Kotlin Night
Kiev

Ktor

Ruslan Ibragimov

Ktor

Agenda

Why yet another server framework

Hello, World!

Architecture

Features

Modularity

Testing

Configuration

Client

Why

Usable with Kotlin
Async
Coroutines Support

VERT.X

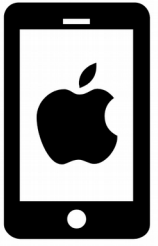
Spark

One more thing...

Multiplatform

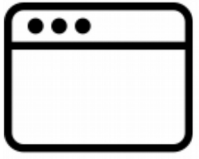
Android
Kotlin/JVM

JVM
Kotlin/JVM



iOS
Kotlin/Native

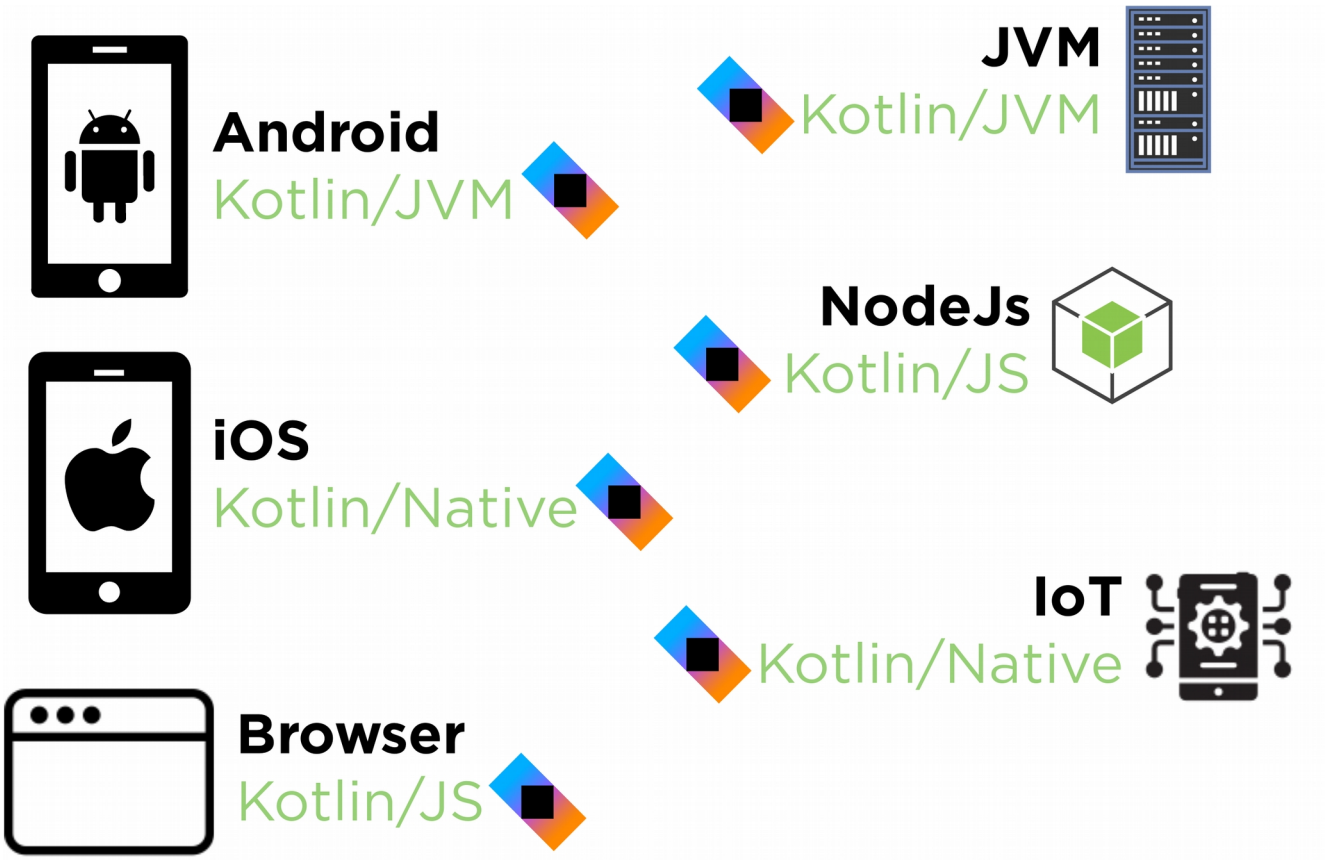
NodeJs
Kotlin/JS



Browser
Kotlin/JS

IoT
Kotlin/Native

Multiplatform



Hello, World!

```
plugins {
    kotlin("jvm") version "1.3.31"
}

repositories {
    jcenter()
}

dependencies {
    implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
    implementation("ch.qos.logback:logback-classic:1.2.1")
    implementation("io.ktor:ktor-server-netty:1.2.1")
    testImplementation("io.ktor:ktor-server-tests:1.2.1")
}
```

Hello, World!

```
fun main() {
    embeddedServer(Netty, 8080) {
        routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

Hello, World!

