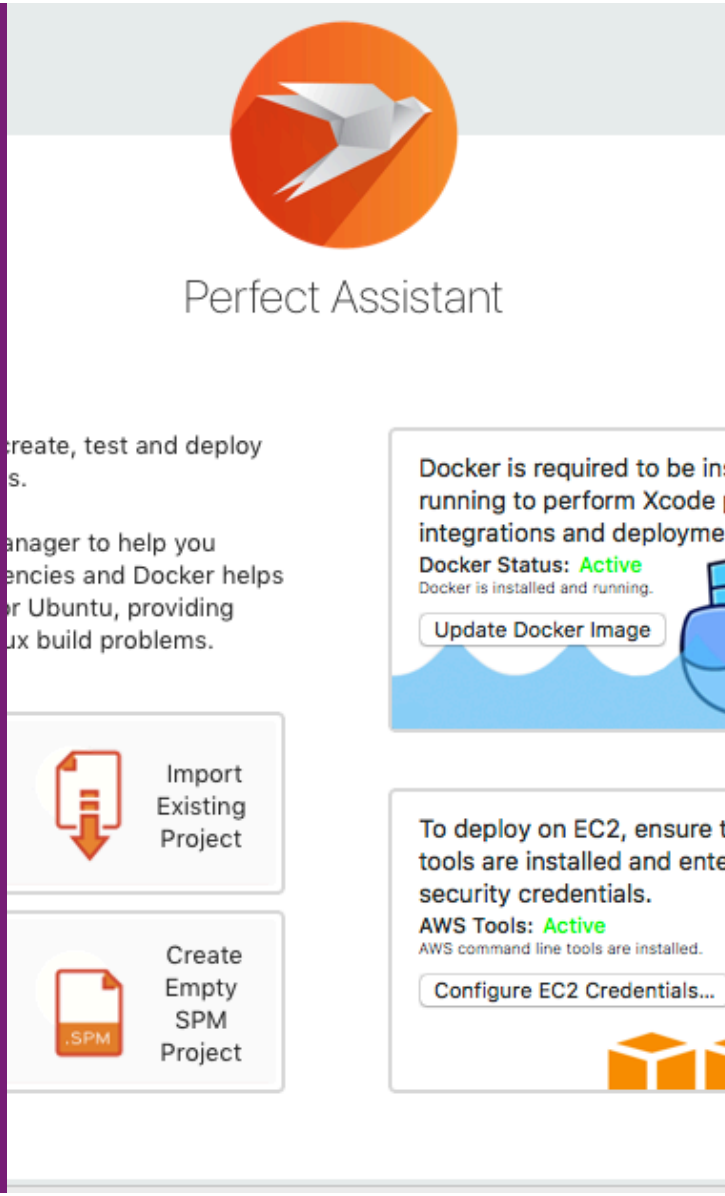

```
lassie says Bark!
```

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

PERFECT ASSISTANT

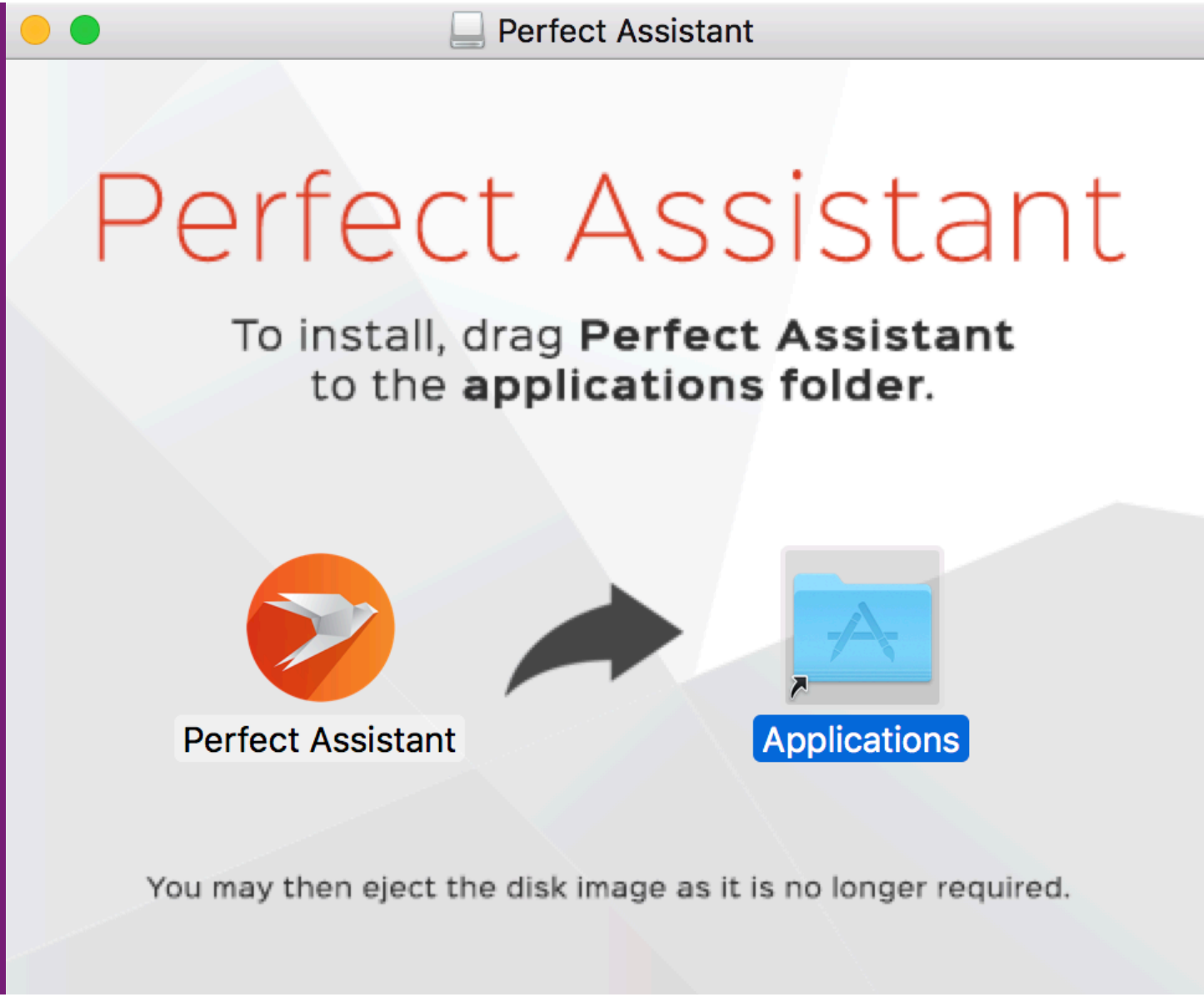
PERFECT ASSISTANT

- 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



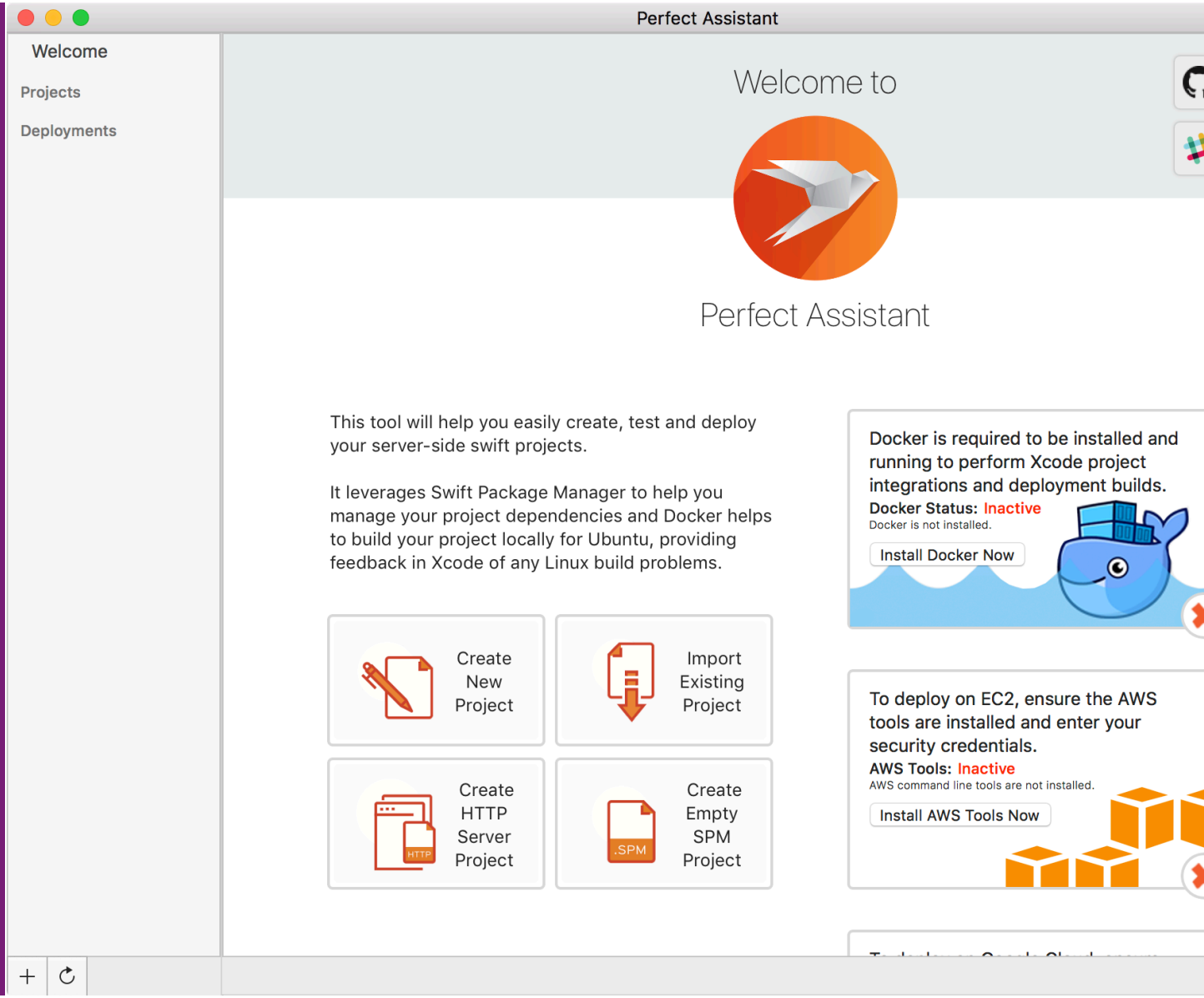
PERFECT ASSISTANT

- Download and install



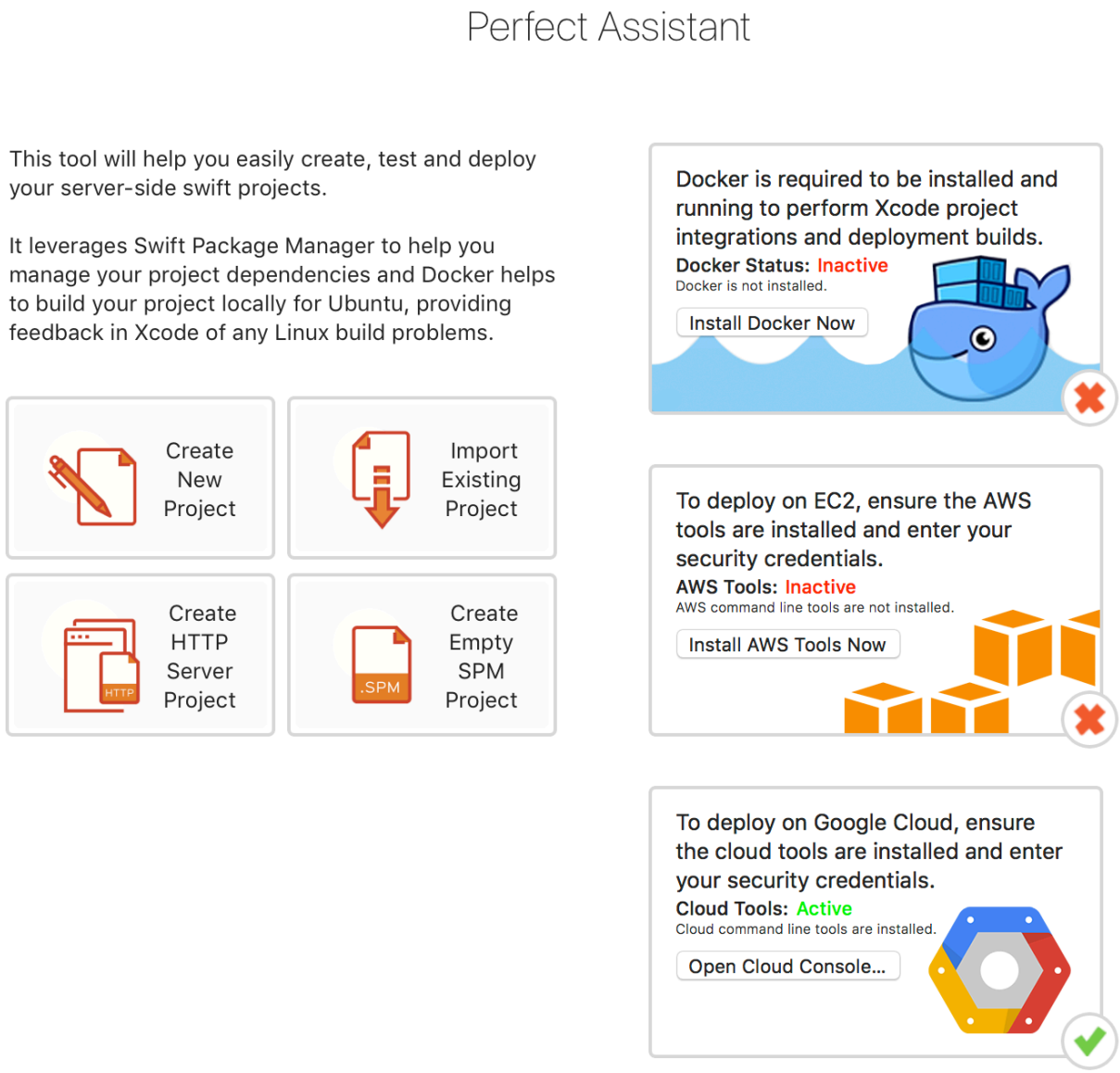
PERFECT ASSISTANT

- Download and install

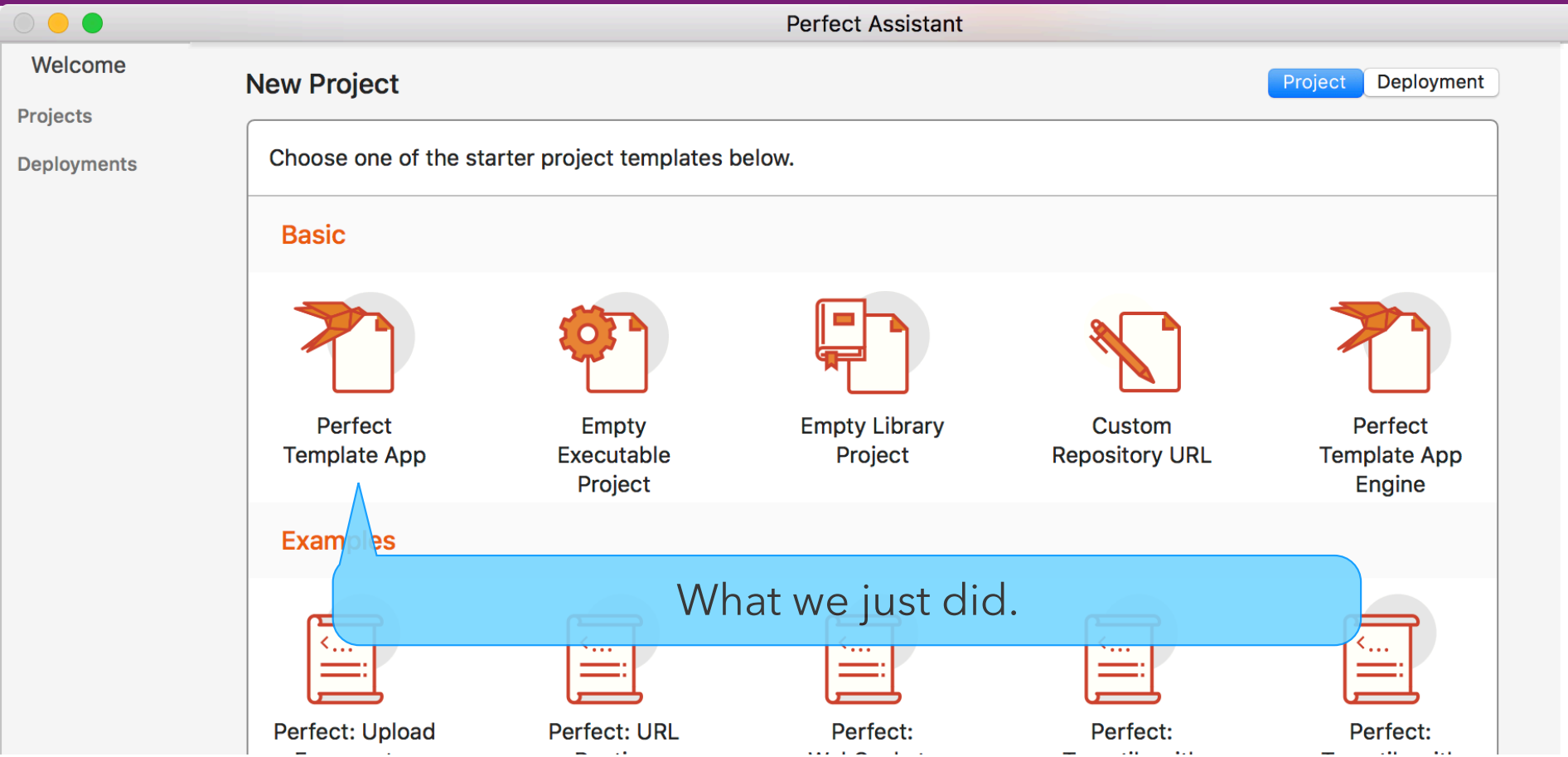


PERFECT ASSISTANT

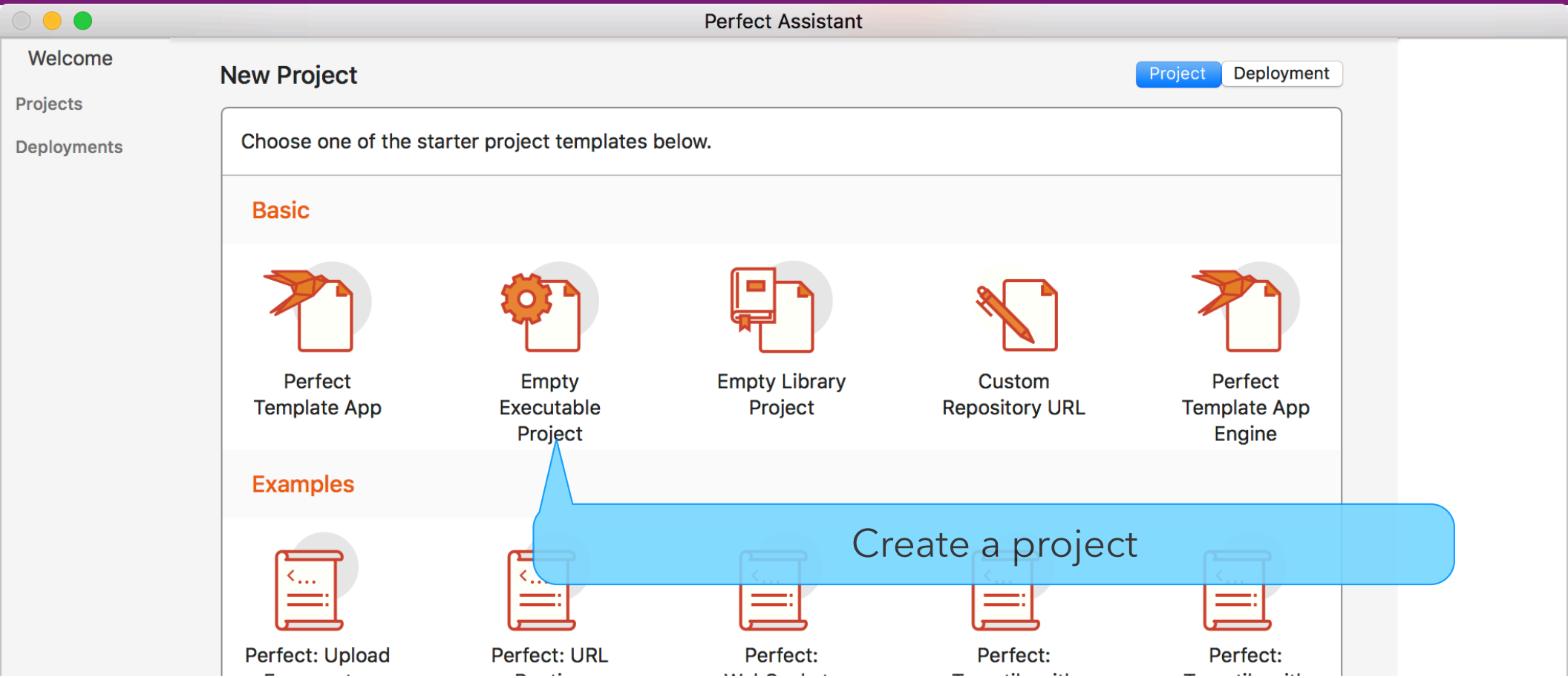
- Deploy using Docker



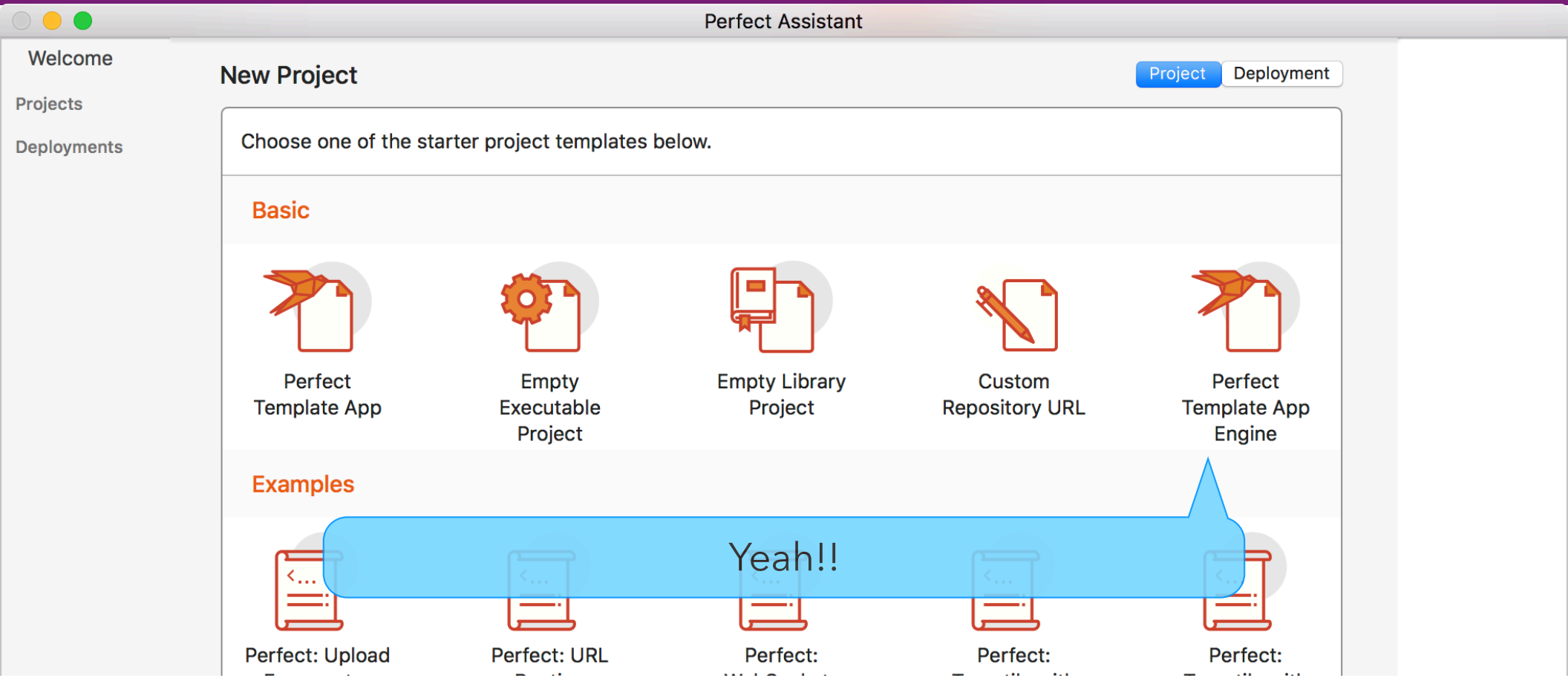
PERFECT ASSISTANT



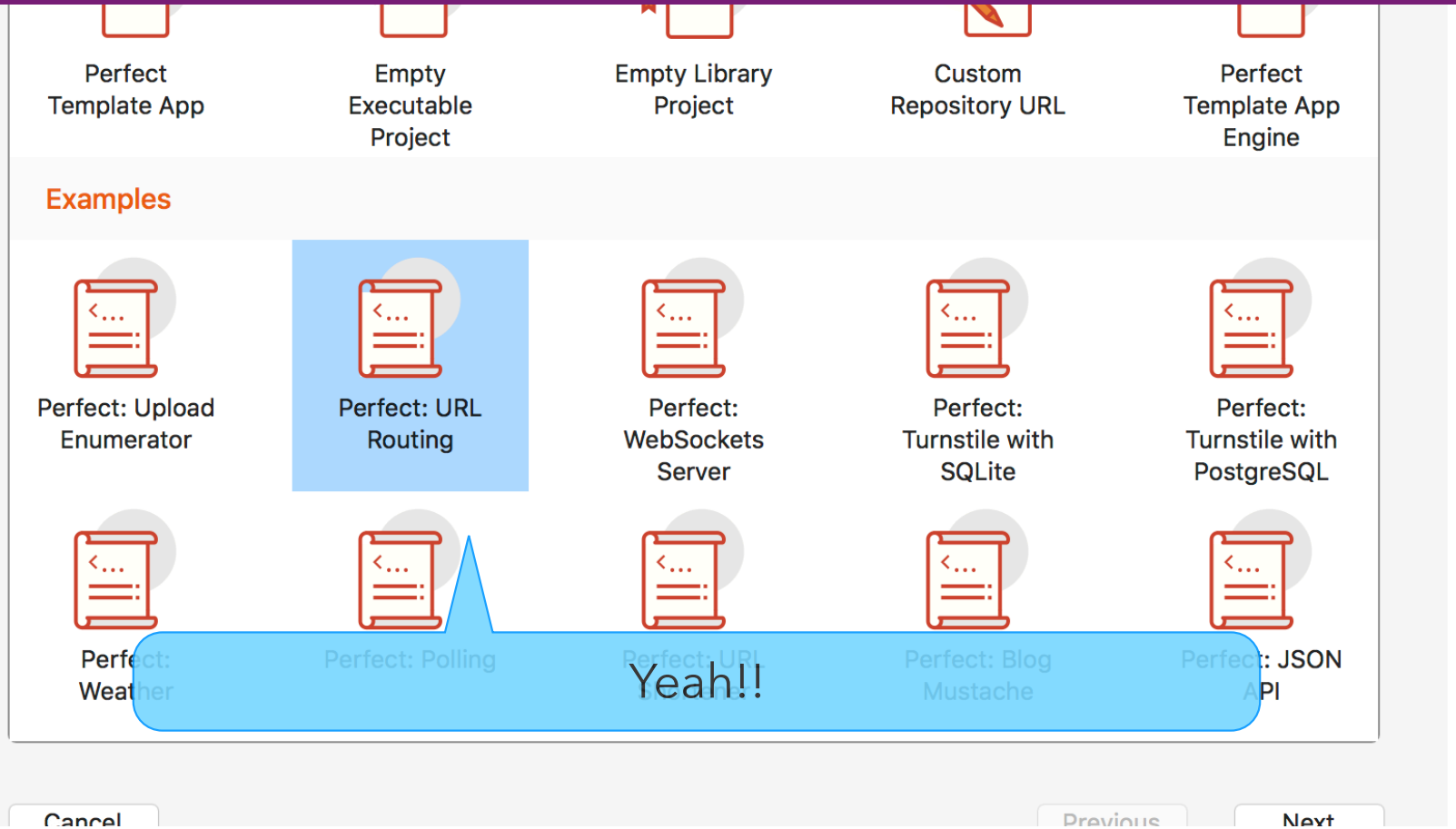
PERFECT ASSISTANT



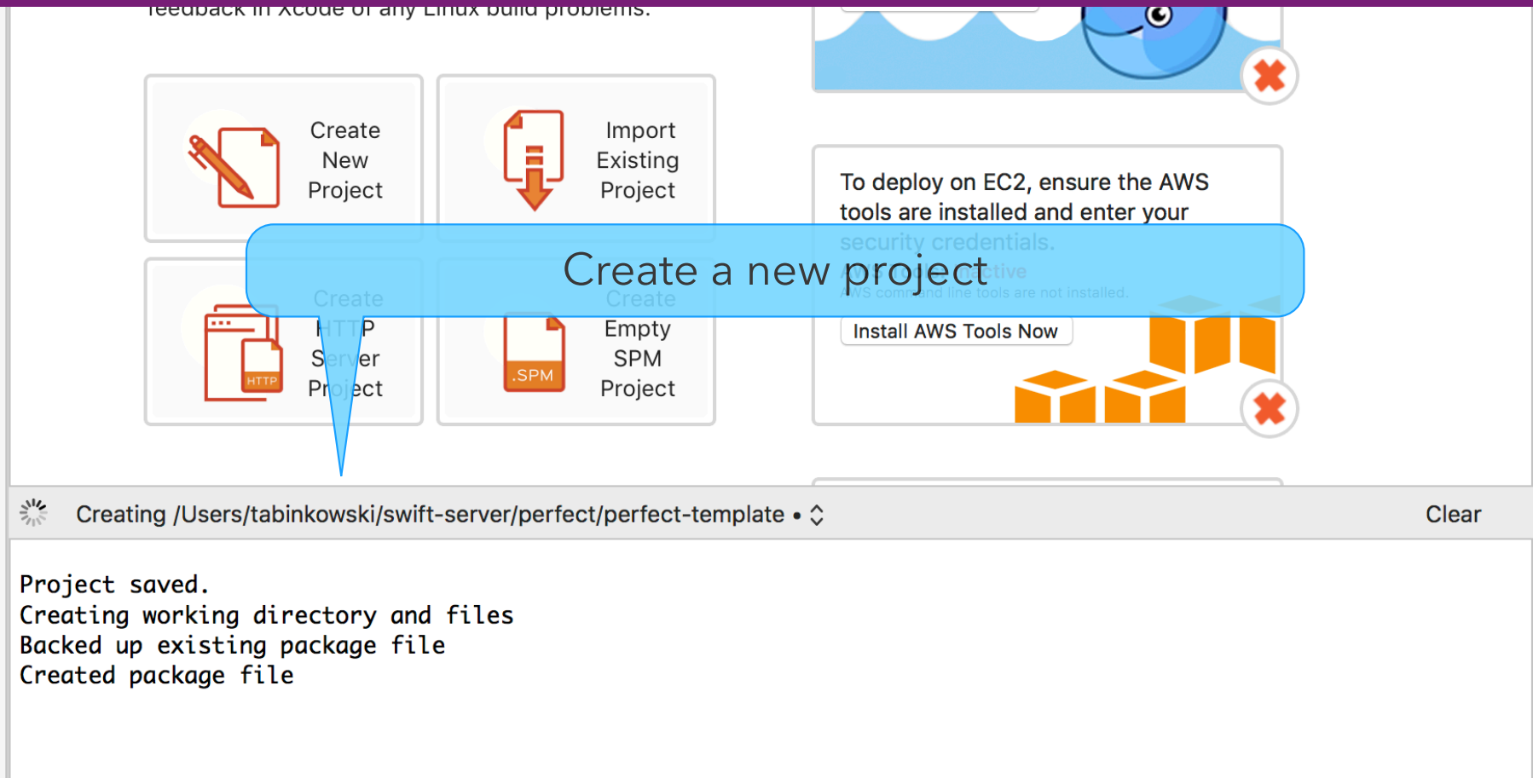
PERFECT ASSISTANT



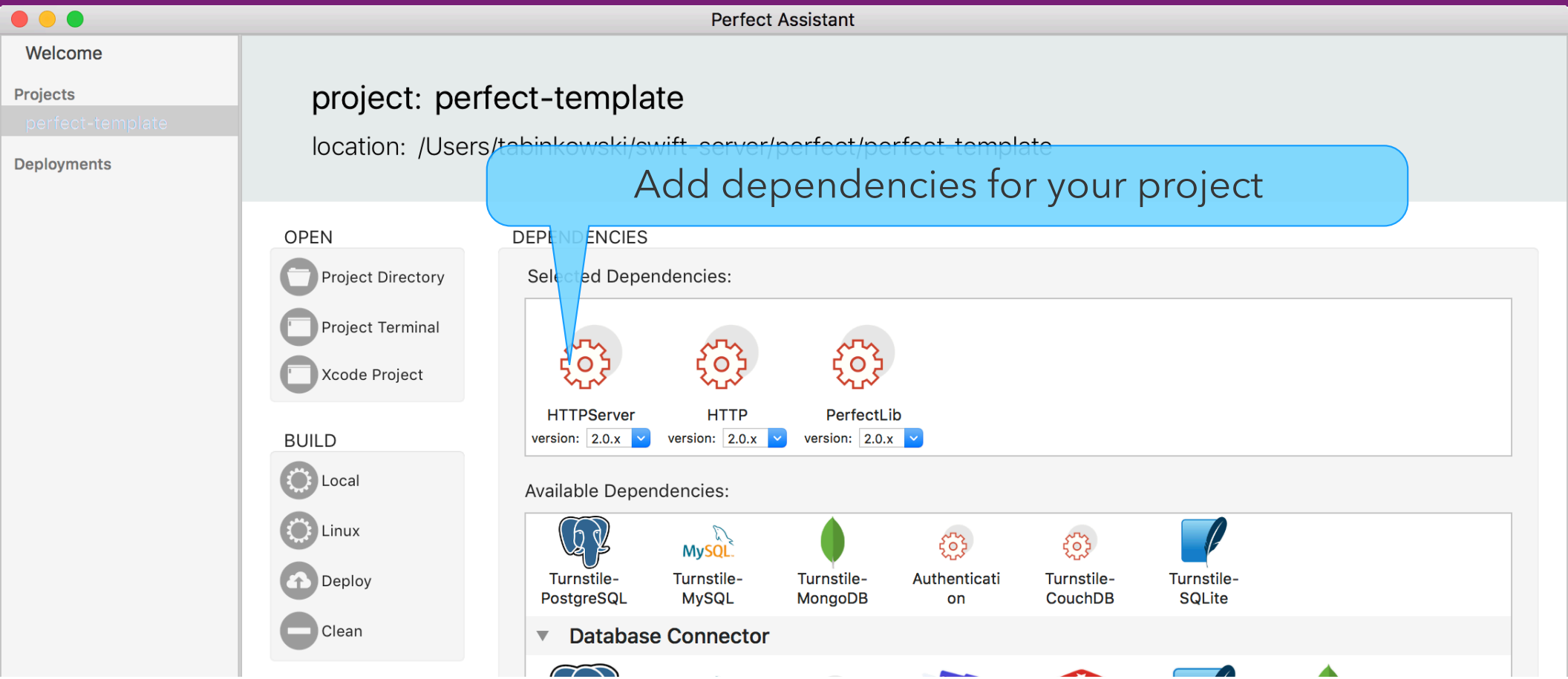
PERFECT ASSISTANT



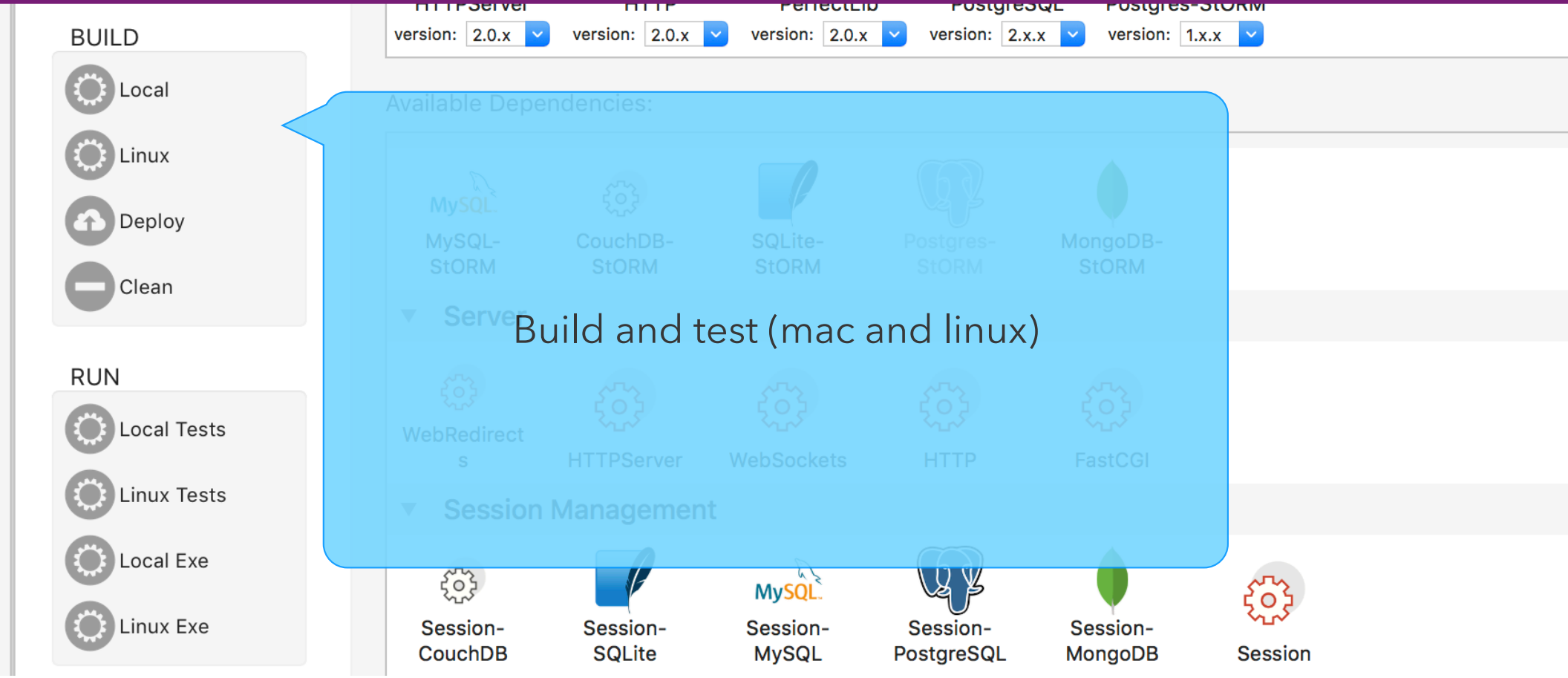
PERFECT ASSISTANT



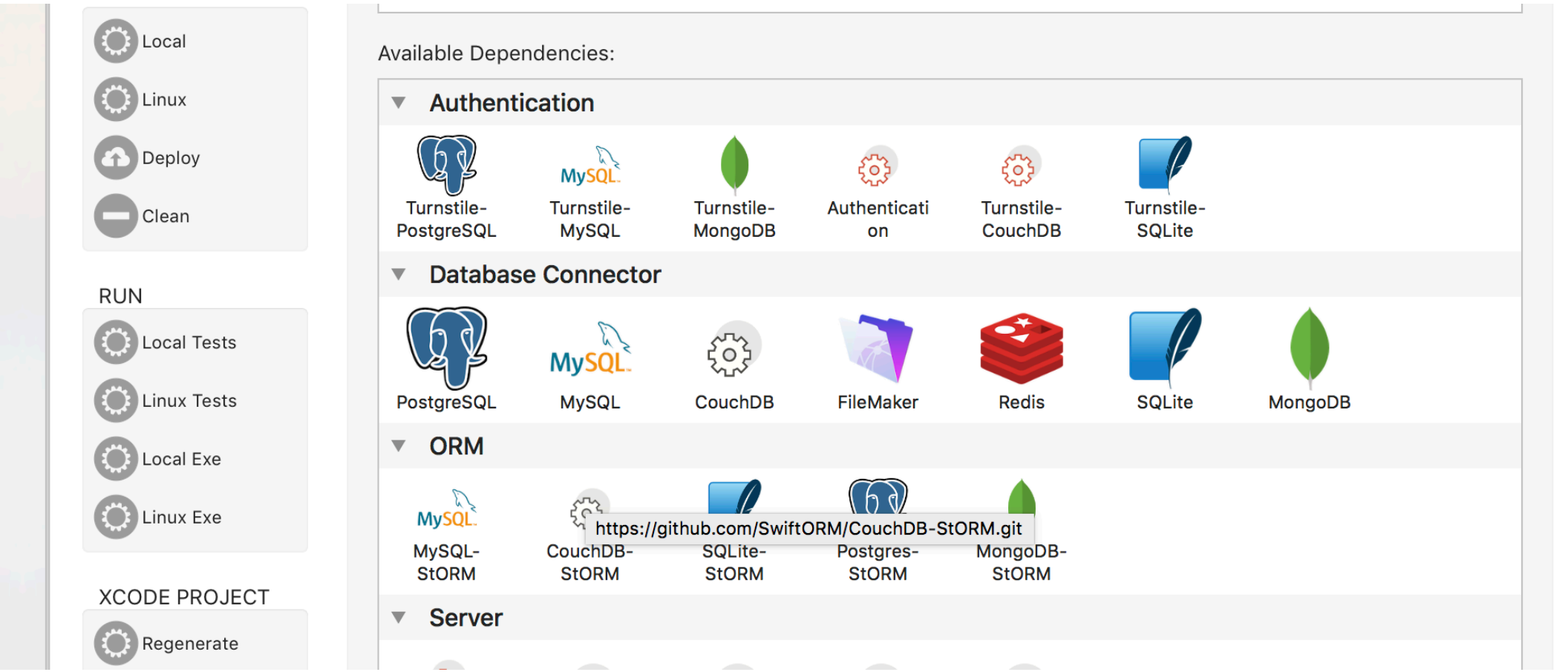
PERFECT ASSISTANT



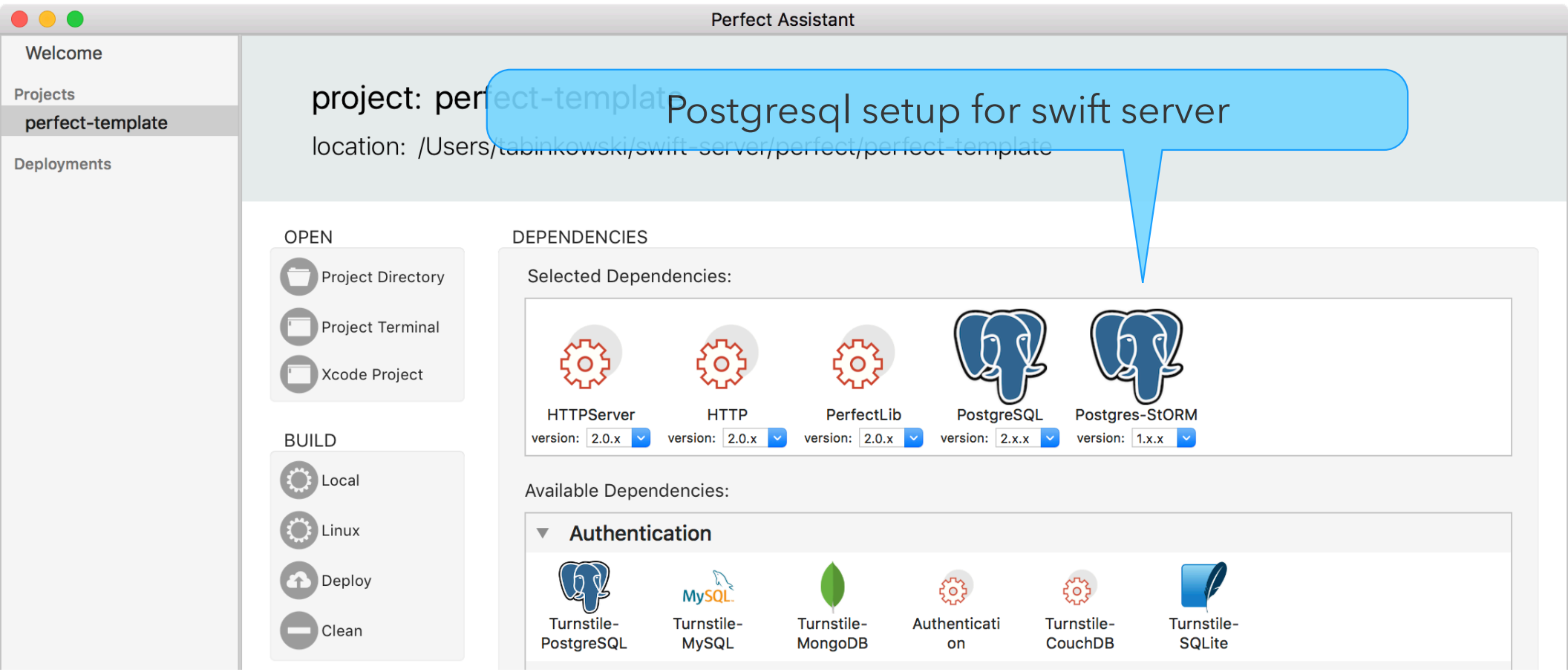
PERFECT ASSISTANT



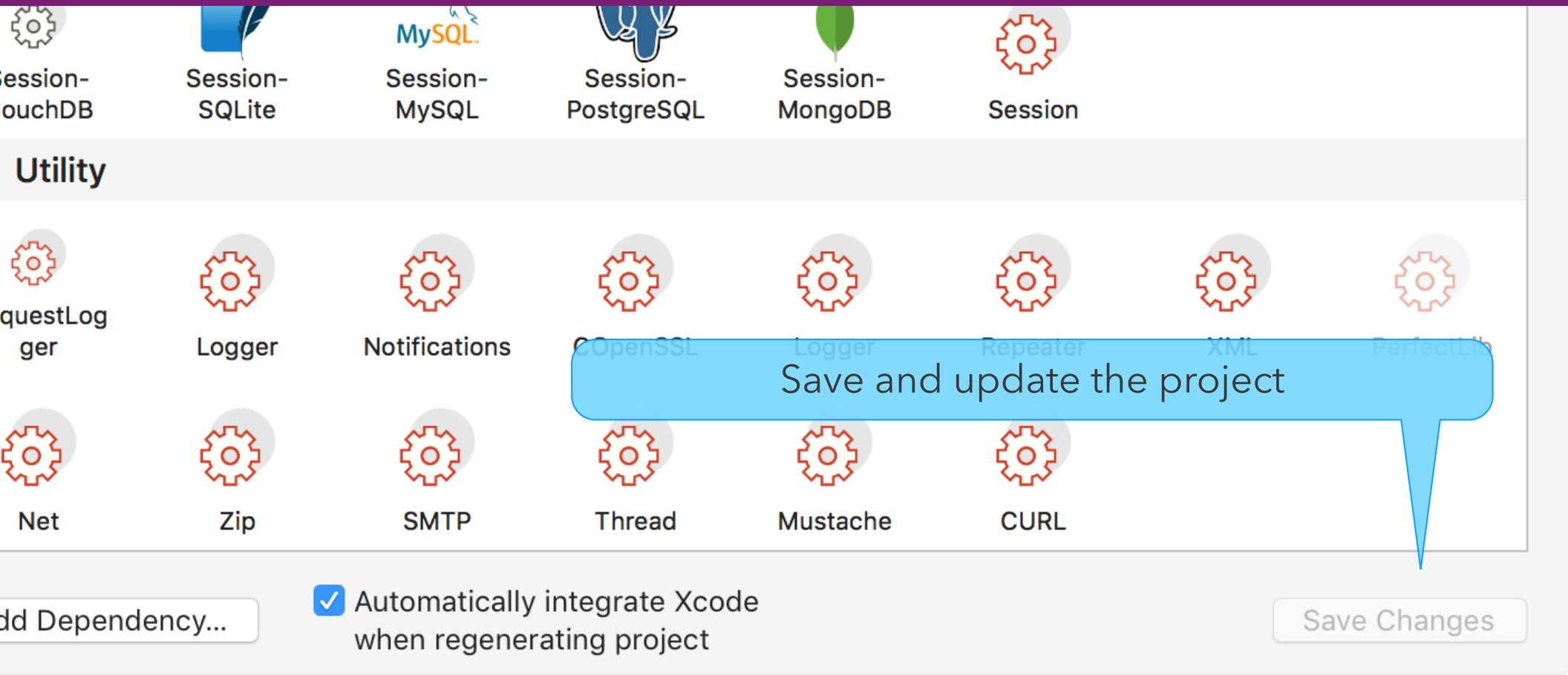
PERFECT ASSISTANT



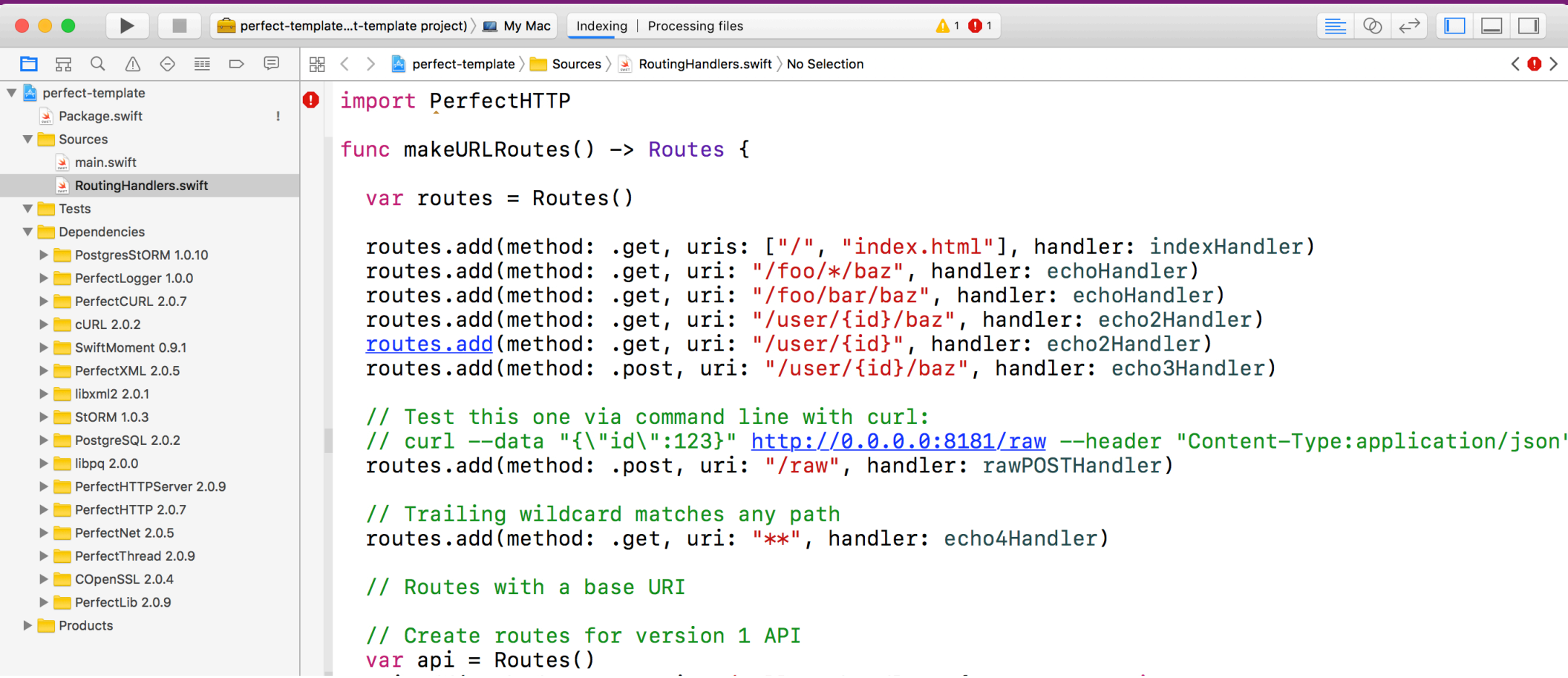
PERFECT ASSISTANT



PERFECT ASSISTANT

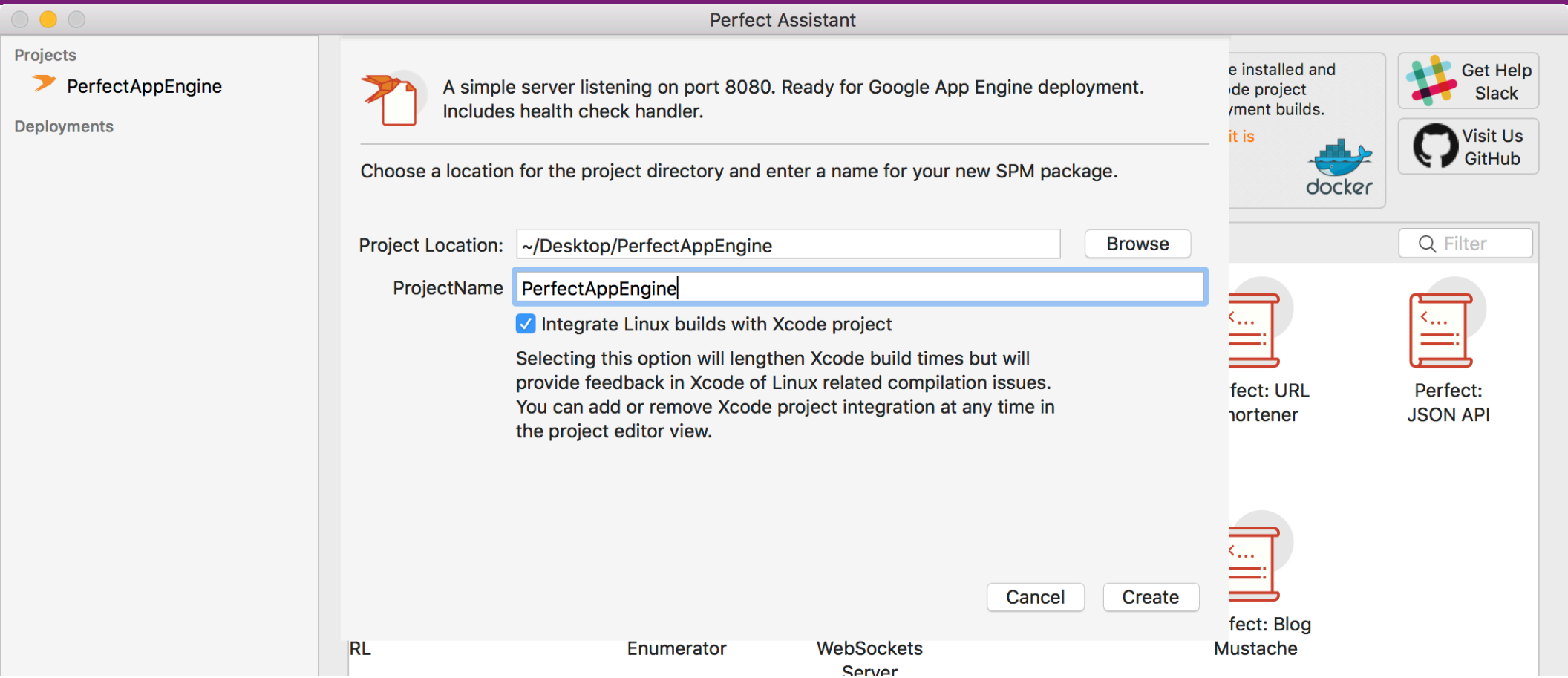


PERFECT ASSISTANT

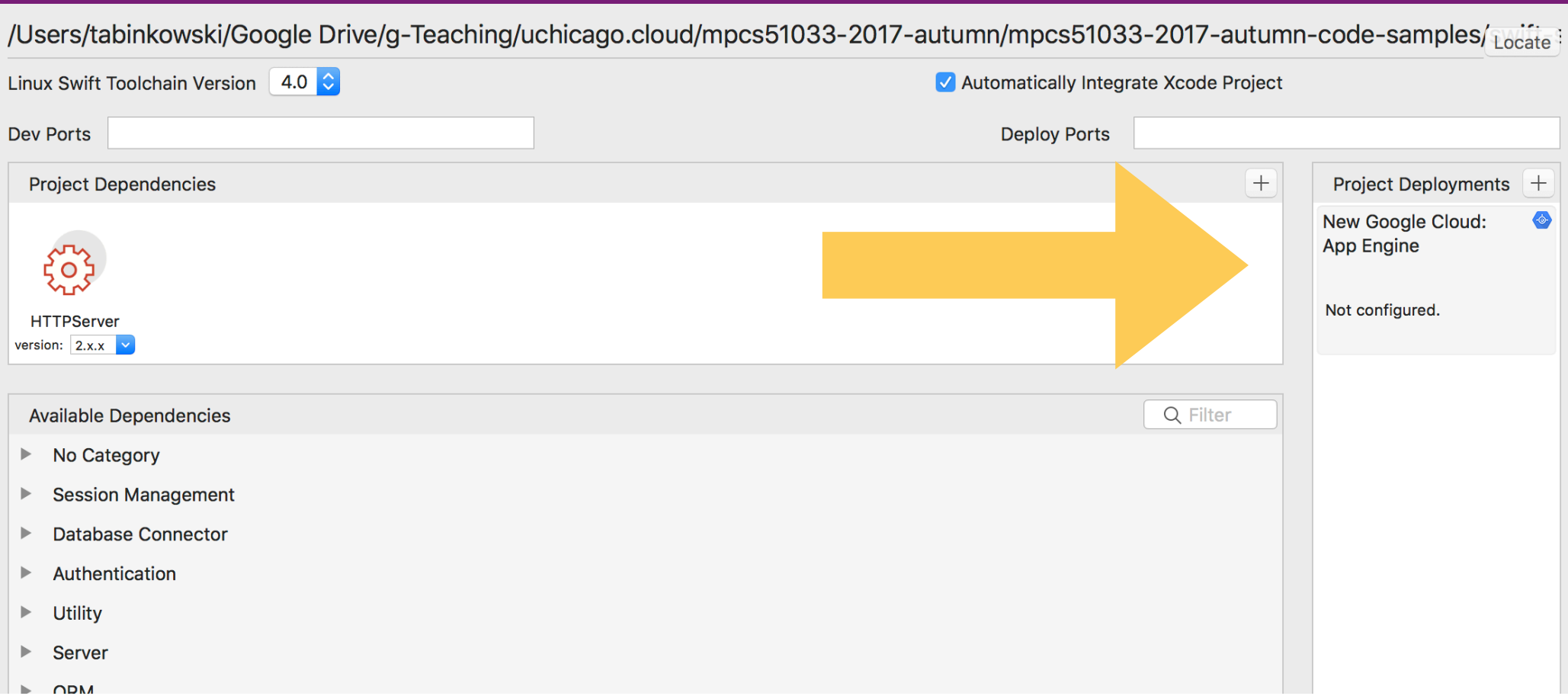


APP ENGINE

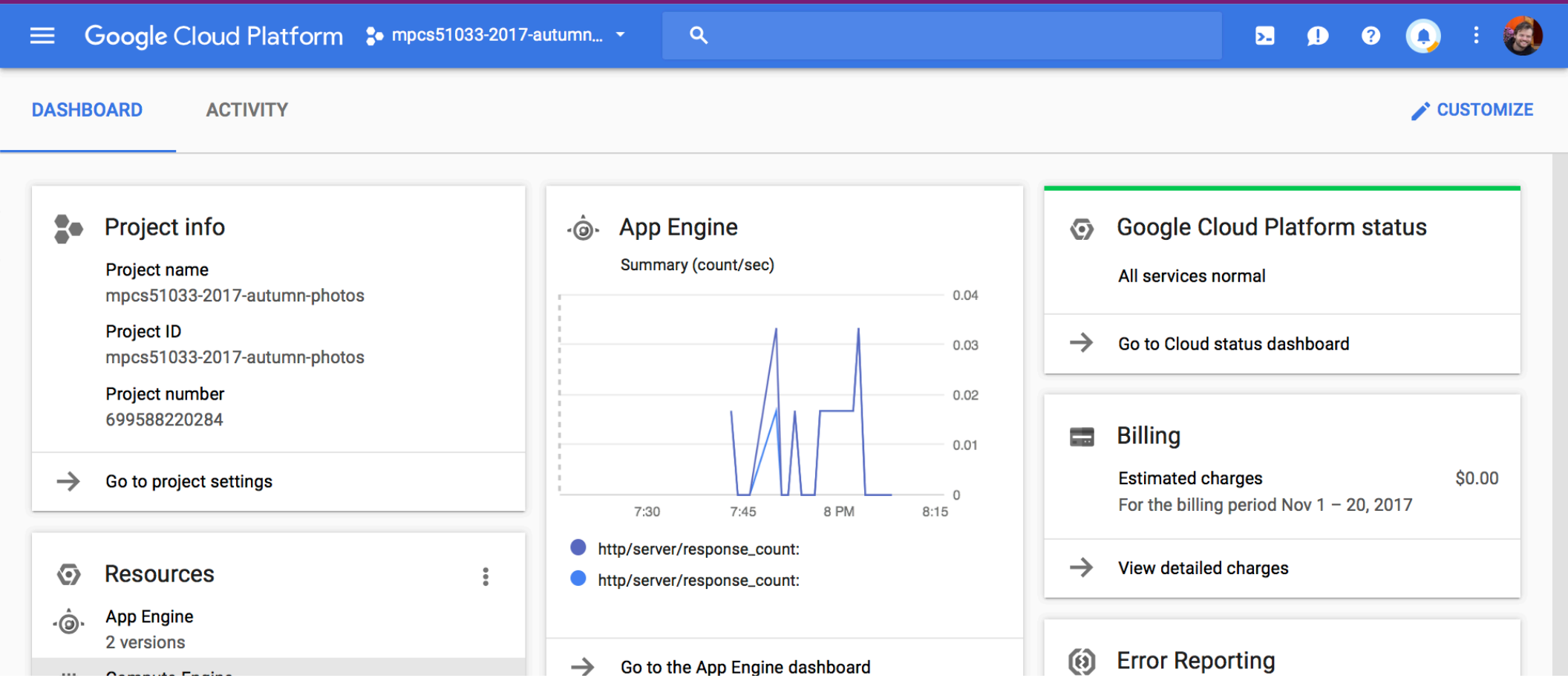
PERFECT ASSISTANT



PERFECT ASSISTANT



PERFECT ASSISTANT



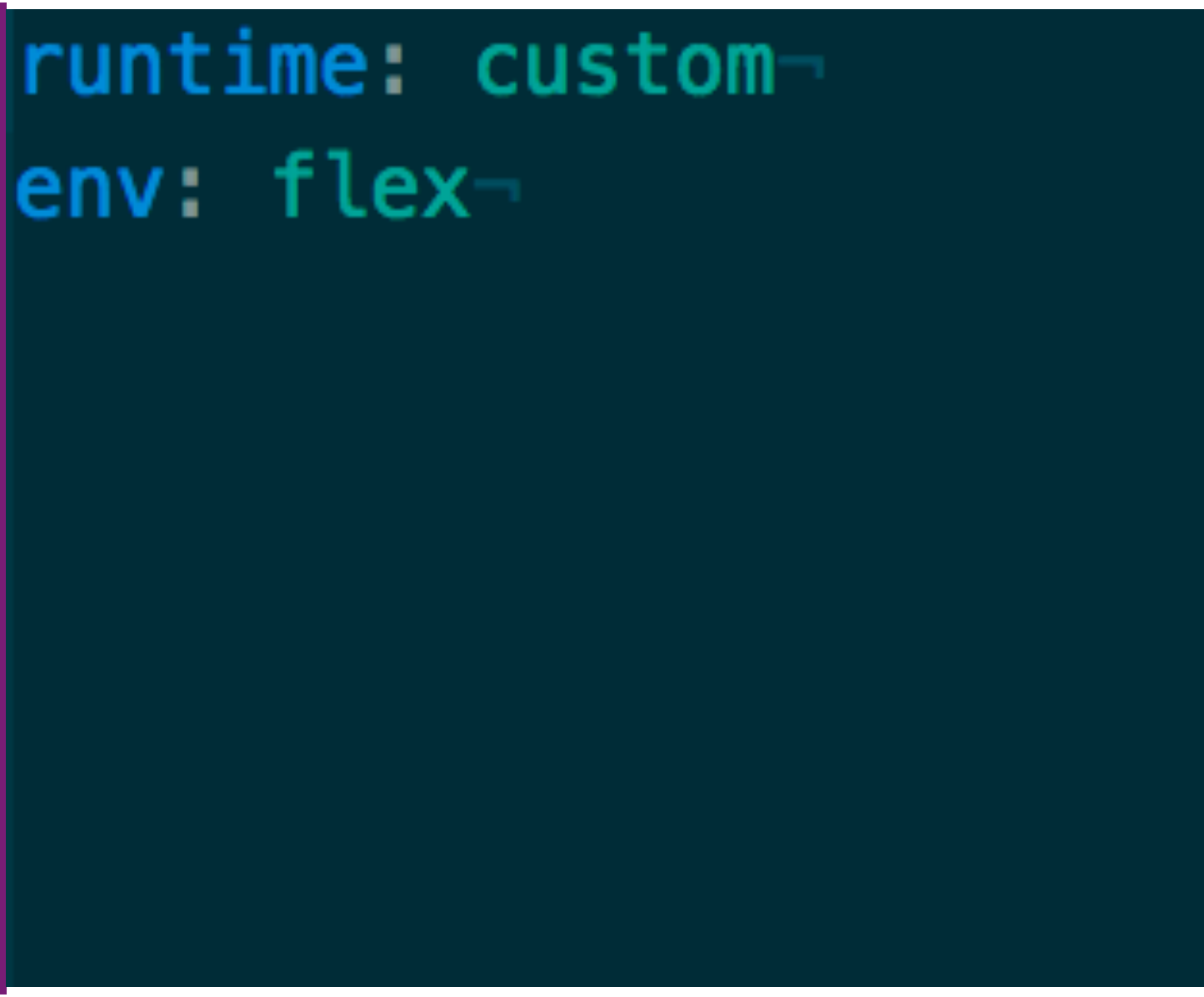
