Vapor 101

Steffen D. Sommer

Lead Vapor Developer @Nodes

- iOS: 3 years
- Vapor: 1,5 months
- Swift = 👌
- Twitter: @steffendsommer

Nodes ❤️ Vapor

- Mar 2016: Nodes invests in Vapor
- Sep 2016: First customer project in Vapor at Nodes
- 2017+: All new projects in Vapor

Project quickstart

Installing the Toolbox

(Remember to install Xcode)

```
curl -sL check.vapor.sh | bash
curl -sL toolbox.vapor.sh | bash
vapor --help
```

Creating the project

```
vapor new HelloWorld
cd HelloWorld
vapor xcode -y
```

Core concepts

Droplets

One type to rule them all: Droplet

- Server/Client
- Routing
- Middlewares
- Providers
- Configuration
- .. and a lot more

A simple Droplet instance

```
import Vapor

let drop = Droplet()

drop.run()
```

Models

The Model **protocol**

- NodeInitializable
- NodeRepresentable
- Preparation

.. but what is Node?

```
public enum Node {
    case null
    case bool(Bool)
    case number(Number)
    case string(String)
    case array([Node])
    case object([String: Node])
    case bytes([UInt8])
}
```

A simple Post model

```
final class Post: Model {
    var id: Node?
    var title: String
    var content: String?
}
```

The NodeInitializable protocol

```
init(node: Node, in context: Context) throws {
    id = try node.extract("id")
    title = try node.extract("title")
    content = node["content"]?.string
}
```

The NodeRepresentable protocol

```
func makeNode(context: Context) throws -> Node {
    return try Node(node: [
        "id": id,
        "title": title,
        "content": content
    ])
}
```

The Preparation **protocol**

```
static func prepare(_ database: Database) throws {
    try database.create(entity) {
        $0.id()
        $0.string("title")
        $0.string("content")
    }
}

static func revert(_ database: Database) throws {
    try database.delete(entity)
}
```

Fluent **and the** Entity **protocol**

- Saving
- Deleting
- Querying
- .. and more

Controllers

Index

```
func index(request: Request) throws -> ResponseRepresentable {
    return try Post.all().makeJSON()
}
```

Create

```
func create(request: Request) throws -> ResponseRepresentable {
    guard let json = request.json else { throw Abort.badRequest }
    var post = try Post(node: json)
    try post.save()
    return post
}
```

Show

```
func show(request: Request, post: Post) throws -> ResponseRepresentable {
    return post
}
```

Delete

```
func delete(request: Request, post: Post) throws -> ResponseRepresentable {
    try post.delete()
    return JSON([:])
}
```

Update

```
func update(request: Request, post: Post) throws -> ResponseRepresentable {
    guard let json = request.json else { throw Abort.badRequest }
    let new = try Post(node: json)
    var post = post
    post.content = new.content
    post.title = new.title
    try post.save()
    return post
}
```

The ResourceRepresentable protocol

```
func makeResource() -> Resource<Post> {
    return Resource(
        index: index,
        store: create,
        show: show,
        modify: update,
        destroy: delete
    )
}
```

Routes

Routes with ResourceRepresentable

```
drop.resource("posts", PostController())
```

Routes without ResourceRepresentable

```
drop.get("posts", handler: PostController().index)
drop.get("posts", Post.self, handler: PostController().show)
drop.post("posts", handler: PostController().create)
```

Routes for websites

```
drop.get("posts/view") { request in
    return try drop.view.make("posts", ["posts": try Post.all().makeNode()])
}
```

Route groups

```
drop.group("v1") { v1 in
    v1.get("posts", handler: PostController().index)
}
```

Views

The Leaf template language

```
<!DOCTYPE html>
<html>
<head>
    <title>Posts</title>
</head>
    <body>
        <ul>
            #loop(posts, "post") {
                #(post.title)!
            }
        </ul>
    </body>
</html>
```

Providers

A simple way of adding functionality to your project

- Add the dependency to **Package.swift**
- Import the module
- Add the provider to your **Droplet** instance
- * Some providers requires adding a config file

1/3: Adding persistence using MySQL

Add dependency to **Package.swift**

```
.Package(url: "https://github.com/vapor/mysql-provider.git", majorVersion: 1)
```

2/3: Adding persistence using MySQL

Import and add it to your **Droplet** instance

```
import VaporMySQL

let drop = Droplet()

try drop.addProvider(VaporMySQL.Provider.self)
```

3/3: Adding persistence using MySQL

Add Config/mysql.json

```
{
    "host": "127.0.0.1",
    "user": "root",
    "password": "",
    "database": "mydb"
}
```

Middlewares

Client <-> Middleware <-> Droplet

Examples of middlewares

- Authentication
- Error reporting/handling
- Data transformation
- Header enforcing/manipulation
- .. etc.