```
fun main() {
    embeddedServer(Netty, 8080) {
        routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

Hello, World!

```
fun main() {
    embeddedServer(Netty, 8080) {
        routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

Hello, World!

```
fun main() {
    embeddedServer(Netty, 8080) {
        routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

Hello, World!

```
fun main() {
    embeddedServer(Netty, 8080) {
        routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

Configuration

Gradle with Kotlin DSL project

with Wrapper

Server Engine: Netty

Ktor 1.2.1

Group: com.knightkyiv

Name: ktor-playground

Version: 0.0.1-SNAPSHOT

Swagger *(Optional)*: +

Build or Install IntelliJ plugin

Server

Filter Server Features

Documentation

- [ ] PartialContent (ktor-server-core)
  Handles requests with the Range header. Generating Accept-Ranges and the Content-Range headers and slicing the served content when required.
  Documentation

Authentication

- [ ] Authentication Basic (ktor-auth)
  Handle Basic authentication
  Documentation
- [ ] Authentication Digest (ktor-auth)
  Handle Digest authentication
  Documentation
- [ ] Authentication JWT (ktor-auth-jwt)
  Handle JWT authentication
  Documentation
- [ ] Authentication LDAP (ktor-auth-ldap)
  Handle LDAP authentication
  Documentation
- [ ] Authentication OAuth (ktor-auth)
  Handle OAuth authentication
  Documentation
- [ ] Authentication (ktor-auth)
  Handle Basic and Digest HTTP Auth, Form authentication and OAuth 1a and 2
  Documentation

Content Negotiation

- [ ] GSON (ktor-gson)
  Handles JSON serialization using GSON library
  Documentation
- [x] Jackson (ktor-jackson)
  Handles JSON serialization using Jackson library
  Documentation
- [-] ContentNegotiation (ktor-server-core)
  Provides automatic content conversion according to Content-Type and Accept headers.
  Documentation

Sockets

- [ ] Raw Sockets (ktor-network)
  Adds Raw Socket support for listening and connecting to tcp and udp sockets
  Documentation
- [ ] Raw Secure SSL/TLS Sockets (ktor-network-tls)
  Adds Raw Socket support for listening and connecting to tcp and udp sockets with secure sockets
  Documentation

Show marked dependencies only

Client

Filter Client Features

HttpClient Engine

- [-] HttpClient Engine (ktor-client-core, ktor-client-core-jvm)
  Core of the HttpClient. Required for libraries.
  Documentation
- [x] Apache HttpClient Engine (ktor-client-apache)
  Engine for the Ktor HttpClient using Apache. Supports HTTP 1.x and HTTP 2.0.
  Documentation
- [ ] CIO HttpClient Engine (ktor-client-cio)
  Engine for the Ktor HttpClient using CIO (Corroutine I/O). Only supports HTTP 1.x.
  Documentation
- [ ] Jetty HttpClient Engine (ktor-client-jetty)
  Engine for the Ktor HttpClient using Jetty. Only supports HTTP 2.x.
  Documentation
- [ ] Mock HttpClient Engine (ktor-client-mock, ktor-client-mock-jvm)
  Engine for using in tests to simulate HTTP responses programmatically.
  Documentation

Features

- [ ] Auth Basic feature HttpClient (ktor-client-auth-basic)
  Supports basic authentication for the Http Client
  Documentation
- [ ] Json serialization for HttpClient (ktor-client-json-jvm, ktor-client-gson)
  Supports JSON serialization for the Http Client
  Documentation
- [ ] WebSockets HttpClient support (ktor-client-websocket)
  HttpClient feature to establish bidirectional communication using WebSockets
  Documentation
- [ ] Logging feature (ktor-client-logging-jvm)
  Logging feature for debugging client calls
  Documentation
- [ ] User agent feature ()
  User agent header support feature
  Documentation

Show marked dependencies only

Application::class