PERFECT ASSISTANT

- Docker file

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

# 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

- app.yaml



PERFECT ASSISTANT

- Deploy

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

THE FUTURE OF SWIFT ON THE SERVER

PERFORMANCE

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

Results Summary

	Perfect	Vapor	Kitura	Zewo	Node.js
Build Times					
1. --clean=dist	2nd Place	4th Place	1st Place	3rd Place	N/A
1. --clean	3rd Place	4th Place	2nd Place	1st Place	N/A
Memory Usage					
1. Starting Usage	1st Place	4th Place	3rd Place	2nd Place	5th Place
2. Peak Lower Load	1st Place	5th Place	4th Place	2nd Place	3rd Place
3. Peak Higher Load	1st Place	5th Place	4th Place	2nd Place	3rd Place
Thread Usage					
1. Starting Threads	1	1	4	1	7
2. Under Load	6	11	11	1	11
Blog Benchmarks					
1. Requests/Sec	1st Place	2nd Place	4th Place	5th Place	3rd Place
2. Latency	2nd Place	4th Place	1st Place	5th Place	3rd Place
JSON Benchmarks					
1. Requests/Sec	1st Place	3rd Place	2nd Place	5th Place	4th Place
2. Latency	1st Place	5th Place	2nd Place	4th Place	3rd Place

FUTURE OF SWIFT ON THE SERVER

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

PRODUCTION READY

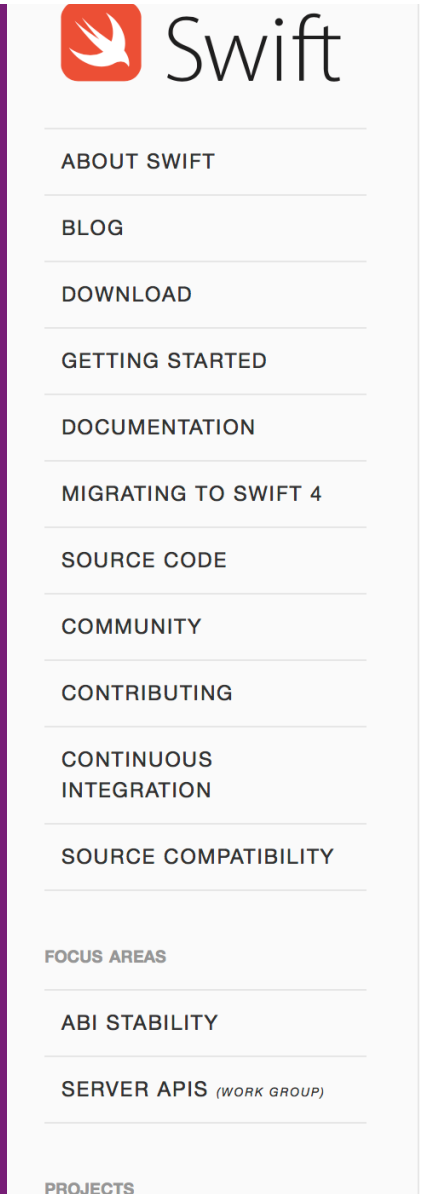
- Is it ready for prime time? Probably not yet.
- Missing frameworks
  - No URLSession
- The frameworks will make the shared code promise hard to keep
