Streamlining our CD flow using ServerSide Swift, Fastlane and Travis-CI

Jari Koopman
iOS Developer @ Wehkamp

Streamlining our CD flow

- About me
- The problem
- The solution

About me

About me

@LotUDev

/MrLotU

/in/jari-koopman/

Streamlining our CD flow

The problem

The problem

The problem

Before

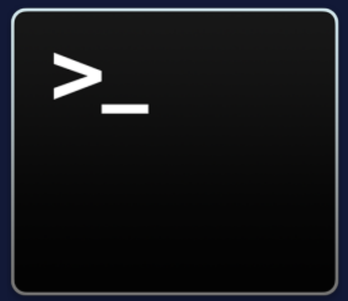
The problem

Before

The problem

Before

After



The problem

Before

The problem

Before

The problem

Before




After




The problem

Before




The problem

Before




The problem

Before




Current






The problem

Current

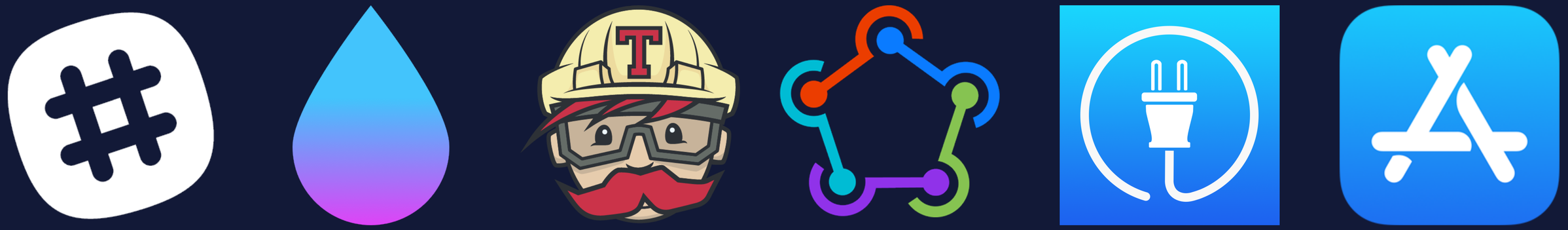





Streamlining our CD flow

The solution

The solution



The solution

The solution

Slack

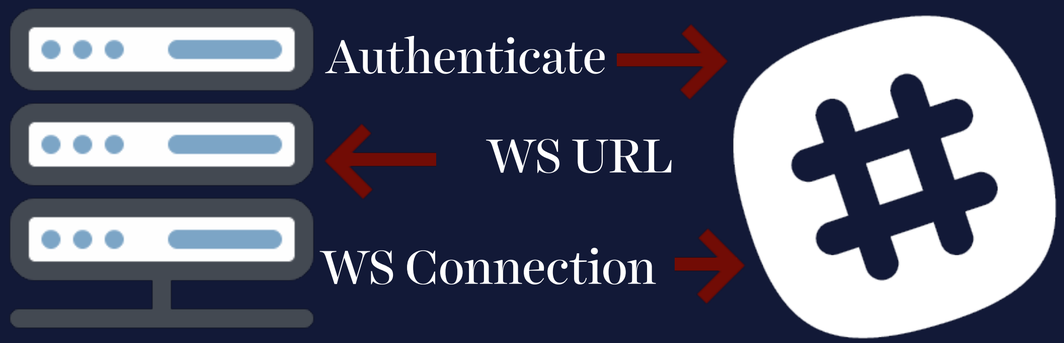
- Bot users
- API’s
  - RTM API
  - Events API

Slack

- Bot users
- API’s
  - RTM API
  - Events API

Slack

- Bot users
- API’s
  - RTM API



Slack

- Bot users
- API's
  - RTM API
  - Events API

Slack

- Bot users
- API’s
  - RTM API
  - Events API

The solution

The solution

Vapor

- Connect to the RTM API
- Parse incoming data
- Respond to “commands”
- Make the magic happen