```
fun main() {
    embeddedServer(Netty, 8080) { this: Application
        routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

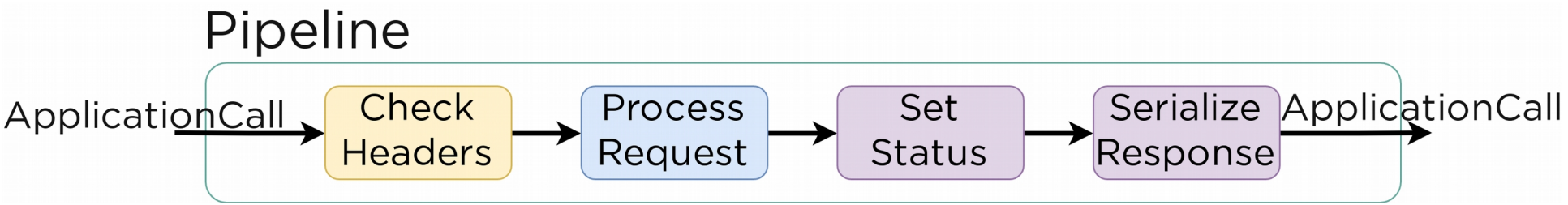
Application::class

```
fun main() {
    embeddedServer(Netty, 8080) { this: Application
        this.routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

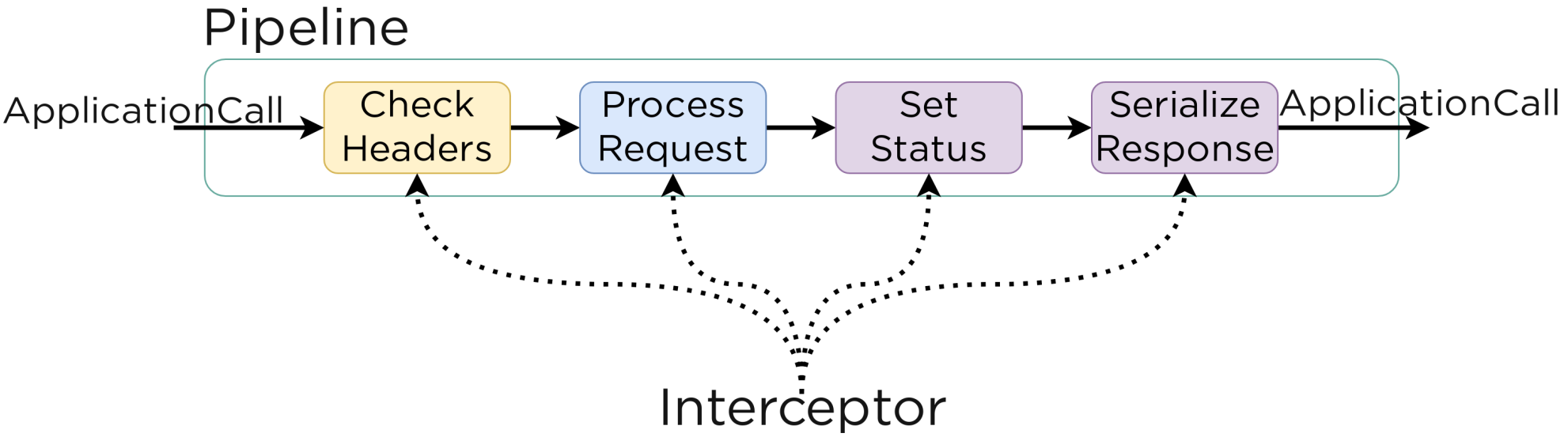
Application::class

Essentially is pipeline

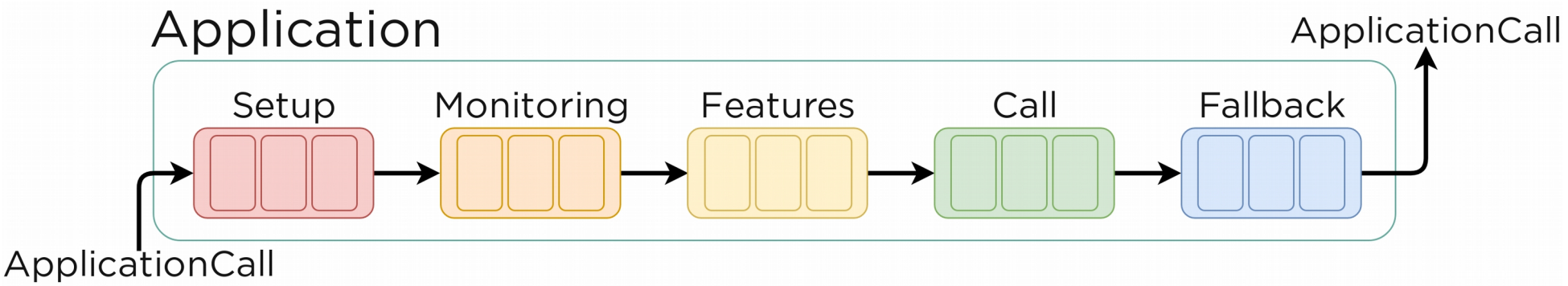
Application::class



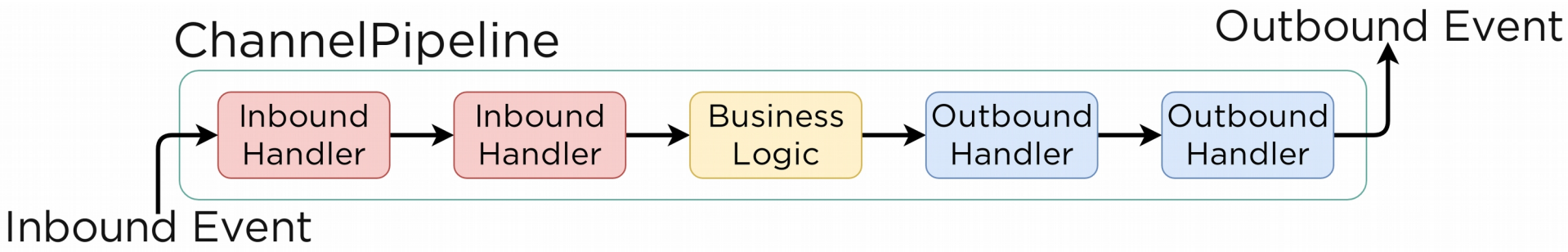
Application::class