  - Custom models still need to be converted

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.

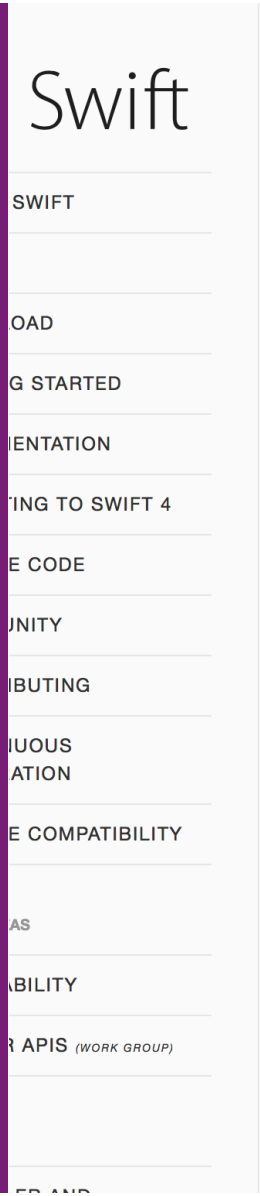
Modules

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 dependendies



Package Manager

The Swift Package Manager is a tool for managing the distributi
Swift code. It's integrated with the Swift build system to automa
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.

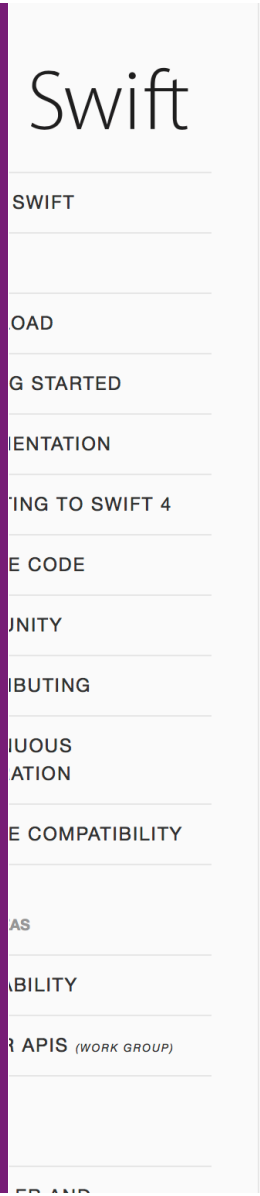
Modules

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

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

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



Package Manager

The Swift Package Manager is a tool for managing the distributi
Swift code. It's integrated with the Swift build system to automa
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.

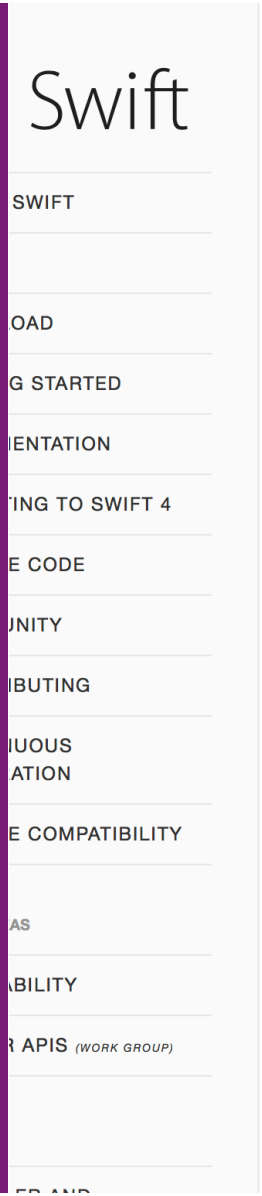
Modules

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

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

SWIFT PACKAGE MANAGER

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



Package Manager

The Swift Package Manager is a tool for managing the distributi
Swift code. It's integrated with the Swift build system to automa
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.

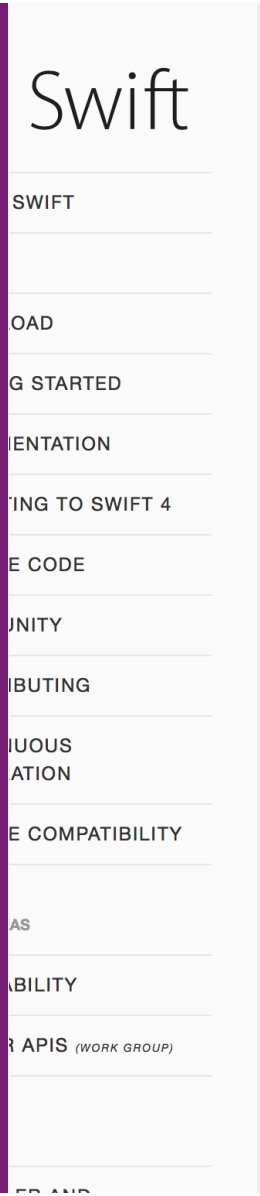
Modules

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