Vapor

- Connect to the RTM API
- Parse incoming data
- Respond to “commands”
- Make the magic happen

Vapor

- Connect to the RTM API
- Parse incoming data

```
struct RTMSinPayload: Content {
    var type: String
    var subtype: String?

    var eType: SlackEvent {
        return SlackEvent(rawValue: type) ?? .unknown
    }
}
```

```
func onText(ws: WebSocket, text: String) {
    if text.contains("bot_id") { return }
    if let jsonData = text.data(using: .utf8) {
        guard let sinData = try? JSONDecoder().decode(RTMSinPayload.self, from: jsonData) else { print("Couldn't decode SinData"); return }
        switch sinData.eType {
        case .hello:
            print("Hello recieved")
        case .message:
            guard let messageData = try? JSONDecoder().decode(RTMMessageData.self, from: jsonData) else { return }
            self.cmdHandler?.handle(messageData.text ?? "", withData: messageData)
        case .userTyping, .userChange:
            break // We don't care about this
        default:
            print(sinData.type)
        }
    }
}
```

Vapor

- Connect to the RTM API
- Parse incoming data

```
sock.cmdHandler?.register(Command(triggers: ["ping"], execute: { (socket, args, event) in
    socket.sendMessage(channel: event.channel, text: "pong")
}))
```

```
func register(_ command: Command) {
    self.commands.append(command)
}
```

```
func handle(_ message: String, withData data: RTMMessageData) {
    guard message.starts(with: prefix) else { return }
    let message = String(message[message.range(of: prefix)!.upperBound...])
    let components = message.split(separator: " ")
    commands.forEach { (command) in
        if command.triggers.contains("\(components.first ?? "")") {
            self.RTMSocket.logger.info("Command \(components.first ?? "") executed by user <@\(data.user ?? "")>")
            command.execute(self.RTMSocket, components.compactMap { if $0 == components.first { return nil } else { return "\($0)"} }, data)
        }
    }
}
```

Vapor

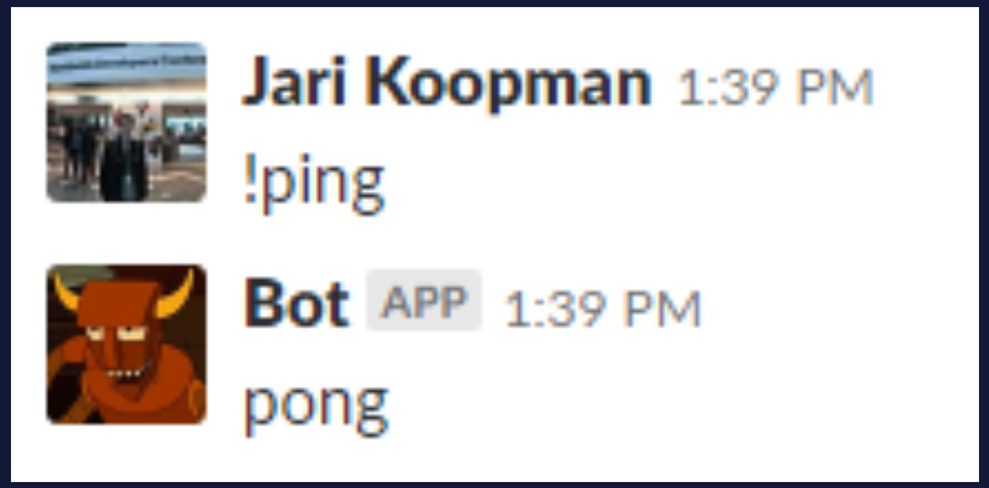
- Connect to the RTM API
- Parse incoming data
- Respond to “commands”

```
sock.cmdHandler?.register(Command(triggers: ["ping"], execute: { (socket, args, event) in
    socket.sendMessage(channel: event.channel, text: "pong")
}))
```

```
func sendMessage(channel: String, text: String) {
    var headers = HTTPHeaders()
    headers.add(name: "Authorization", value: "Bearer Token :D")
    _ = self.client.post("https://slack.com/api/chat.postMessage", headers: headers) { req in
        let data = SlackMessageBody(channel: channel, text: text, as_user: true)
        try req.content.encode(data)
    }
}
```