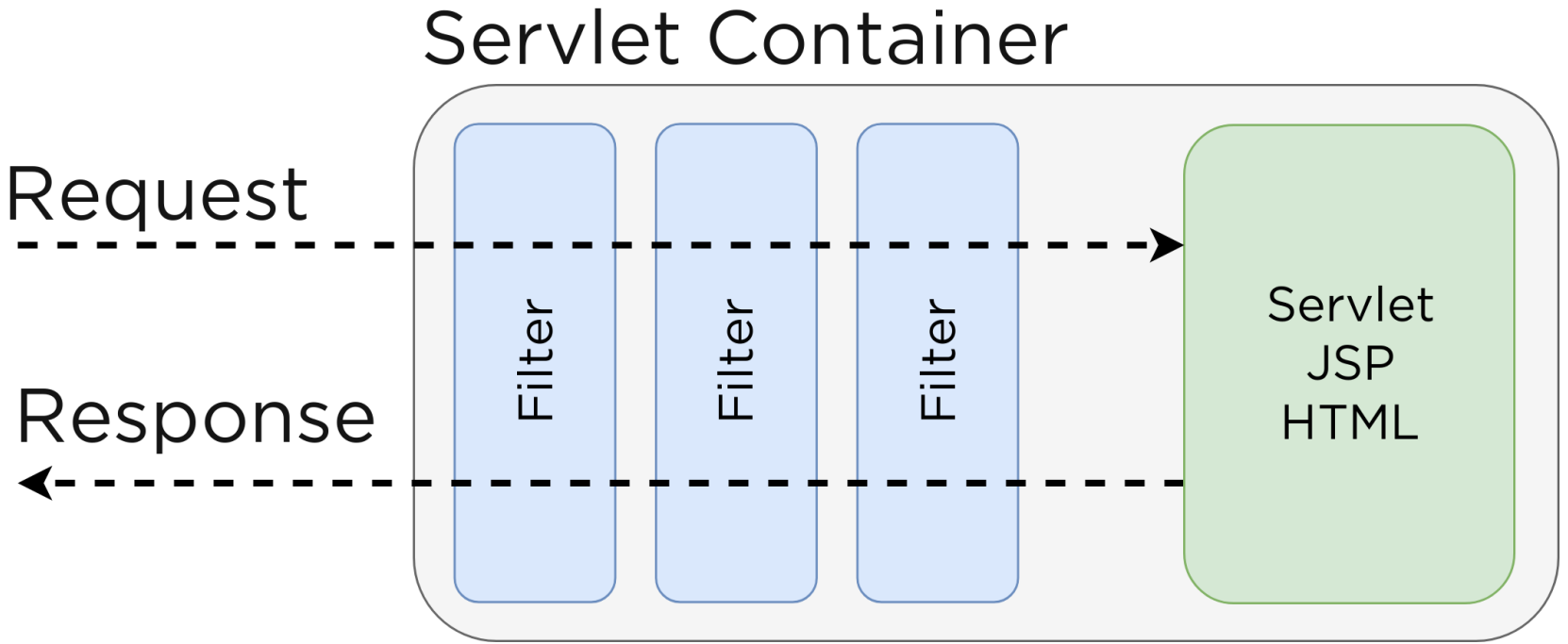
Pipeline Phase



Netty


ChannelPipeline
Outbound Event
Inbound Handler
Inbound Handler
Business Logic
Outbound Handler
Outbound Handler
Inbound Event

Servlets


Servlet Container
Request
Response
Filter
Filter
Filter
Servlet
JSP
HTML

Pipelines

ApplicationCallPipeline
- Application**Receive**Pipeline
- Application**Send**Pipeline

Pipelines

ApplicationCallPipeline
- Application**Receive**Pipeline
  - ApplicationCall
  - ApplicationReceiveRequest
- Application**Send**Pipeline
  - ApplicationCall
  - OutgoingContent

Interceptors