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

SWIFT PACKAGE MANAGER

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



Package Manager

The Swift Package Manager is a tool for managing the distributi
Swift code. It's integrated with the Swift build system to automa
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.

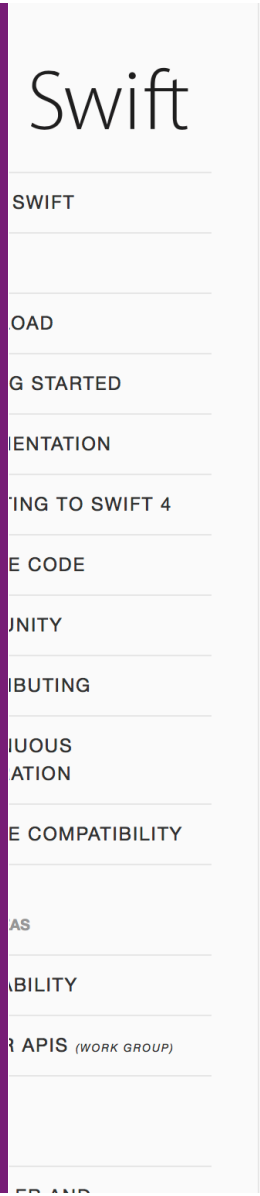
Modules

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

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

SWIFT PACKAGE MANAGER

- Products
  - Target of a package or executable



Package Manager

The Swift Package Manager is a tool for managing the distributi
Swift code. It's integrated with the Swift build system to automa
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.

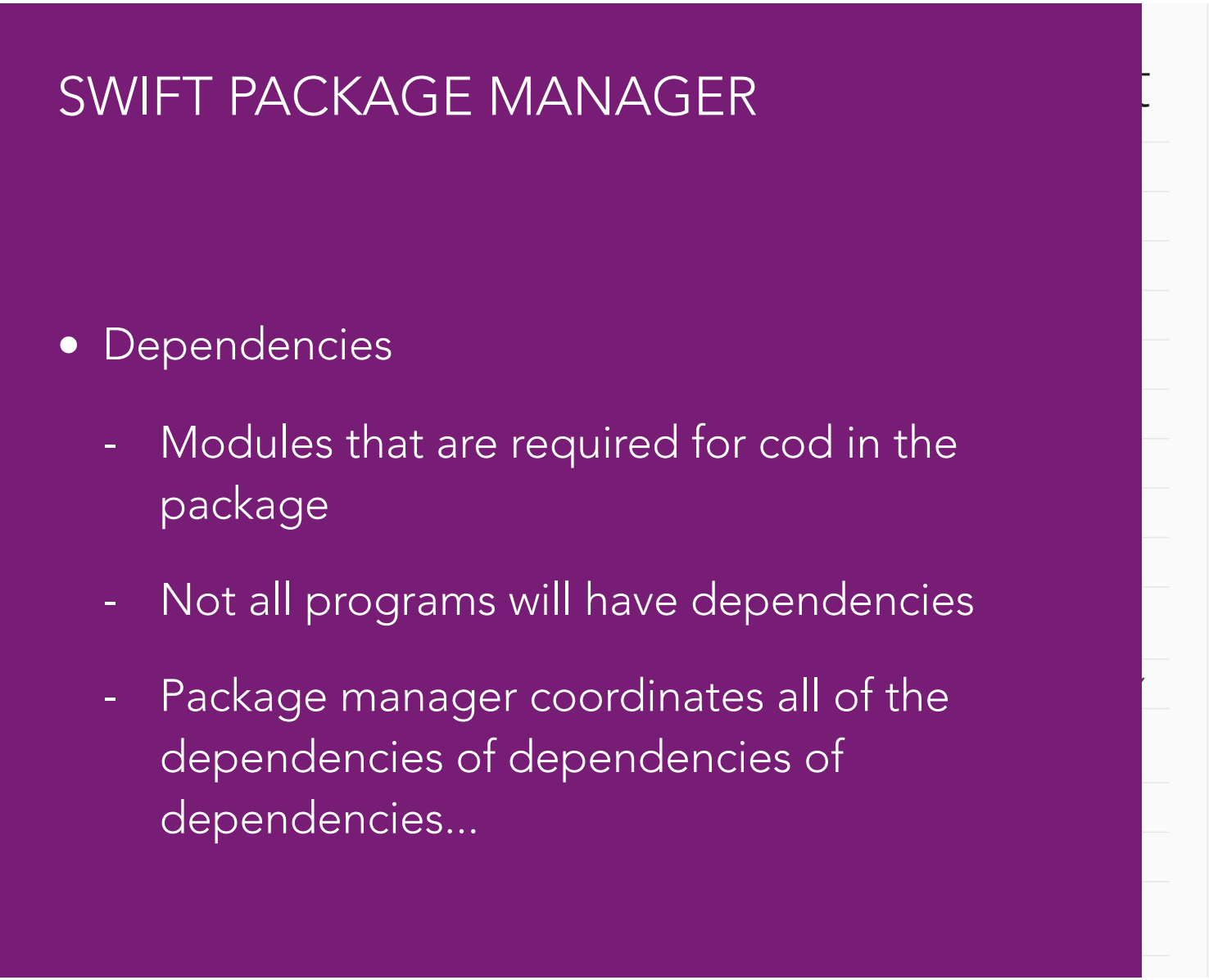
Modules

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

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

SWIFT PACKAGE MANAGER

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



Package Manage

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

The Package Manager is included in Swift 3.0 and

Conceptual Overview

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

Modules

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

A program may have all of its code in a single mo
other modules as *dependencies*. Aside from the h