Adding a middleware

```
import MyErrorMiddleware

let drop = Droplet()

drop.middleware.append(MyErrorMiddleware())
```

1,5 months in: Gotchas

1: Xcode project and tests

- To run tests on Linux you need to have multiple targets
- See https://vapor.github.io/documentation/testing/modules.html for guide
- Use https://github.com/nodes-vapor/template as template for automatic setup

2a: Linux tests

```
class AppLogicTests: XCTestCase {
    static var allTests = [
        ("testExample", testExample)
    ]

    func testExample() throws {
        // Just a dummy test to satisfy GitLab CI.
        XCTAssert(true)
    }
}
```

2b: Linux tests

```
XCTMain([
    testCase(AppLogicTests.allTests),
])
```

See https://github.com/BrettRToomey/toolbox/tree/test-fix-command if you feel #yolo

3: Database transactions

```
func deletePostTransaction(post: Post) throws {
    guard let driver = driver else {
        throw Error.noMySQLDriver
    }

    guard let postId = post.id else {
        throw Error.postIdNotFound
    }

    let connection = try driver.database.makeConnection()
    try connection.transaction {
        // Delete post.
        try connection.execute(
            "UPDATE posts SET deleted_at = ? where id = ?",
            [Date().mysql, postId]
        )

        try connection.execute(
            "UPDATE posttags SET deleted_at = ? WHERE post_id = ?",
            [Date().mysql, postId]
        )
    }
}
```

4: The exists variable

- Required by Fluent to keep track of if a model should be updated or created on **save()**
- When querying through Fluent, the returned models will have the property set
- When doing raw queries and creating models from **Node**'s, remember to set the property

5: Migrations

```
final class PostAddAuthorColumn: Preparation {
    static func prepare(_ database: Database) throws {
        try database.driver.raw("ALTER TABLE posts ADD COLUMN author VARCHAR(255);")
    }

    static func revert(_ database: Database) throws {
        try database.driver.raw("ALTER TABLE posts DROP COLUMN author;")
    }
}
```

Remember to append to the **Droplet** instance:

```
drop.preparations.append(PostAddAuthorColumn.self)
```

6a: The type infer madness

```
func makeNode(context: Context) throws -> Node {
    return try Node(node: [
        "id": id,
        "title": title,
        "content": content,
    ])
}
```

6b: The type infer madness

```
func makeNode(context: Context) throws -> Node {
    return try Node(node: [
        "id": id,
        "title": title,
        "content": content,
        "createdAt": createdAt,
        "deletedAt": deletedAt,
        "updatedAt": updatedAt,
        "tags": tags,
        "authorId": authorId,
        "totalComments": totalComments,
        "type": type,
        "authorsOnly": authorsOnly,
        "relatedPost": relatedPost,
        "coAuthorId": coAuthorId,
        "summary": summary,
        "subtitle": subtitle
    ])
}
```

6c: The type infer madness

```
func makeNode(context: Context) throws -> Node {
    var node: [String: NodeRepresentable] = [:]
    node["id"] = id
    node["title"] = title
    node["content"] = content
    node["createdAt"] = createdAt
    node["deletedAt"] = deletedAt
    node["updatedAt"] = updatedAt
    node["tags"] = tags
    node["authorId"] = authorId
    node["totalComments"] = totalComments
    node["type"] = type
    node["authorsOnly"] = authorsOnly
    node["relatedPost"] = relatedPost
    node["coAuthorId"] = coAuthorId
    node["summary"] = summary
    node["subtitle"] = subtitle
    return try Node(node: node)
}
```

7: **macOS** Foundation **!= Linux** Foundation

- Be careful of what you are using from **Foundation**
- Keep an eye on https://github.com/apple/swift-corelibs-foundation for status on the port of **Foundation**

Useful resources

- Vapor Docs: http://docs.vapor.codes
- The Vapor Source code: https://github.com/vapor
- #help on Slack (qutheory.slack.com)
- Vapor @Nodes: https://github.com/nodes-vapor

Questions?