```
fun main() {
    embeddedServer(Netty, 8080) { this: Application
        this.routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

Application.intercept

```
embeddedServer(Netty, 8080) {
    intercept(ApplicationCallPipeline.Monitoring) {
        // log request headers
        call.request.headers
            .forEach { name, values →
                println("$name: ${values.joinToString()}")
            }
    }
}.start(wait = true)
```

Application.intercept

```
embeddedServer(Netty, 8080) {
    intercept(ApplicationCallPipeline.Monitoring) {
        // log request headers
        call.request.headers
            .forEach { name, values →
                println("$name: ${values.joinToString()}")
            }
    }
}.start(wait = true)
```

Application.intercept

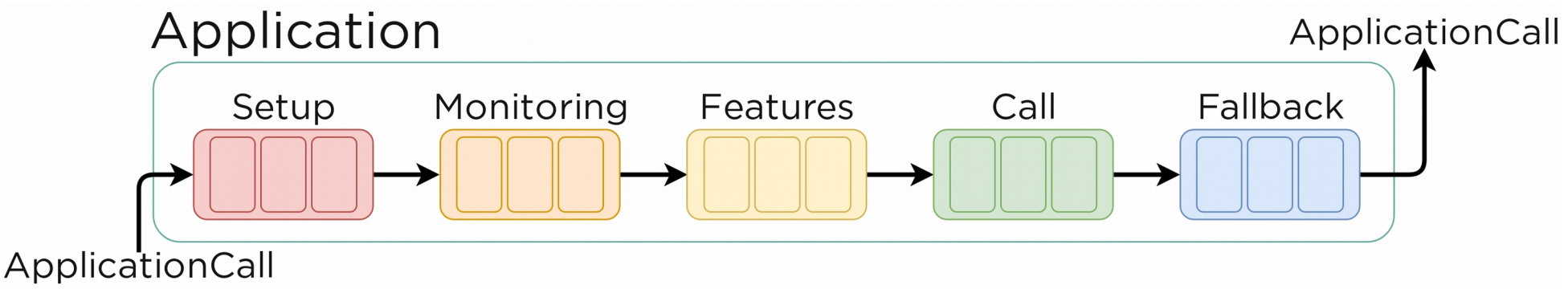
```
embeddedServer(Netty, 8080) {
    intercept(ApplicationCallPipeline.Monitoring) {
        // log request headers
        call.request.headers
            .forEach { name, values →
                println("$name: ${values.joinToString()}")
            }
    }
}.start(wait = true)
```

Application.intercept

```
embeddedServer(Netty, 8080) {
    intercept(ApplicationCallPipeline.Monitoring) {
        // log request headers
        call.request.headers
            .forEach { name, values →
                println("$name: ${values.joinToString()}")
            }
    }
}.start(wait = true)
```