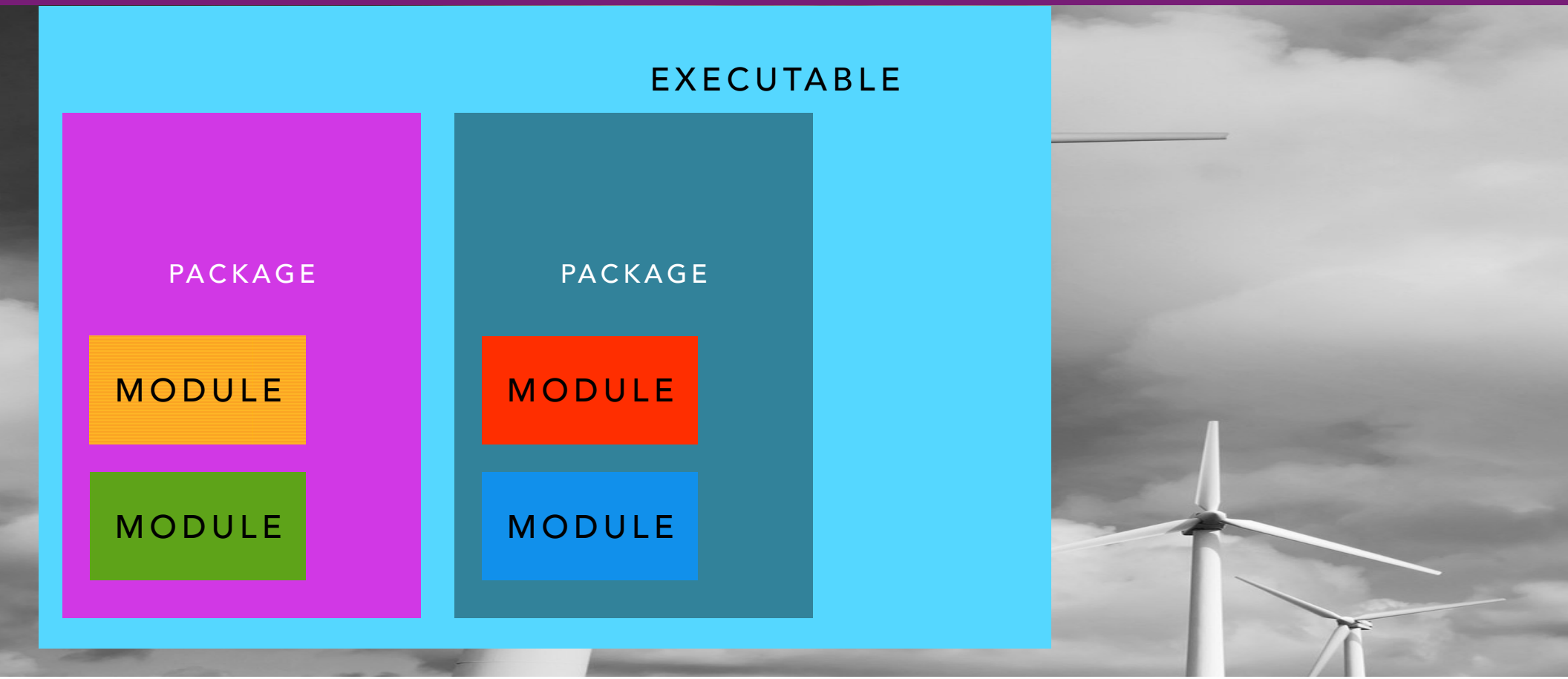
SWIFT PACKAGE MANAGER

SWIFT PACKAGE MANAGER

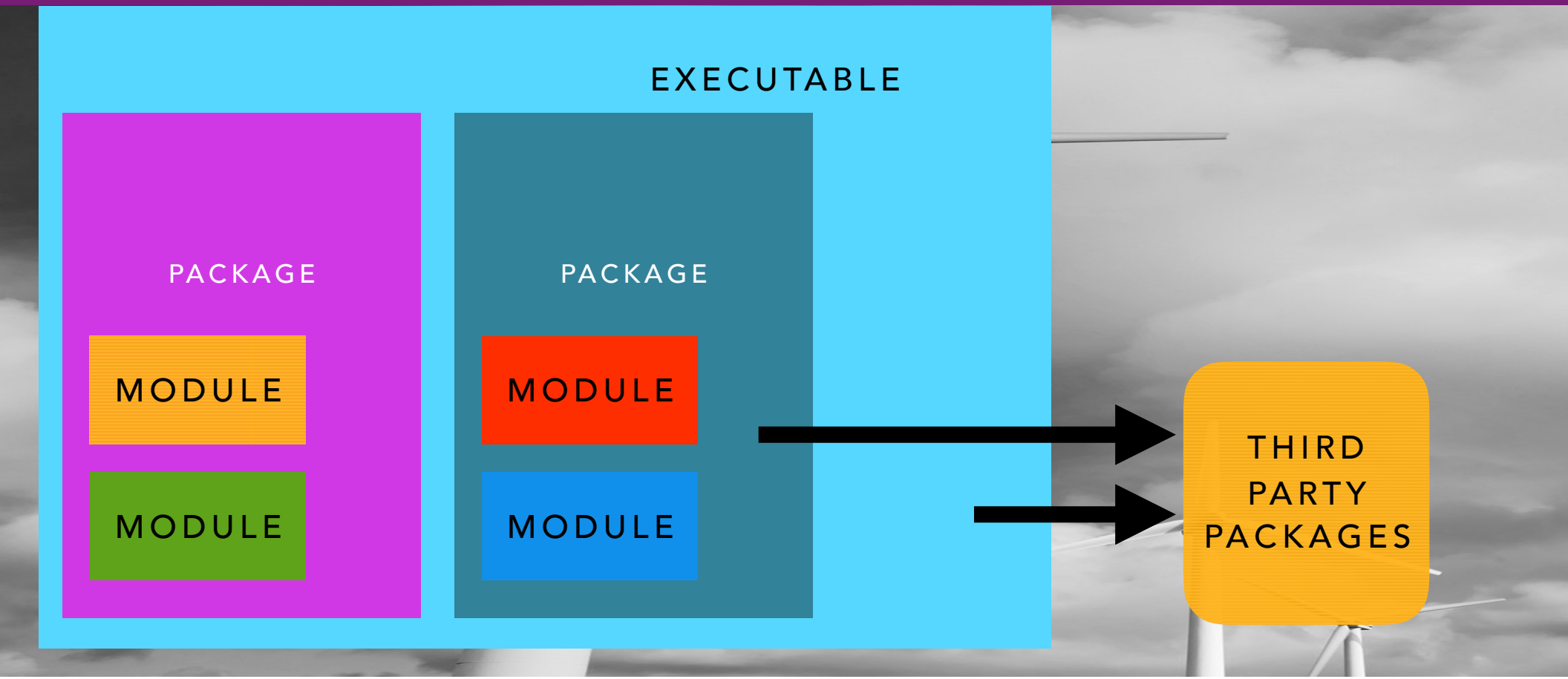
SWIFT PACKAGE MANAGER

SWIFT PACKAGE MANAGER

SWIFT PACKAGE MANAGER



SWIFT PACKAGE MANAGER



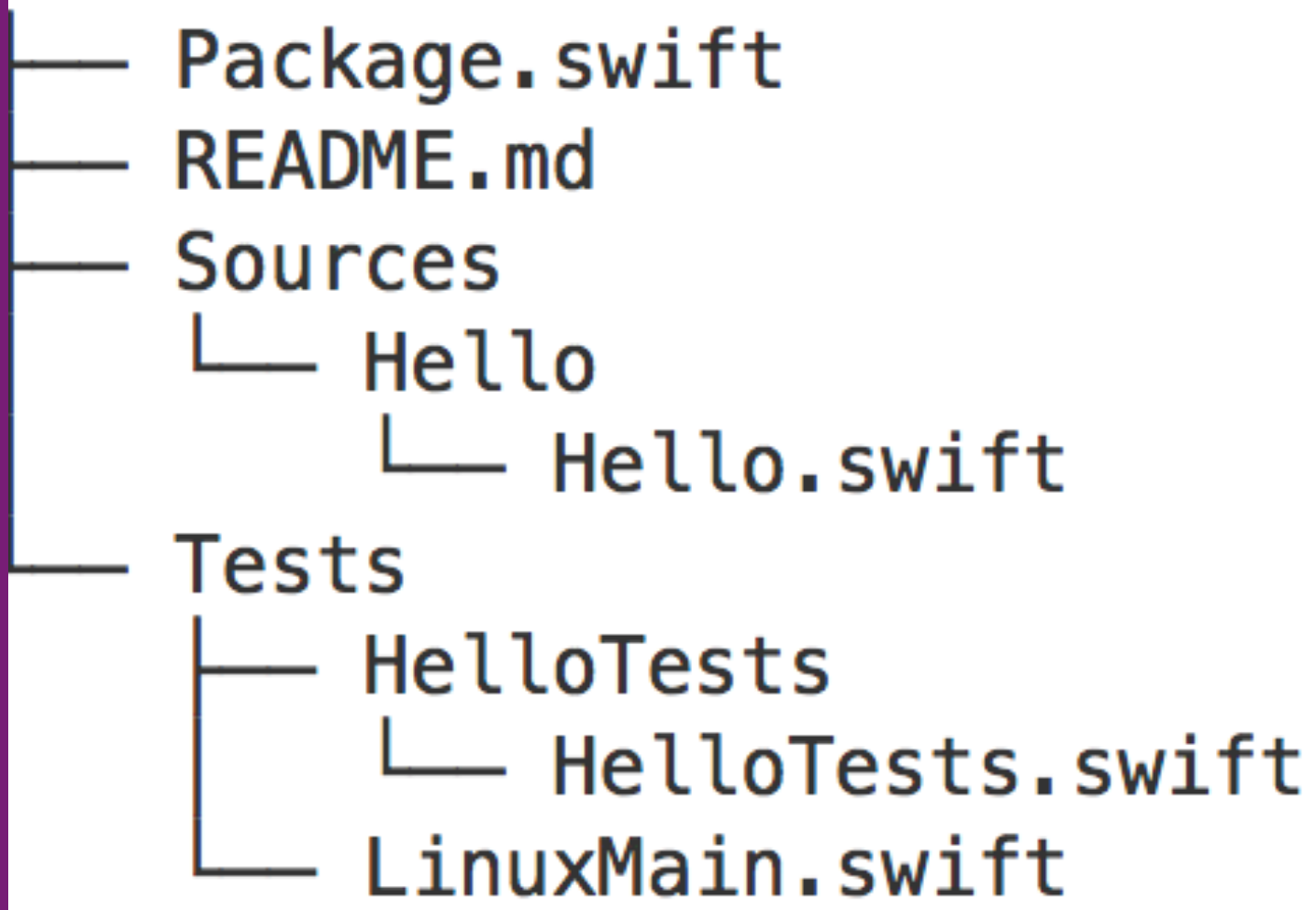
CREATE A PACKAGE

CREATE A PACKAGE

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

CREATE A PACKAGE

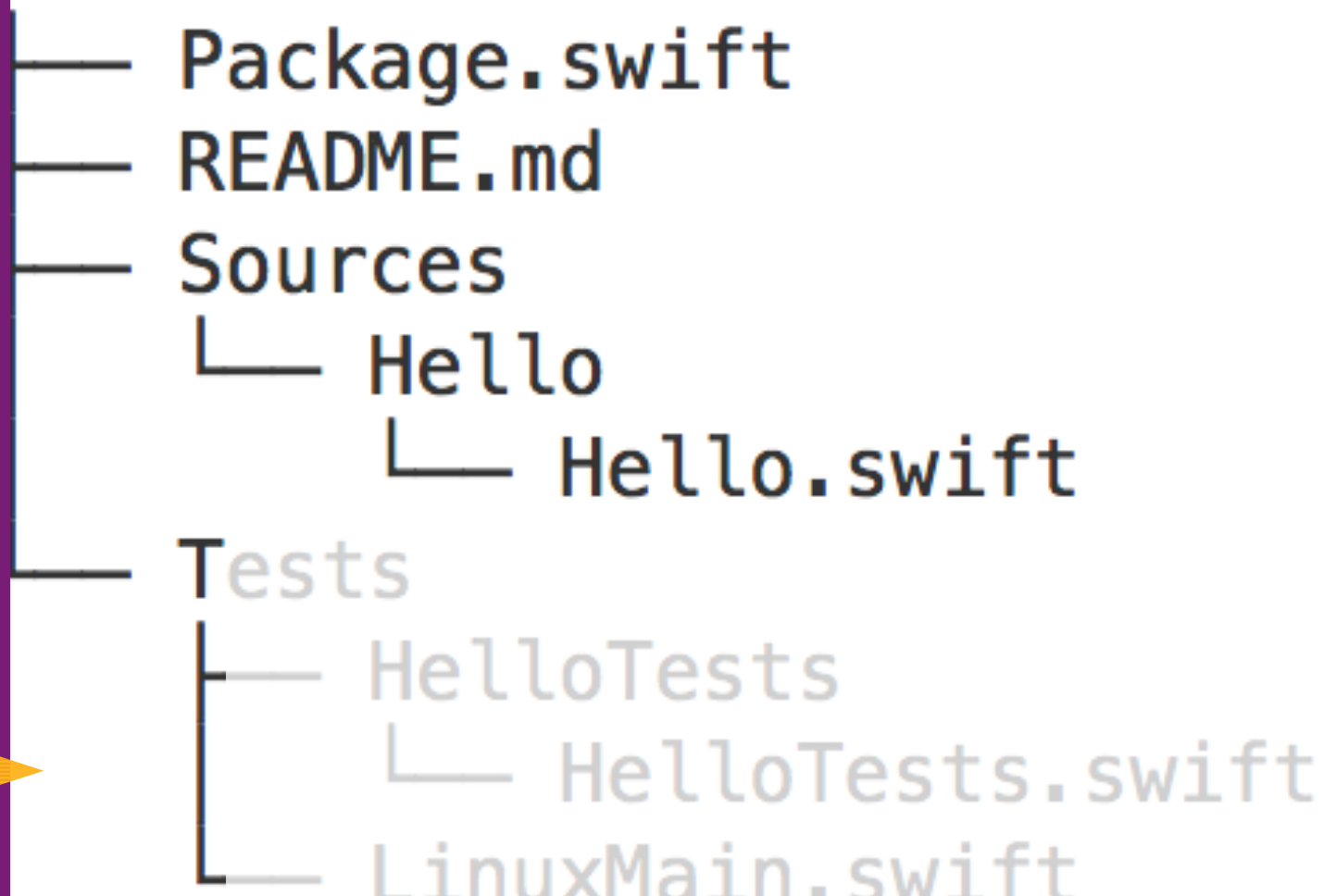
- Creates a directory/file structure
- Name of folder is name of package