Vapor

- Connect to the RTM API
- Parse incoming data
- Respond to “commands”



Vapor

- Connect to the RTM API
- Parse incoming data
- Respond to “commands”
- Make the magic happen

Vapor

- Make the magic happen

```
func sendTravisCommand(branch: String, script: String, message: String, repo: String) {
    var headers = HTTPHeaders()
    headers.add(name: .authorization, value: "token Token :D")
    headers.add(name: .contentType, value: "application/json")
    headers.add(name: .accept, value: "application/json")
    headers.add(name: "Travis-API-Version", value: "3")
    _ = self.client.post("https://api.travis-ci.com/repo/\(repo)/requests", headers: headers, beforeSend: { (req) in
        let body = TravisData(branch: branch, script: script, message: message)
        try req.content.encode(body)
    }).map { response in
        guard response.http.status.code > 200 &&
              response.http.status.code < 300 else {
                self.logger.warning("Travis API request failed! Check logs for more info"); print(response)
                return
        }
    }
}
```

The solution

The solution

Travis CI 

- What’s Travis
- Why Travis
- Travis VS Jenkins
- Separation of concerns

Travis CI 

- What's Travis
- Why Travis
- Travis VS Jenkins
- Separation of concerns

```
branches:
  only:
    - myCoolBranch
    - myOtherBranch

language: swift
xcode_workspace: yourProject.xcworkspace
xcode_scheme: yourScheme
os: osx
osx_image: xcode9.4
before_install:
  - pod repo update
script:
  - gem install xcpretty && \
    xcodebuild clean build -sdk iphonesimulator -workspace yourProject.xcworkspace -scheme yourScheme | xcpretty
```

Travis CI 

- What's Travis
- Why Travis
- Travis VS Jenkins
- Separation of concerns

Travis CI 

- What's Travis
- Why Travis
- Travis VS Jenkins
- Separation of concerns

Travis CI 

- What's Travis
- Why Travis
- Travis VS Jenkins
- Separation of concerns

App	Metadata
Code	Fastlane
travis.yml	travis.yml

The solution

The solution

Fastlane 

- What’s Fastlane
- Configuring Fastlane

Fastlane 

- What's Fastlane
- Configuring Fastlane

Fastlane 

- What's Fastlane
- Configuring Fastlane

```
desc "Submit a new Beta Build to Apple TestFlight"
lane :beta do
  build = increment_build_number(build_number: ENV["TRAVIS_BUILD_NUMBER"])
  gym(
    scheme: "yourScheme",
    clean: true,
    export_xcargs: "-allowProvisioningUpdates",
  )
  pilot(skip_waiting_for_build_processing: true, changelog: ENV["TRAVIS_COMMIT_MESSAGE"])
  add_git_tag(build_number: build)
  push_git_tags
  slack(message: "App successfully added to TestFlight!")
end
```

```
desc "Deploy a new version to the App Store"
lane :submit do |params|
  deliver(
    force: true,
    submit_for_review: true,
    build_number: params[:build_number],
    app_version: params[:app_version],
    overwrite_screenshots: true,
    phased_release: params[:phased],
    )
  slack(message: "App successfully sumitted for review!")
end
```

```
lane :release do
  Spaceship::Tunes.login("appleid@apple.com")
  app = Spaceship::Tunes::Application.find("com.yoursite.appidentifier")
  version = app.edit_version

  version.release!

  slack(message: "App successfully released to App Store!")
end
```

The solution

Streamlining our CD flow

Now what

Now what

- More integrations!

Streamlining our CD flow using ServerSide Swift, Fastlane and Travis-CI

Jari Koopman
iOS Developer @ Wehkamp

Thanks for listening!