Application Call

Application
Request
Response
Attributes



Feature

Routing

ContentNegotiation

Auth

Call Logging

CORS

Metrics

Compression

etc. see **ApplicationFeature**

Feature

```
routing {
    get("/") {
        call.respondText("I am Groot!", ContentType.Text.Html)
    }
}


install(Routing) {
    get("/") {
        call.respondText("I am Groot!", ContentType.Text.Html)
    }
}
```

Feature

```
embeddedServer(Netty, 8080) {
    install(DefaultHeaders)
    install(CallLogging)

    // ...
}.start(wait = true)
```

Interceptor → Feature

```
intercept(ApplicationCallPipeline.Monitoring) {
    // log request headers
    call.request.headers
        .forEach { name, values →
            println("$name: ${values.joinToString()}")
        }
}
```

Interceptor → Feature

```
typealias Configuration = Unit
class HeaderLoggingFeature {
    companion object Feature : ApplicationFeature<ApplicationCallPipeline,
Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
        override fun install(
            pipeline: ApplicationCallPipeline,
            configure: Configuration.() → Unit
        ): HeaderLoggingFeature {
            pipeline.intercept(ApplicationCallPipeline.Monitoring) {
                call.request.headers
                    .forEach { name, values →
                        println("$name: ${values.joinToString()}")
                    }
            }
            return HeaderLoggingFeature()
        }
    }
}
```

Interceptor → Feature

```
typealias Configuration = Unit
class HeaderLoggingFeature {
    companion object Feature : ApplicationFeature<ApplicationCallPipeline,
Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
        override fun install(
            pipeline: ApplicationCallPipeline,
            configure: Configuration.() → Unit
        ): HeaderLoggingFeature {
            pipeline.intercept(ApplicationCallPipeline.Monitoring) {
                call.request.headers
                    .forEach { name, values →
                        println("$name: ${values.joinToString()}")
                    }
            }
            return HeaderLoggingFeature()
        }
    }
}
```

Interceptor → Feature

```
typealias Configuration = Unit
class HeaderLoggingFeature {
    companion object Feature : ApplicationFeature<ApplicationCallPipeline,
Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
        override fun install(
            pipeline: ApplicationCallPipeline,
            configure: Configuration.() → Unit
        ): HeaderLoggingFeature {
            pipeline.intercept(ApplicationCallPipeline.Monitoring) {
                call.request.headers
                    .forEach { name, values →
                        println("$name: ${values.joinToString()}")
                    }
            }
            return HeaderLoggingFeature()
        }
    }
}
```

Interceptor → Feature

```
typealias Configuration = Unit
class HeaderLoggingFeature {
    companion object Feature : ApplicationFeature<ApplicationCallPipeline,
Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
        override fun install(
            pipeline: ApplicationCallPipeline,
            configure: Configuration.() → Unit
        ): HeaderLoggingFeature {
            pipeline.intercept(ApplicationCallPipeline.Monitoring) {
                call.request.headers
                    .forEach { name, values →
                        println("$name: ${values.joinToString()}")
                    }
            }
            return HeaderLoggingFeature()
        }
    }
}
```

Interceptor → Feature

```
typealias Configuration = Unit
class HeaderLoggingFeature {
    companion object Feature : ApplicationFeature<ApplicationCallPipeline,
Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
        override fun install(
            pipeline: ApplicationCallPipeline,
            configure: Configuration.() → Unit
        ): HeaderLoggingFeature {
            pipeline.intercept(ApplicationCallPipeline.Monitoring) {
                call.request.headers
                    .forEach { name, values →
                        println("$name: ${values.joinToString()}")
                    }
            }
            return HeaderLoggingFeature()
        }
    }
}
```

Interceptor → Feature

```
typealias Configuration = Unit
class HeaderLoggingFeature {
    companion object Feature : ApplicationFeature<ApplicationCallPipeline,
Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
        override fun install(
            pipeline: ApplicationCallPipeline,
            configure: Configuration.() → Unit
        ): HeaderLoggingFeature {
            pipeline.intercept(ApplicationCallPipeline.Monitoring) {
                call.request.headers
                    .forEach { name, values →
                        println("$name: ${values.joinToString()}")
                    }
            }
            return HeaderLoggingFeature()
        }
    }
}
```

Interceptor → Feature

```
intercept(ApplicationCallPipeline.Monitoring) {
    // log request headers
    call.request.headers
        .forEach { name, values →
            println("$name: ${values.joinToString()}")
        }
}

install(HeaderLoggingFeature)

install(HeaderLoggingFeature) {
    exclusions = listOf("User-Agent")
}
```