CREATE A PACKAGE

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

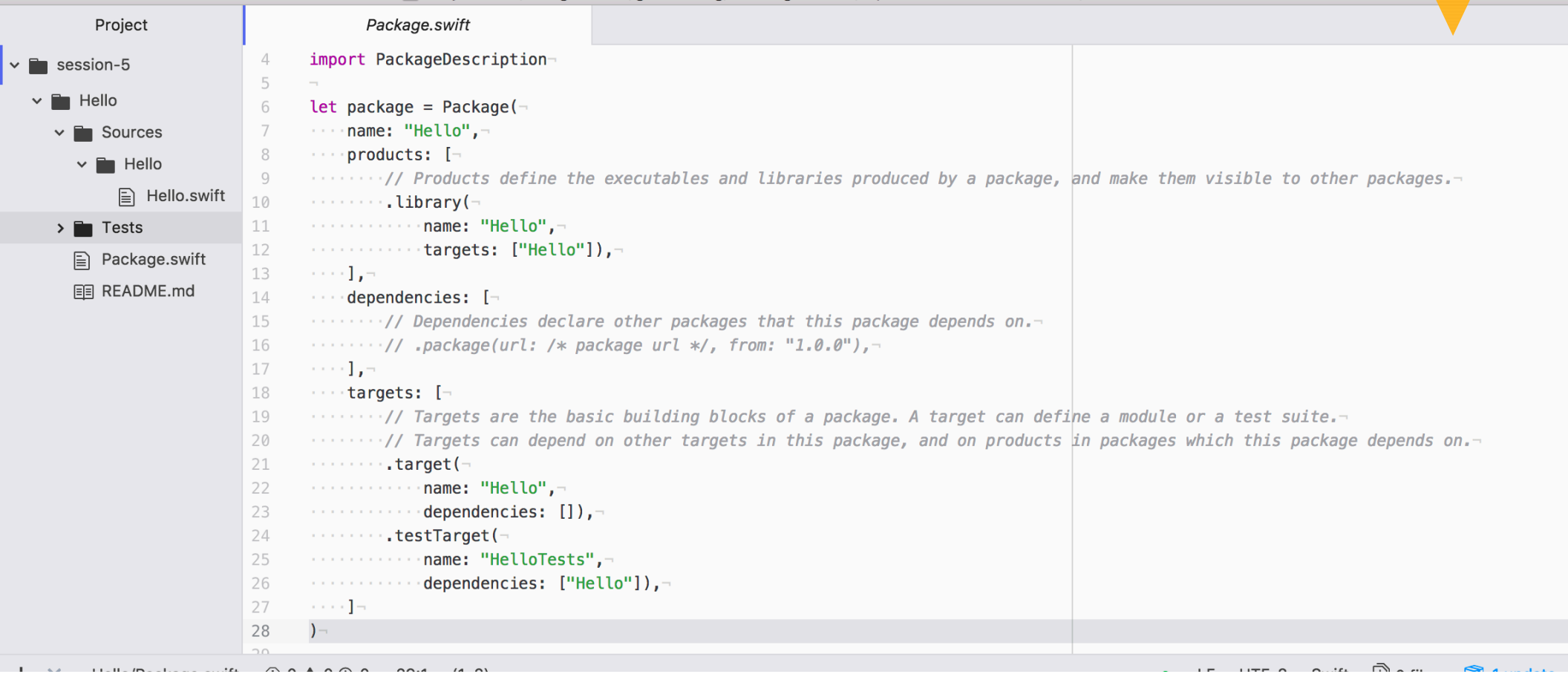
IGNORE TESTS FOR NOW



CREATE A PACKAGE

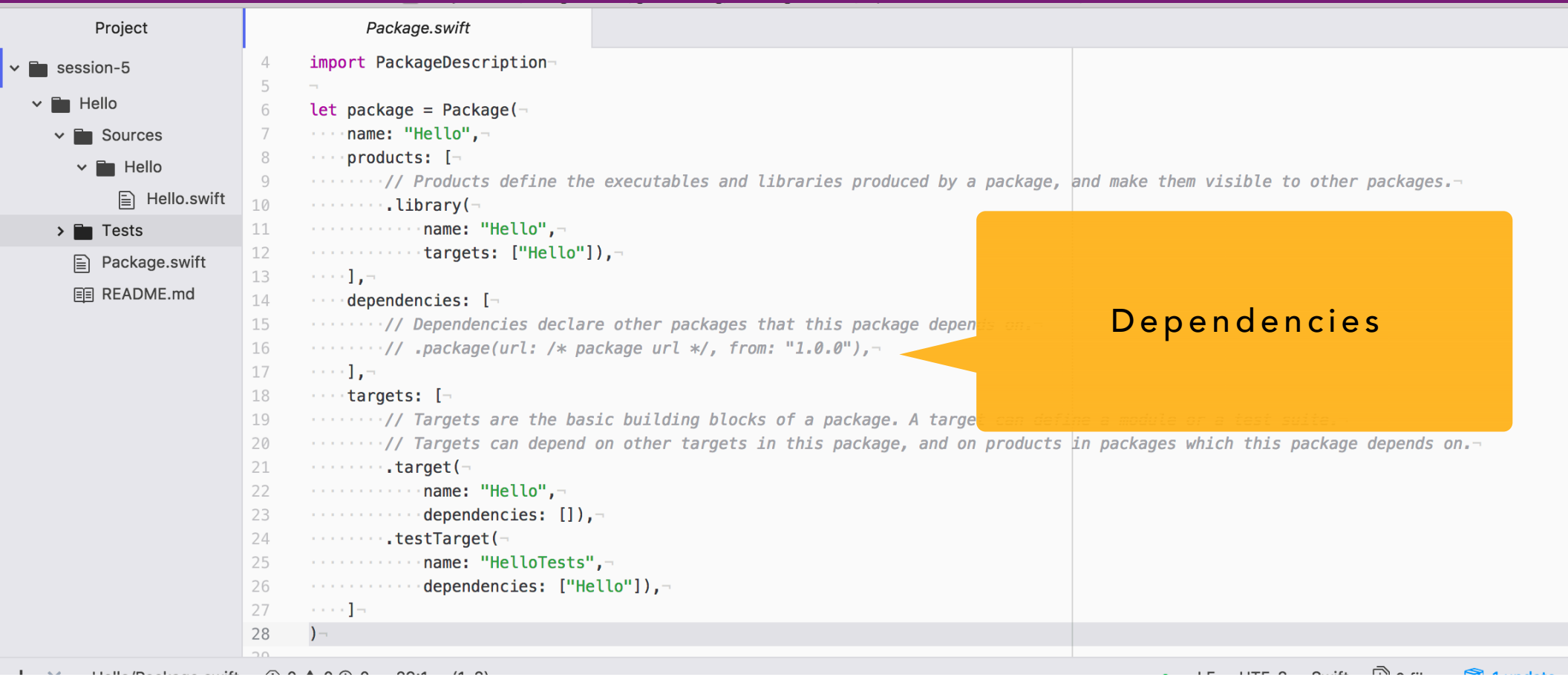
CREATE A PACKAGE

Manifest file describing the package

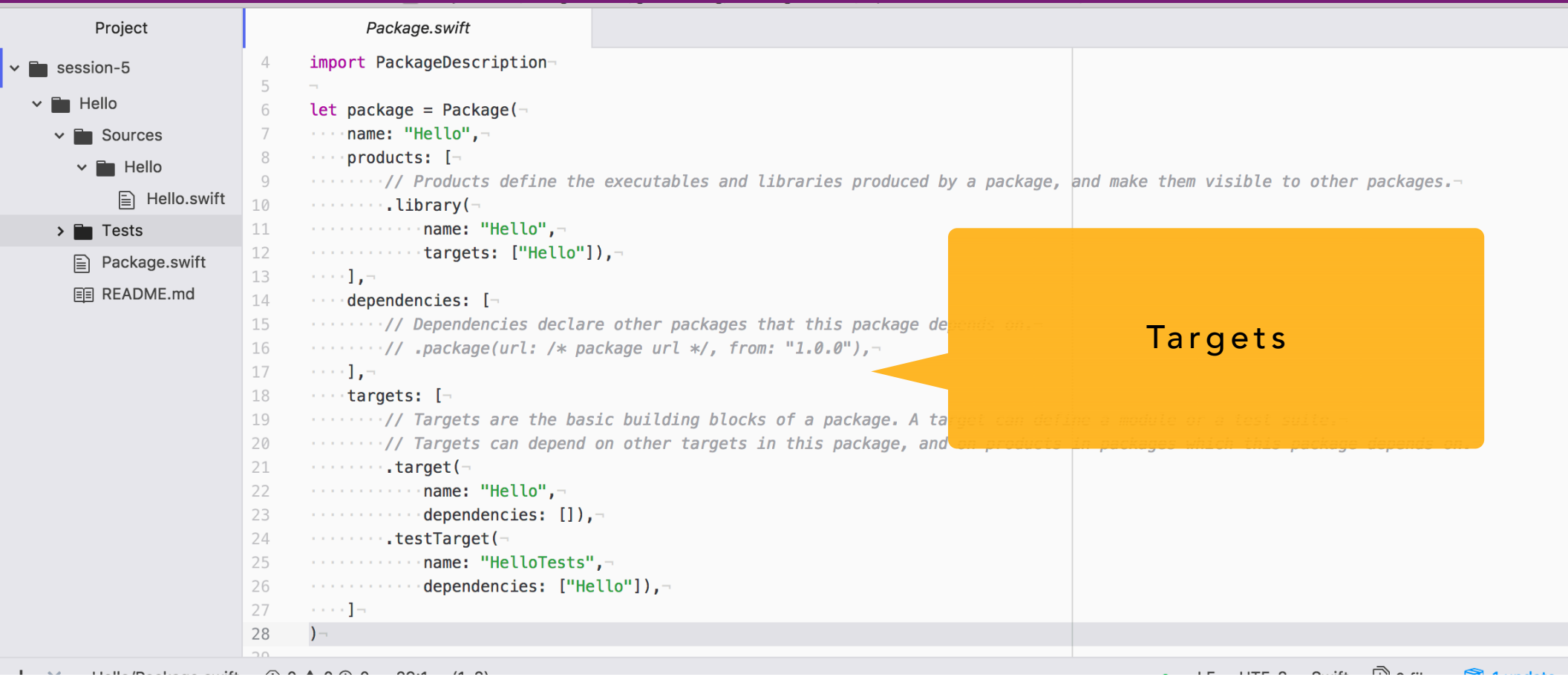


CREATE A PACKAGE

CREATE A PACKAGE

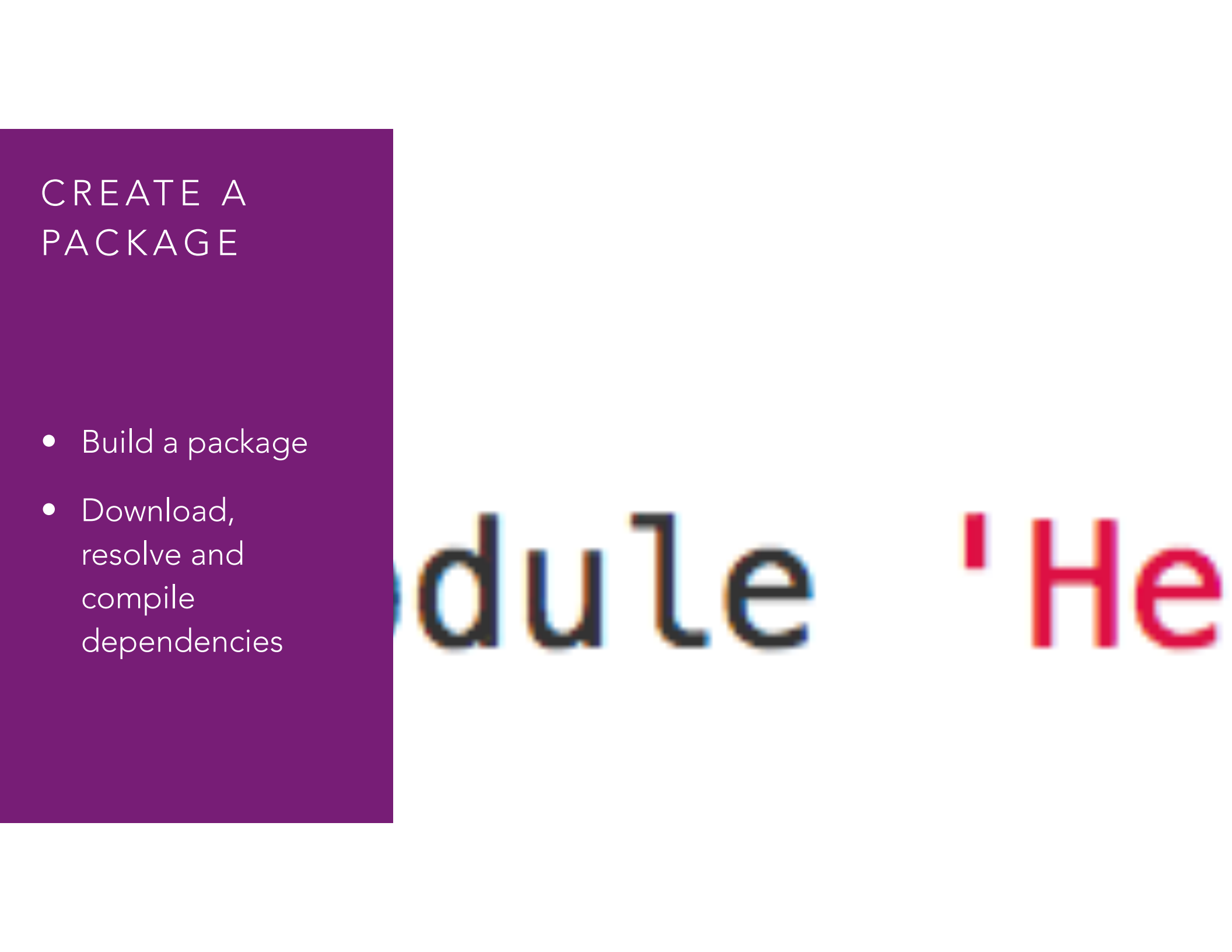


CREATE A PACKAGE

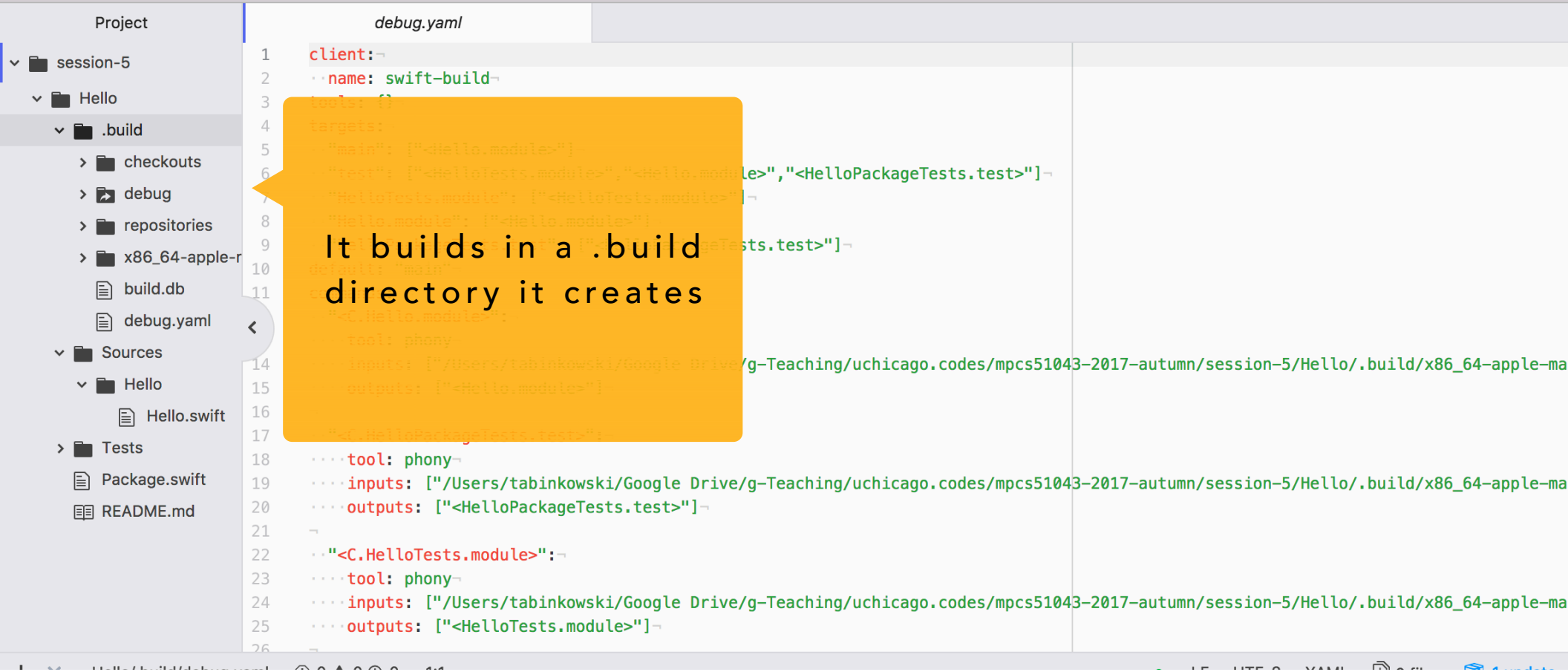


CREATE A PACKAGE

- Build a package
- Download, resolve and compile dependencies



CREATE A PACKAGE



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
```

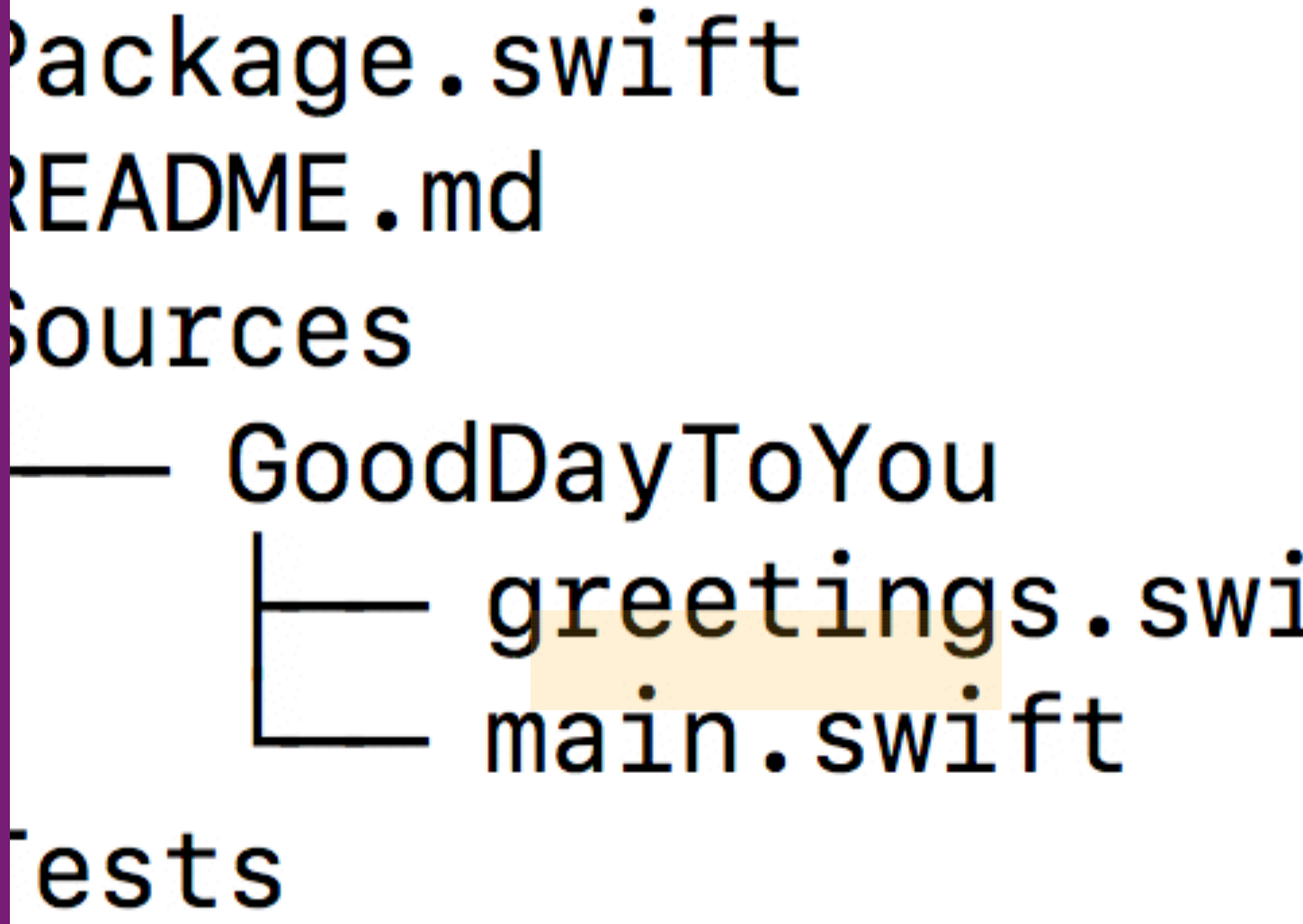
CREATE A PACKAGE

- 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

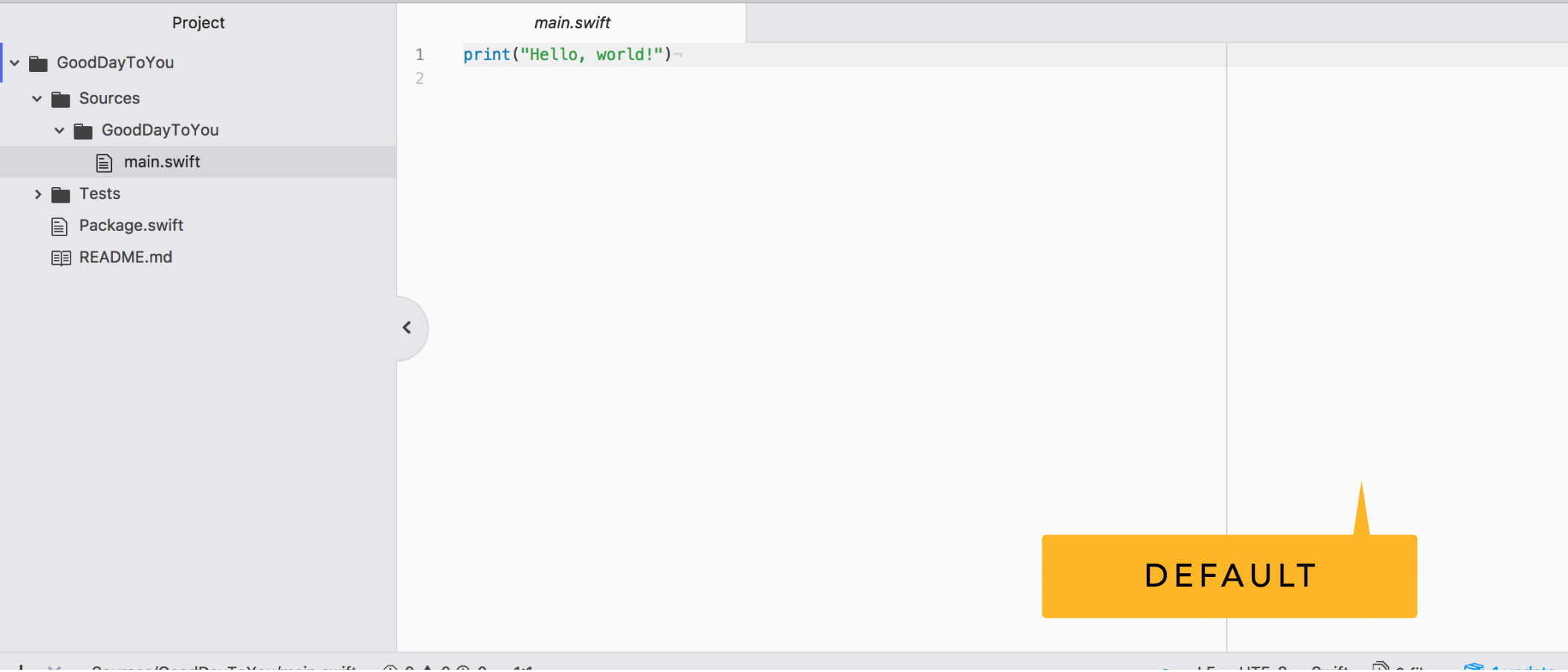


BUILD AN EXECUTABLE

- Create an executable package

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

BUILD AN EXECUTABLE



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
% swift run GoodDayToYou
```

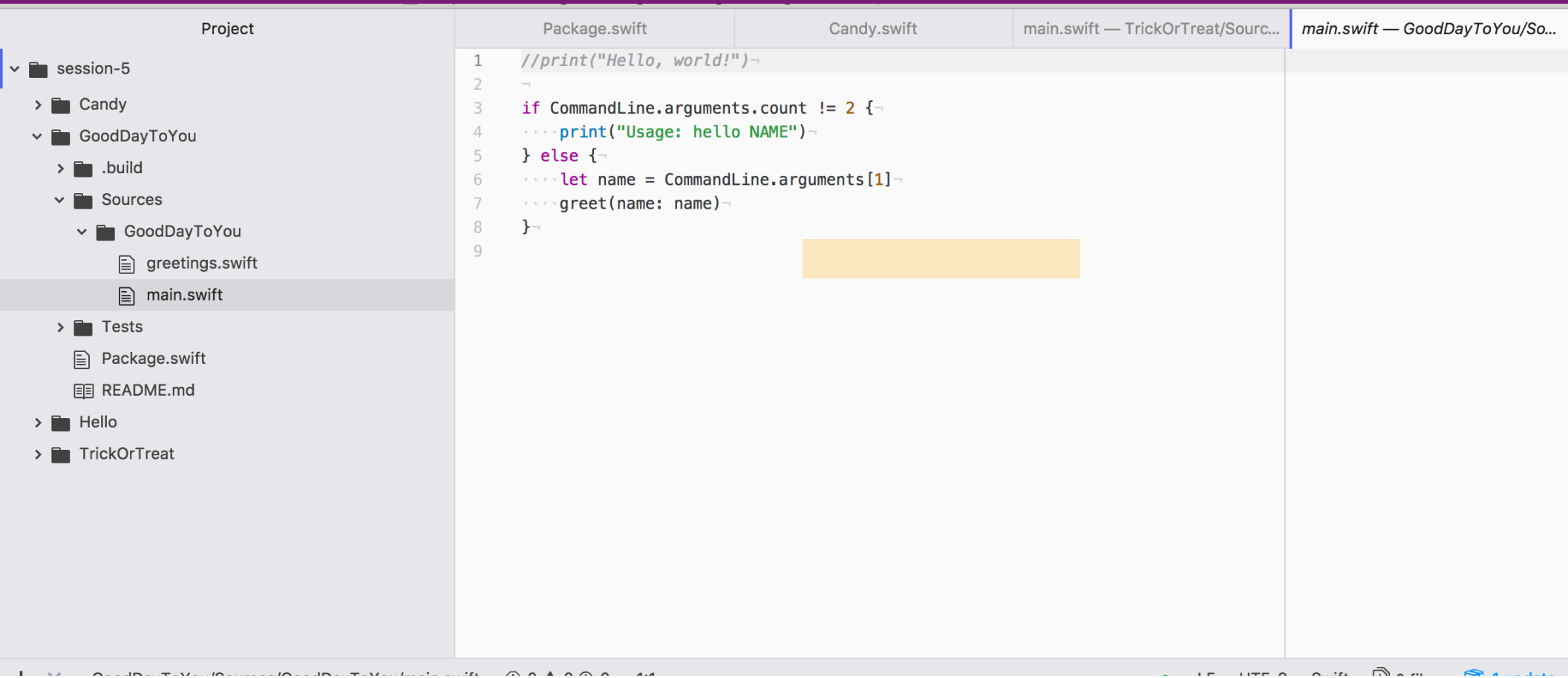
This is the location of the executable

CREATE A PACKAGE

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

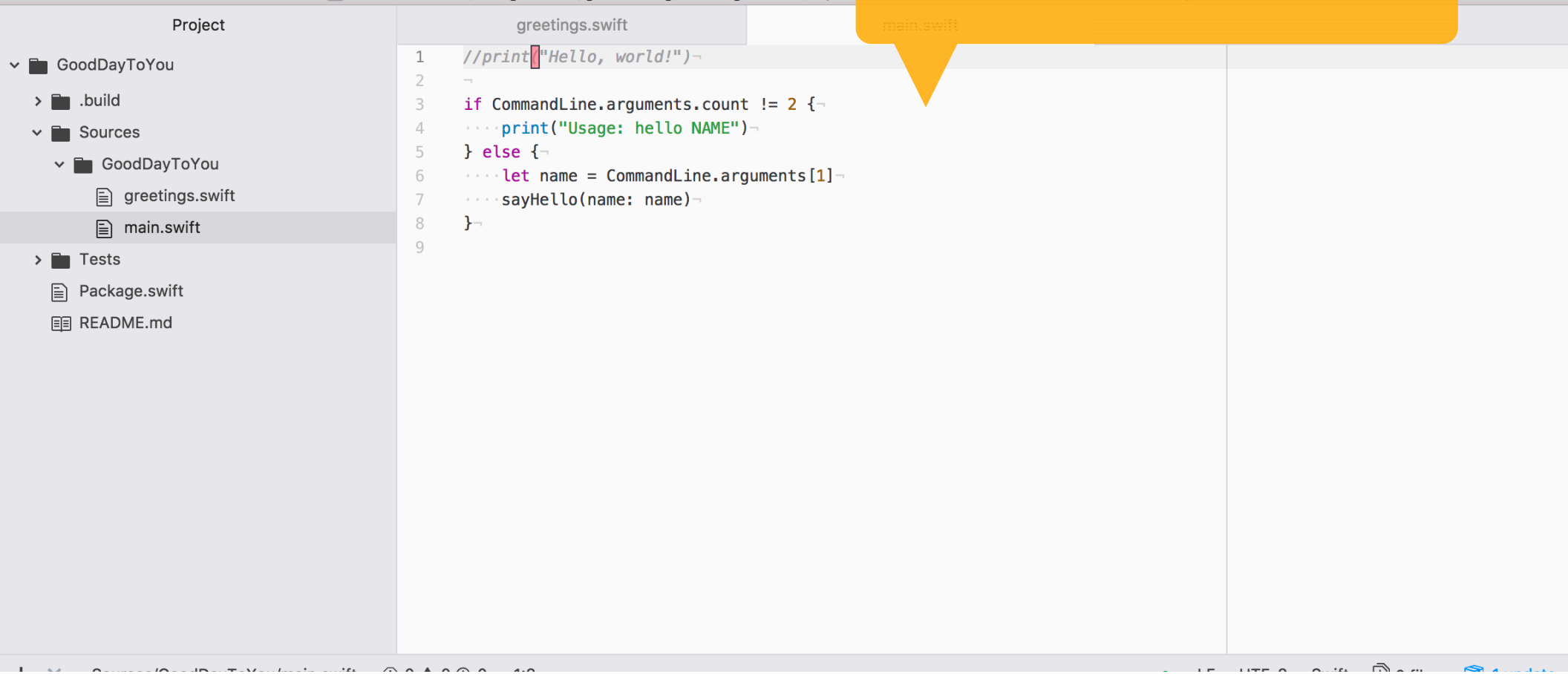
CREATE A PACKAGE

CREATE A PACKAGE



CREATE A PACKAGE

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



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/mpcs51043-2017-autumn/session-5/GoodDayToYou
589 % swift run GoodDayToYou Andrew
Good day to you Andrew 😀.
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2017-autumn/session-5/GoodDayToYou
590 %
```

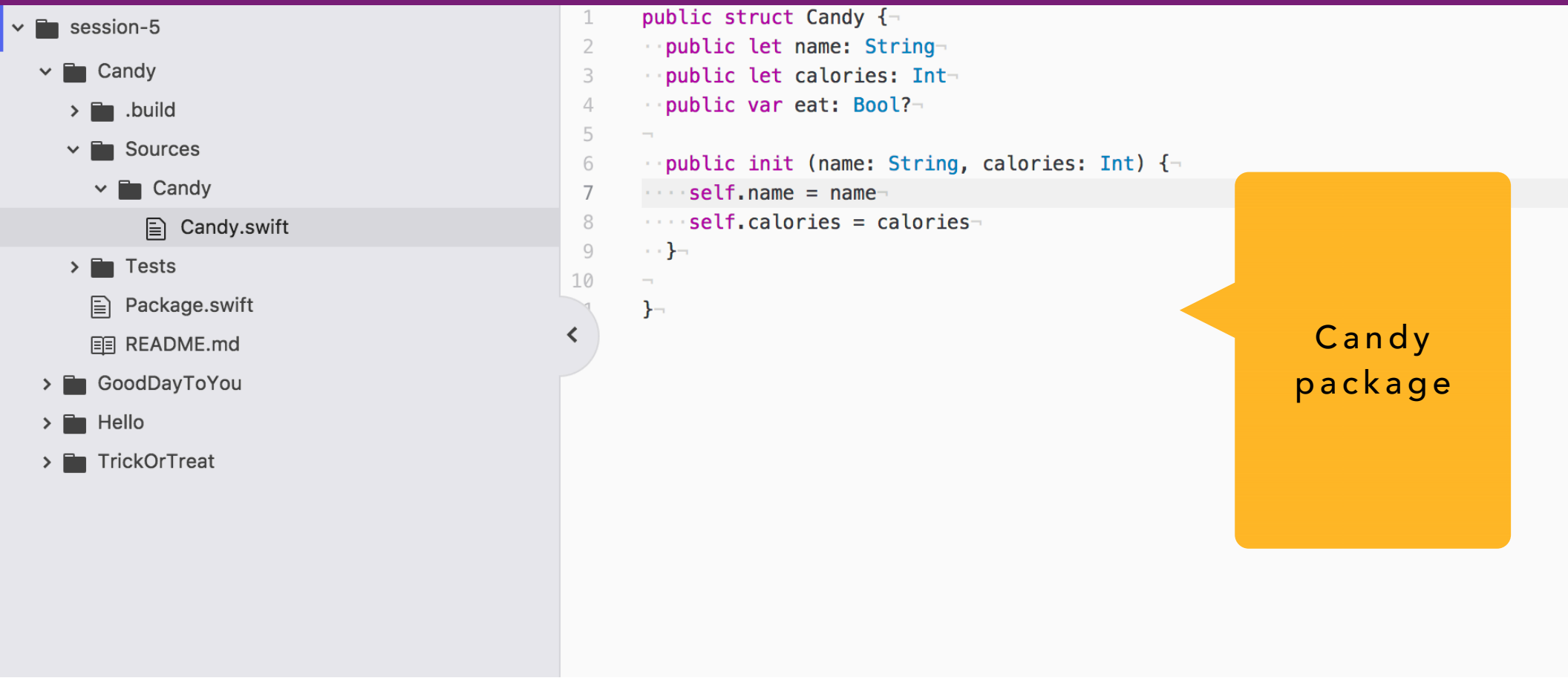
EXECUTABLES WITH PACKAGES

EXECUTABLES WITH PACKAGES

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

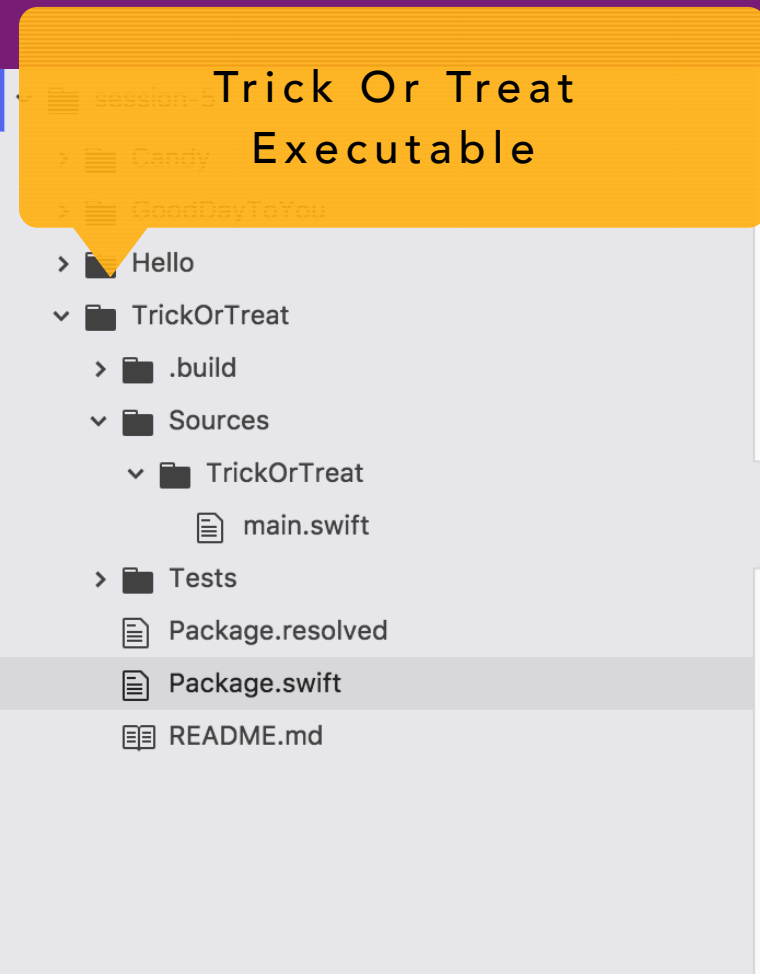
```
# Build it
% swift build
Compile Swift Module 'GoodDayToYou' (1 sources)
Linking ./.build/x86_64-apple-macosx10.10/
debug/GoodDayToYou

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

EXECUTABLES WITH PACKAGES



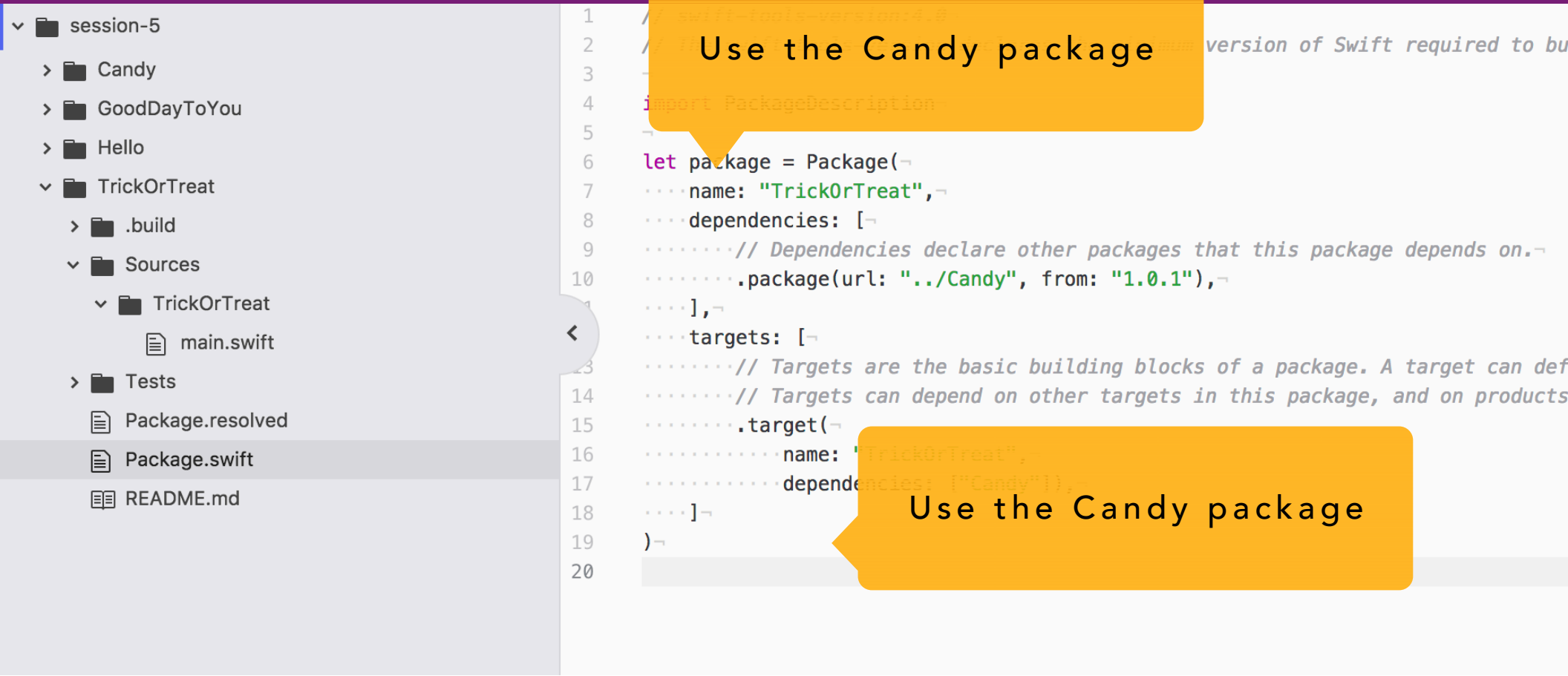
EXECUTABLES WITH PACKAGES

Trick Or Treat Executable

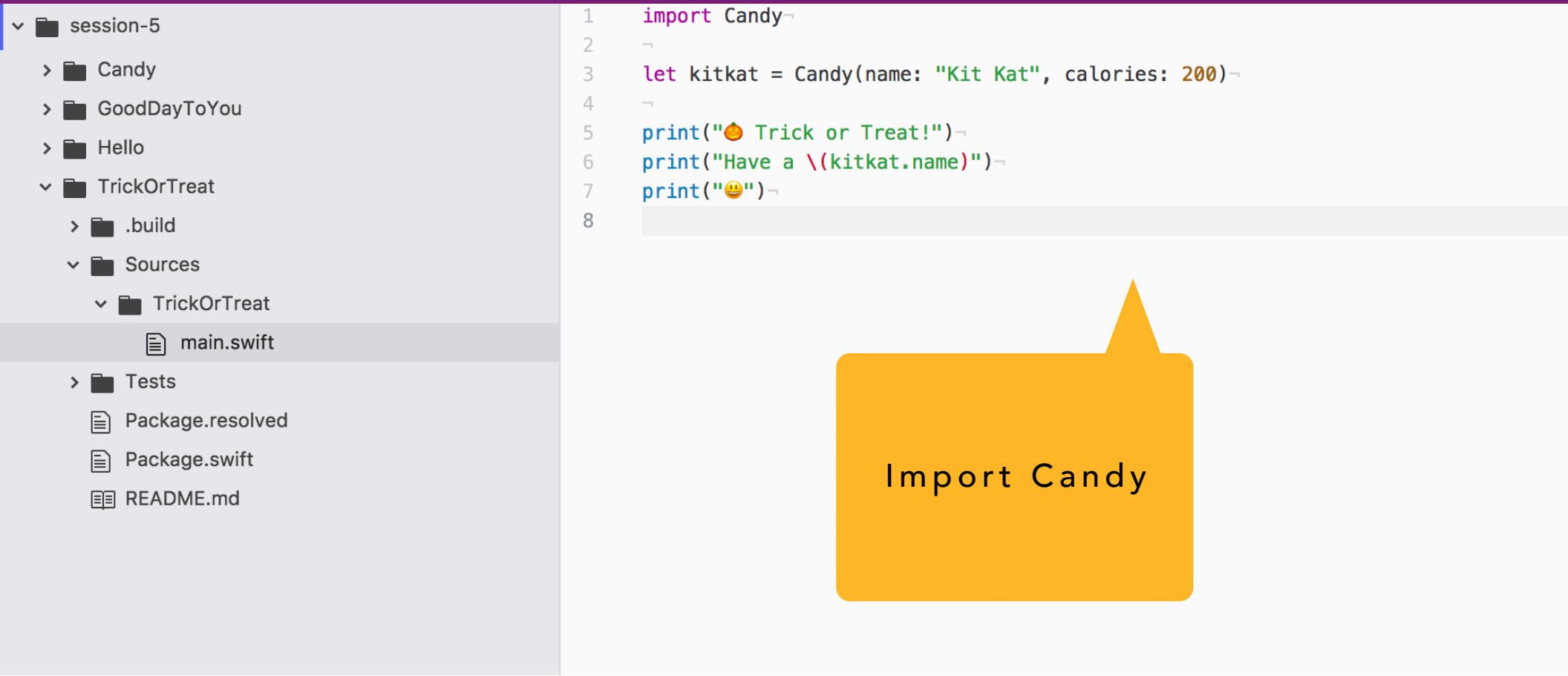



```
// swift-tools-version:4.0
// The swift-tools-version declares the minimum version of Swift required to bu

import PackageDescription

let package = Package(
    name: "TrickOrTreat",
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(url: "../Candy", from: "1.0.1"),
    ],
    targets: [
        // Targets are the basic building blocks of a package. A target can def
        // Targets can depend on other targets in this package, and on products
        .target(
            name: "TrickOrTreat",
            dependencies: ["Candy"]),
    ]
)
```

EXECUTABLES WITH PACKAGES



EXECUTABLES WITH PACKAGES

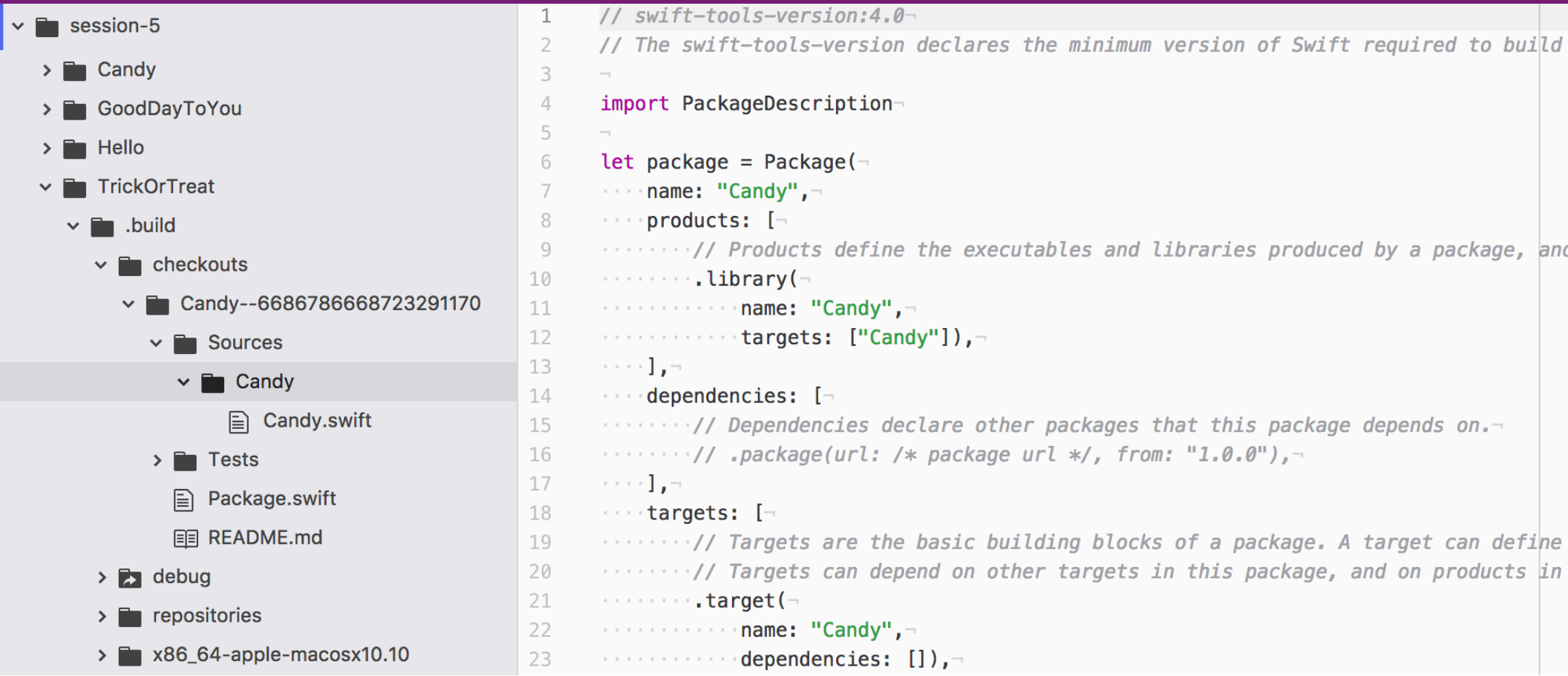


EXECUTABLES WITH PACKAGES

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



WITH XCODE

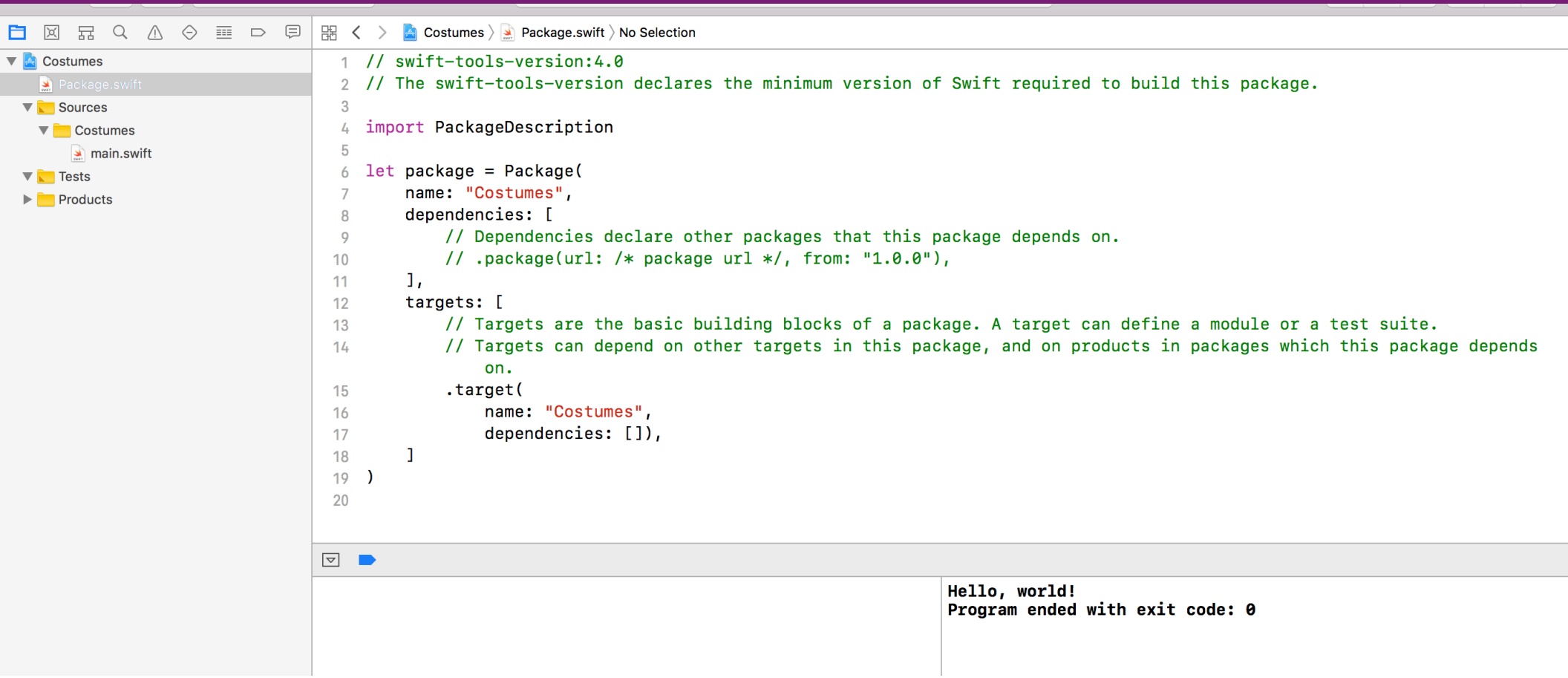
WITH XCODE

Costumes — -bash — 93×14

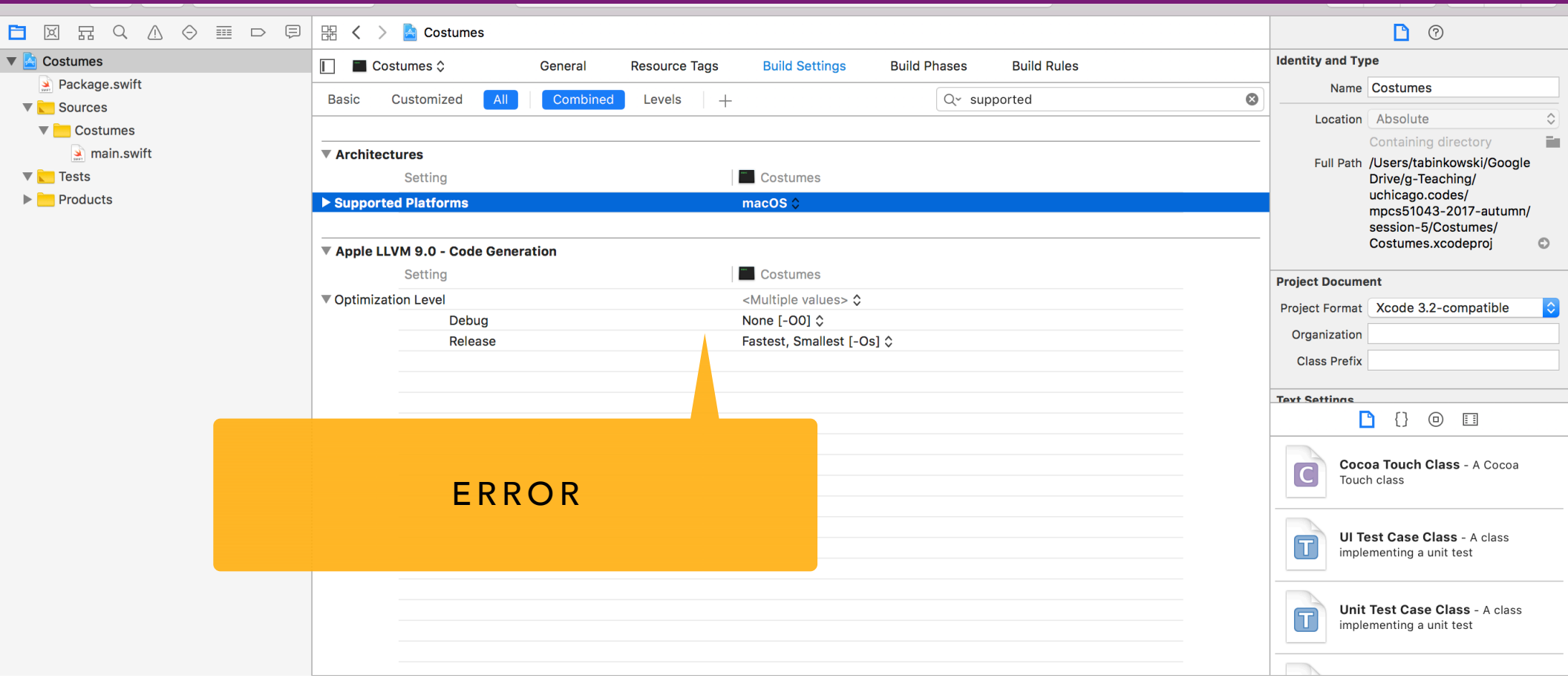
es/firebase/functions — -bash | ...1033-2017-autumn-playground — -bash | ...mn/session-5/GoodDayToYou — -bash | ...-autumn/session-5/Costumes

```
wift 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
wift package generate-xcodeproj
ed: ./Costumes.xcodeproj
wski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.codes/mpcs51043-2017-aut
Costumes
```