Content Negotiation Feature

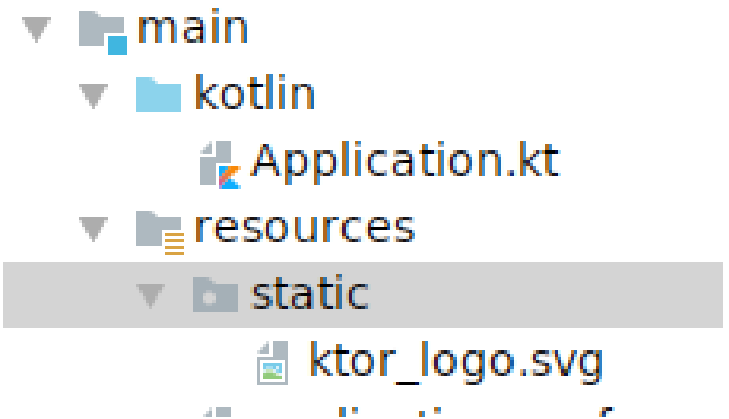
```
install(ContentNegotiation) {
    jackson {
        enable(SerializationFeature.INDENT_OUTPUT)
    }
}

routing {
    get("/resp") {
        call.respond(mapOf("hello" to "world"))
    }
}

// response
{
  "hello" : "world"
}
```

Static Files Serving Feature

```
static("/static") {
    resources("static")
}
```



Modularity

```
fun main() {
    embeddedServer(Netty, 8080) {
        routing {
            get("/") {
                call.respondText("I am Groot!", ContentType.Text.Html)
            }
        }
    }.start(wait = true)
}
```

Modularity

```
@ContextDsl
fun Application.helloWorld() {
    routing {
        get("/") {
            call.respondText("I am Groot!", ContentType.Text.Html)
        }
    }
}

fun main() {
    embeddedServer(Netty, 8080) {
        helloWorld()
    }.start(wait = true)
}
```

Modularity

```
fun main() {
    embeddedServer(Netty, 8080) {
        configuration()
        users()
        applications()
    }.start(wait = true)
}
```

Testing

```
@Test
fun testRoot() {
    withTestApplication({
        configuration()
        helloWorld()
    }) {
        handleRequest(HttpMethod.Get, "/").apply {
            assertEquals(HttpStatusCode.OK, response.status())
            assertEquals("HELLO WORLD!", response.content)
        }
    }
}
```

Testing

```
@Test
fun testRoot() {
    withTestApplication({
        configuration()
        helloWorld()
    }) {
        handleRequest(HttpMethod.Get, "/").apply {
            assertEquals(HttpStatusCode.OK, response.status())
            assertEquals("HELLO WORLD!", response.content)
        }
    }
}
```