WITH XCODE


Costumes
Package.swift
No Selection
Costumes
Package.swift
Sources
Costumes
main.swift
Tests
Products
// swift-tools-version:4.0
// The swift-tools-version declares the minimum version of Swift required to build this package.
import PackageDescription
let package = Package(
name: "Costumes",
dependencies: [
// Dependencies declare other packages that this package depends on.
// .package(url: /* package url */, from: "1.0.0"),
],
targets: [
// Targets are the basic building blocks of a package. A target can define a module or a test suite.
// Targets can depend on other targets in this package, and on products in packages which this package depends on.
.target(
name: "Costumes",
dependencies: []),
]
)
Hello, world!
Program ended with exit code: 0

WITH XCODE


Costumes
Package.swift
Sources
Costumes
main.swift
Tests
Products
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
<Multiple values>
Debug
None [-O0]
Release
Fastest, Smallest [-Os]
ERROR
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
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

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:

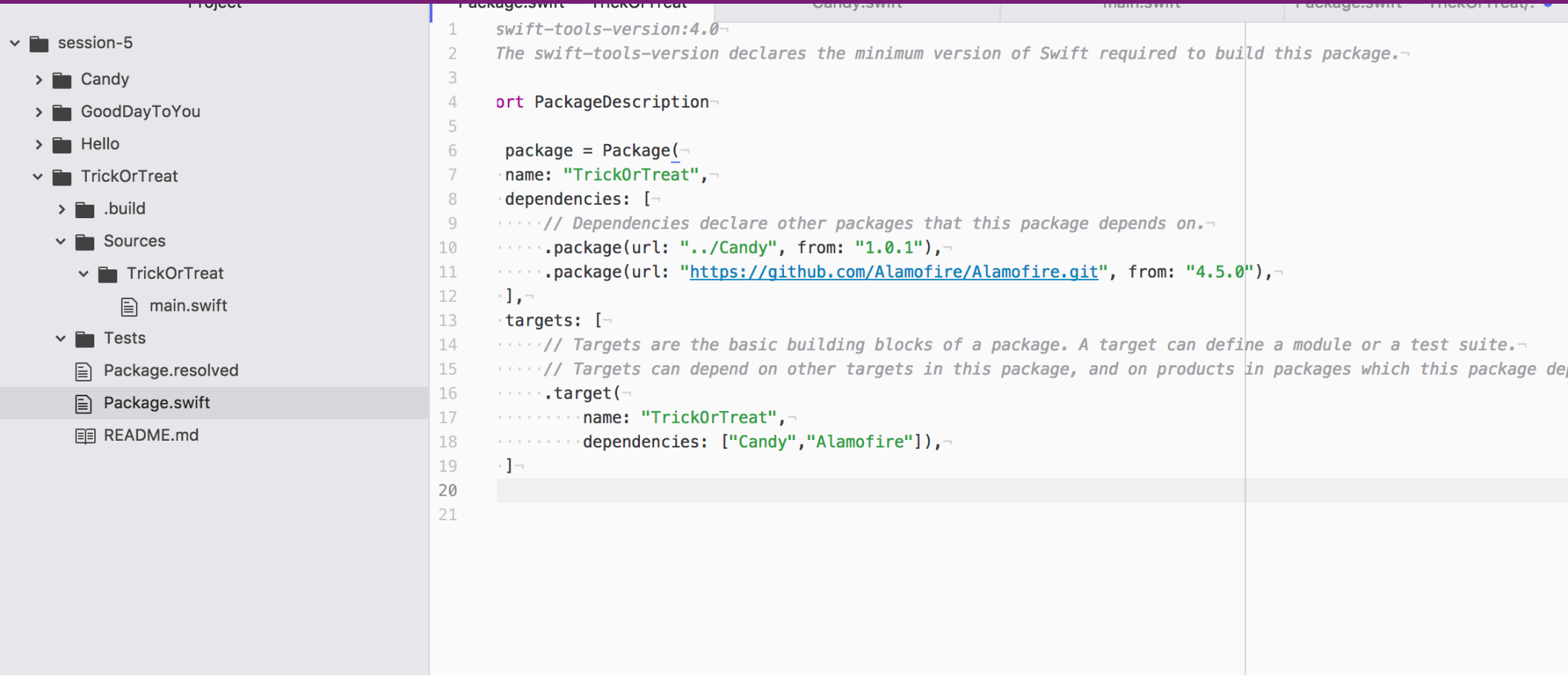
```
$ brew update
$ brew install carthage
```

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



THIRD PARTY PACKAGES

awesome-lists ios linux

2,591 commits **2** branches **0** releases **387** contributors CC0-1.0

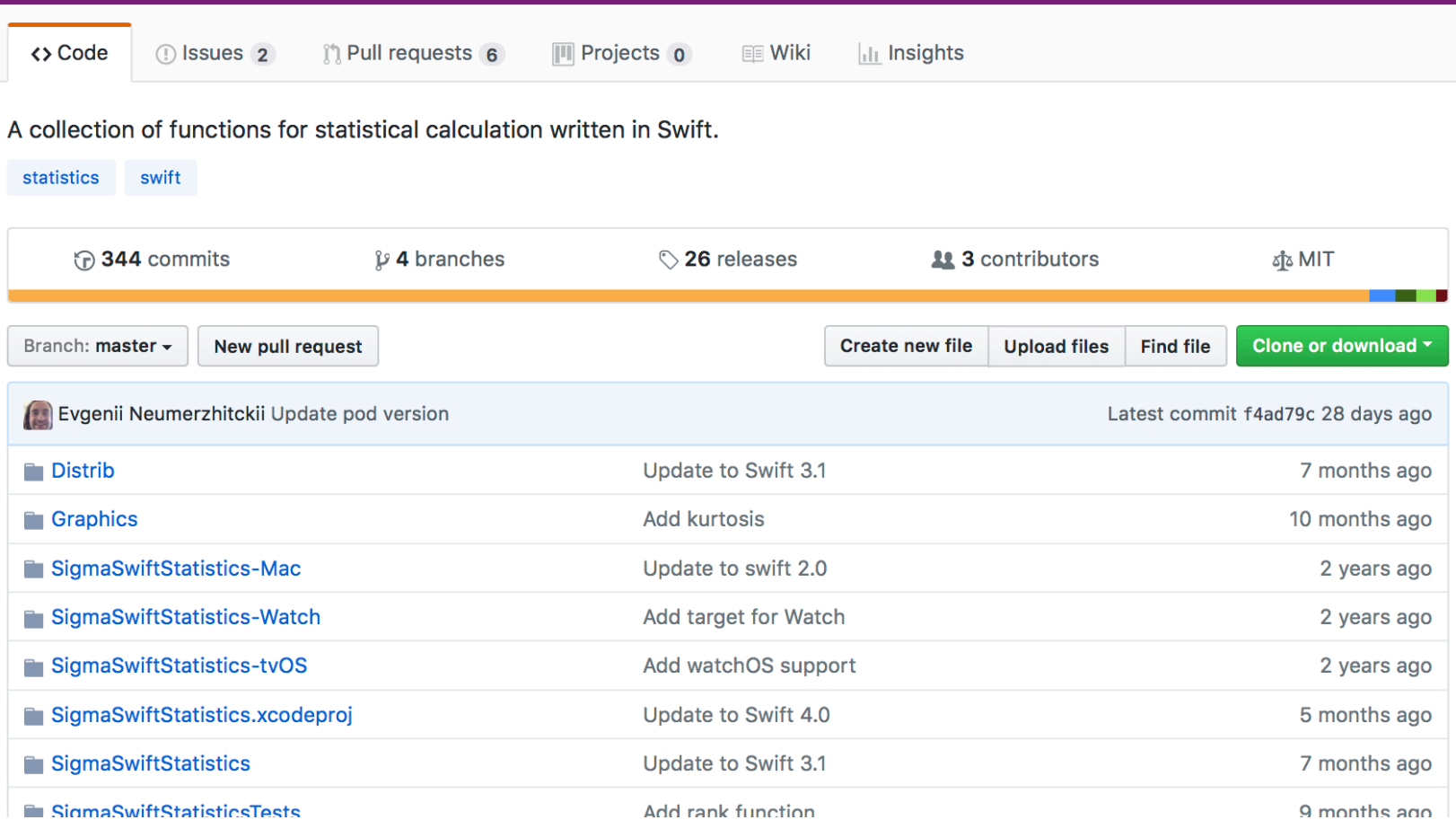
Branch: **master** New pull request Create new file Upload files Find file Clone or download

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

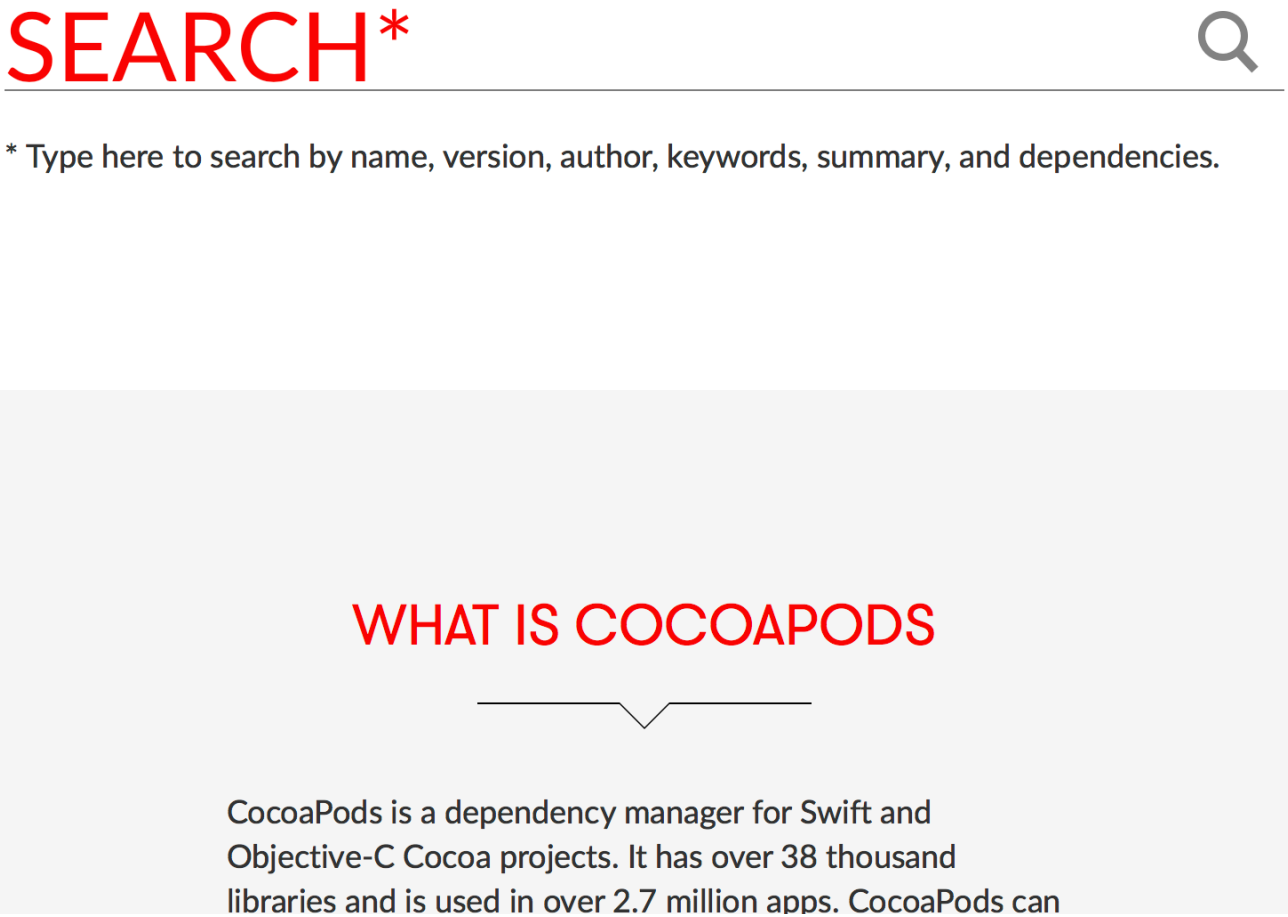
THIRD PARTY PACKAGES



OTHER PACKAGE MANAGERS

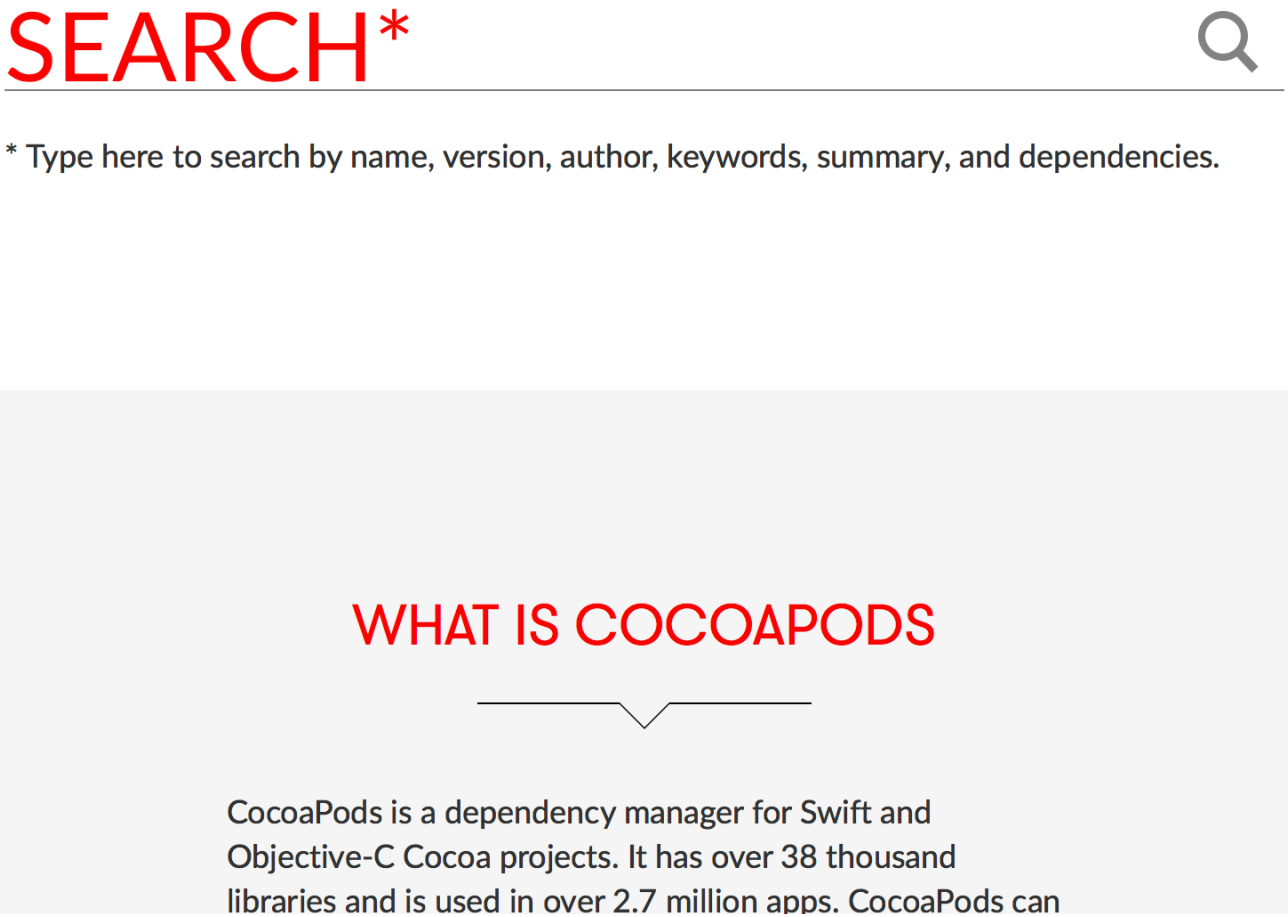
OTHER PACKAGE MANAGERS

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



OTHER PACKAGE MANAGERS

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



OTHER PACKAGE MANAGERS

- https://www.youtube.com/watch?v=iEAjvNRdZa0

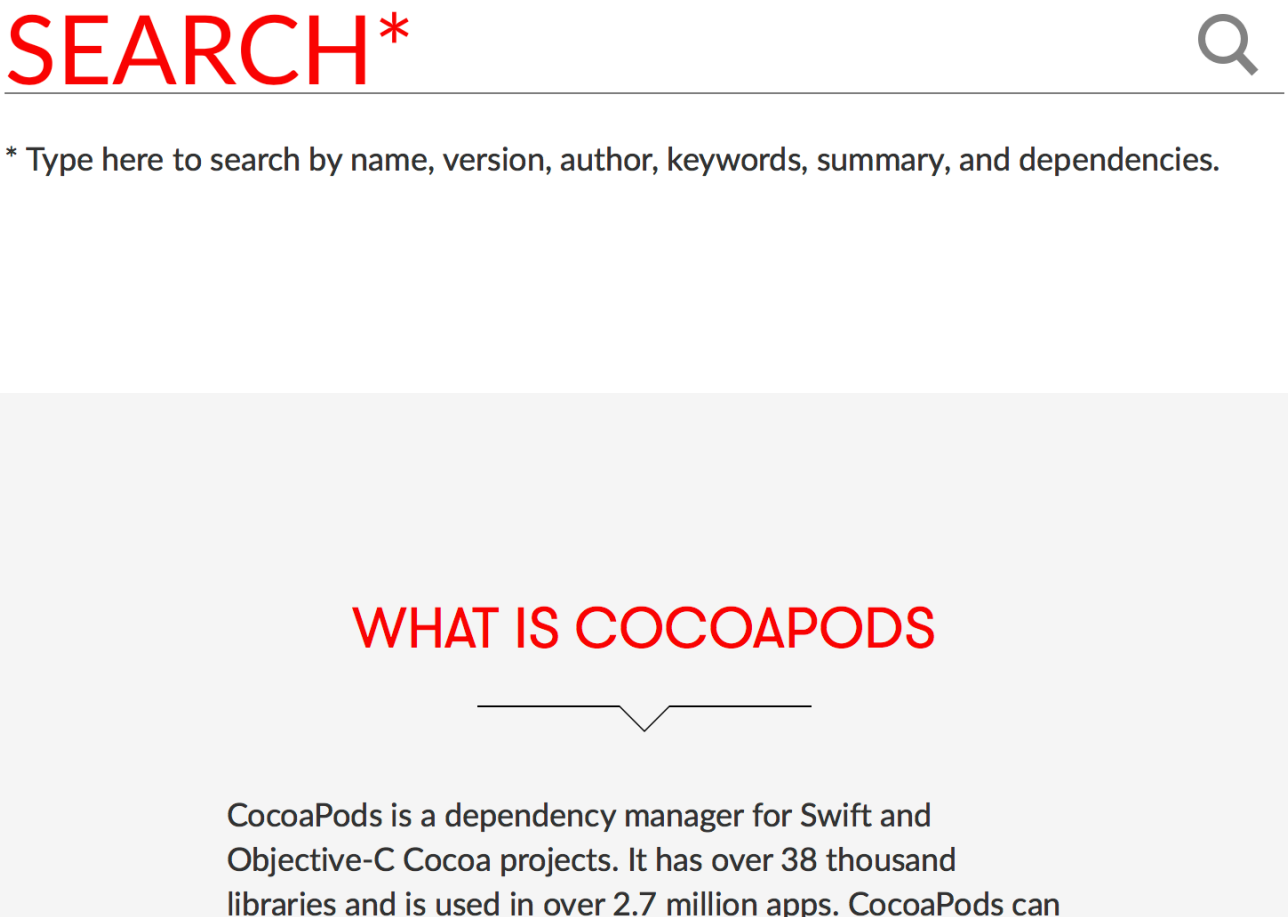
ntroduction to CocoaPods (Route 85)

5 views

849 11 SHARE

OTHER PACKAGE MANAGERS

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



OTHER PACKAGE MANAGERS

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

Carthage

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