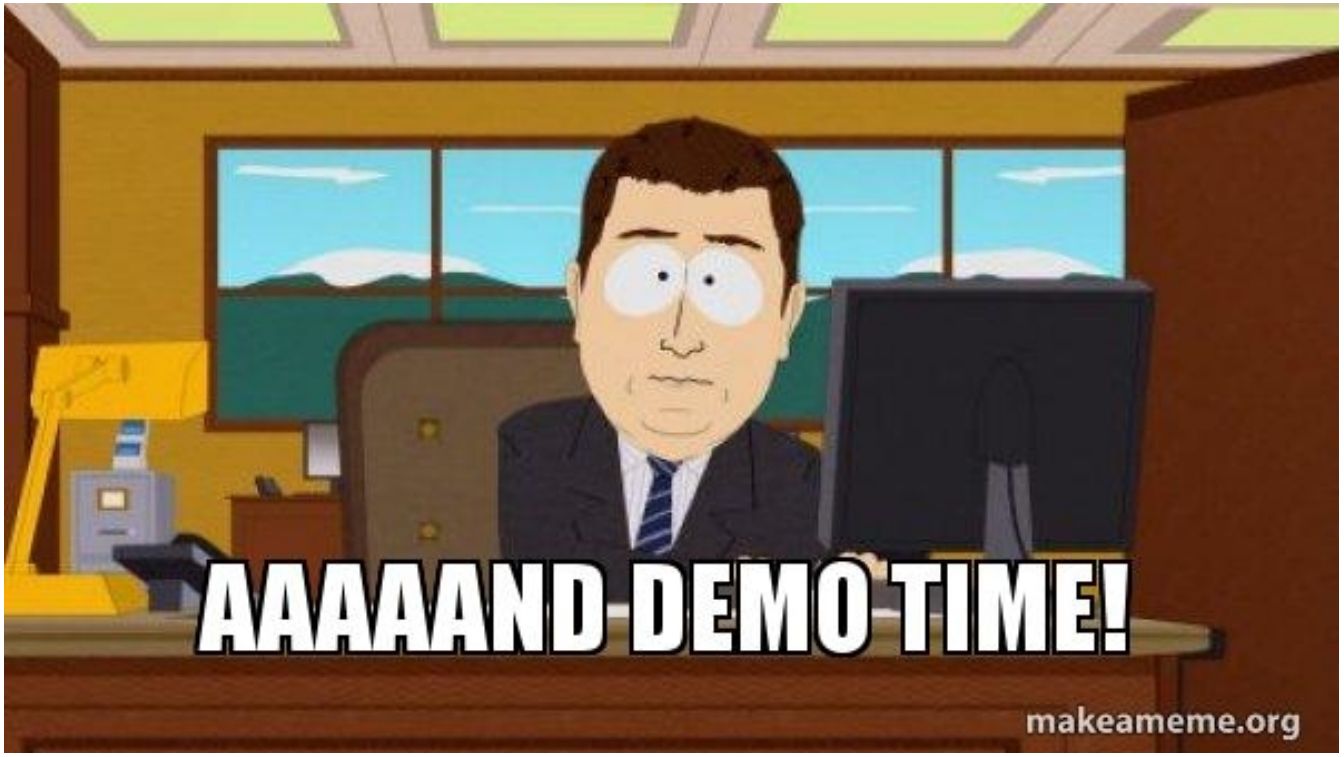
Testing

```
@Test
fun testRoot() {
    withTestApplication({
        configuration()
        helloWorld()
    }) {
        handleRequest(HttpMethod.Get, "/").apply {
            assertEquals(HttpStatusCode.OK, response.status())
            assertEquals("HELLO WORLD!", response.content)
        }
    }
}
```

Autoreloading

```
// application.conf
ktor {
    deployment {
        port = 8080
        port = ${?PORT}
        watch = ["ktor-playground"]
    }
    application {
        modules = [ com.knightkyiv.ApplicationKt.module ]
    }
}
```

Autoreloading



Autoreloading

Works on JDK8!

Ktor Client

```
val client = HttpClient(Apache)
```

Ktor Client

```
fun Application.myApp() {
    val client = HttpClient(Apache)
    routing {
        get("/call") {
            val text = client.get<String>(
                host = "localhost",
                port = 8081,
                path = "/text"
            )
            call.respondText(text)
        }
    }
}
```

Ktor Client

```
val client = HttpClient(Apache) {
    install(JsonFeature)
}

val user = client.get<User>(
    host = "localhost",
    port = 8081,
    path = "/json"
)
```

Configuration

```
interface ApplicationConfig {
    fun property(path: String): ApplicationConfigValue
    fun propertyOrNull(path: String): ApplicationConfigValue?
    fun config(path: String): ApplicationConfig
    fun configList(path: String): List<ApplicationConfig>
}
```

Configuration

```
deploymentConfig.propertyOrNull("shareWorkGroup")
    ?.getString()
    ?.toBoolean()
    ?.let {
        shareWorkGroup = it
    }
```

Configuration

```
// application.conf
jdbc {
    user = "tgto"
    password = "tgto"
    url = "jdbc:postgresql://tgto_database:5432/tgto"
    driver = "org.postgresql.Driver"
}
```

```
data class JdbcConfig(
    val user: String,
    val password: String,
    val url: String,
    val driver: String
)
```

Configuration

```
// build.gradle.kts
implementation("io.github.config4k:config4k:0.4.1")


// Application.module
val jdbcConfig = ConfigFactory.load().extract<JdbcConfig>("jdbc")
```

Server Engines

Netty
Jetty
Tomcatty
Servletty 3.0+
CIO
TestEngine

Client Engines

Apache
CIO
Jetty
OkHttp
Android
iOS
Js (JavaScript)
Curl
MockEngine

Ktor Adoption

Kotlin Census 2018

Total – 4400
Spring Boot – 1598
Ktor – 1065

Takeaway

Ktor – **connected** systems, **MPP**

Application – **Pipeline**

Pipeline – Interceptors/**Features**

Modularity

Testability


Kotlin
Belarus
User Group
USE THE
KOTLIN