LEAN & MEAN - GO MICROSERVICES WITH DOCKER SWARM MODE AND SPRING CLOUD

ERIK LUPANDER
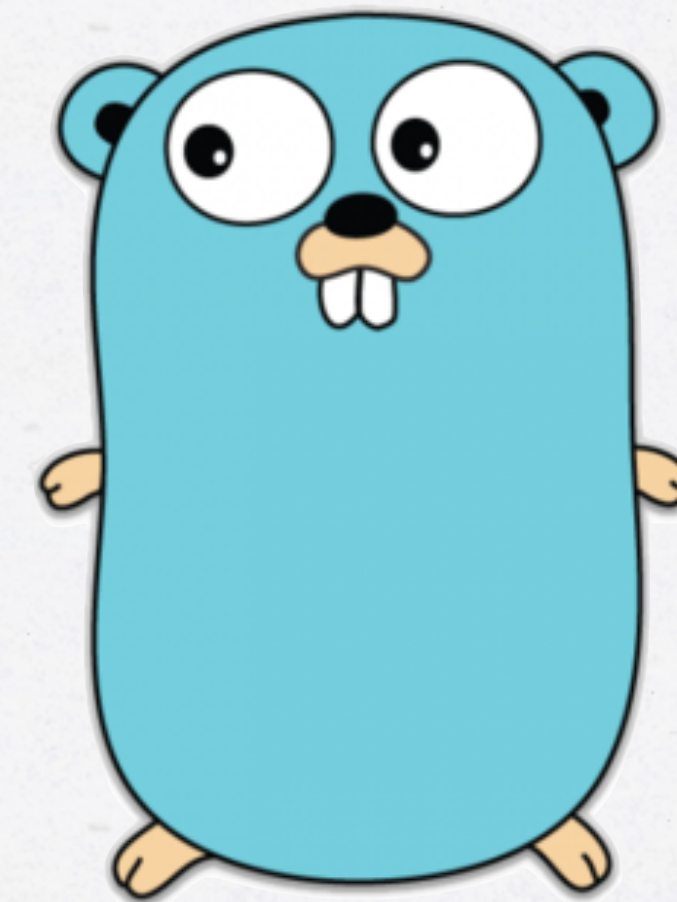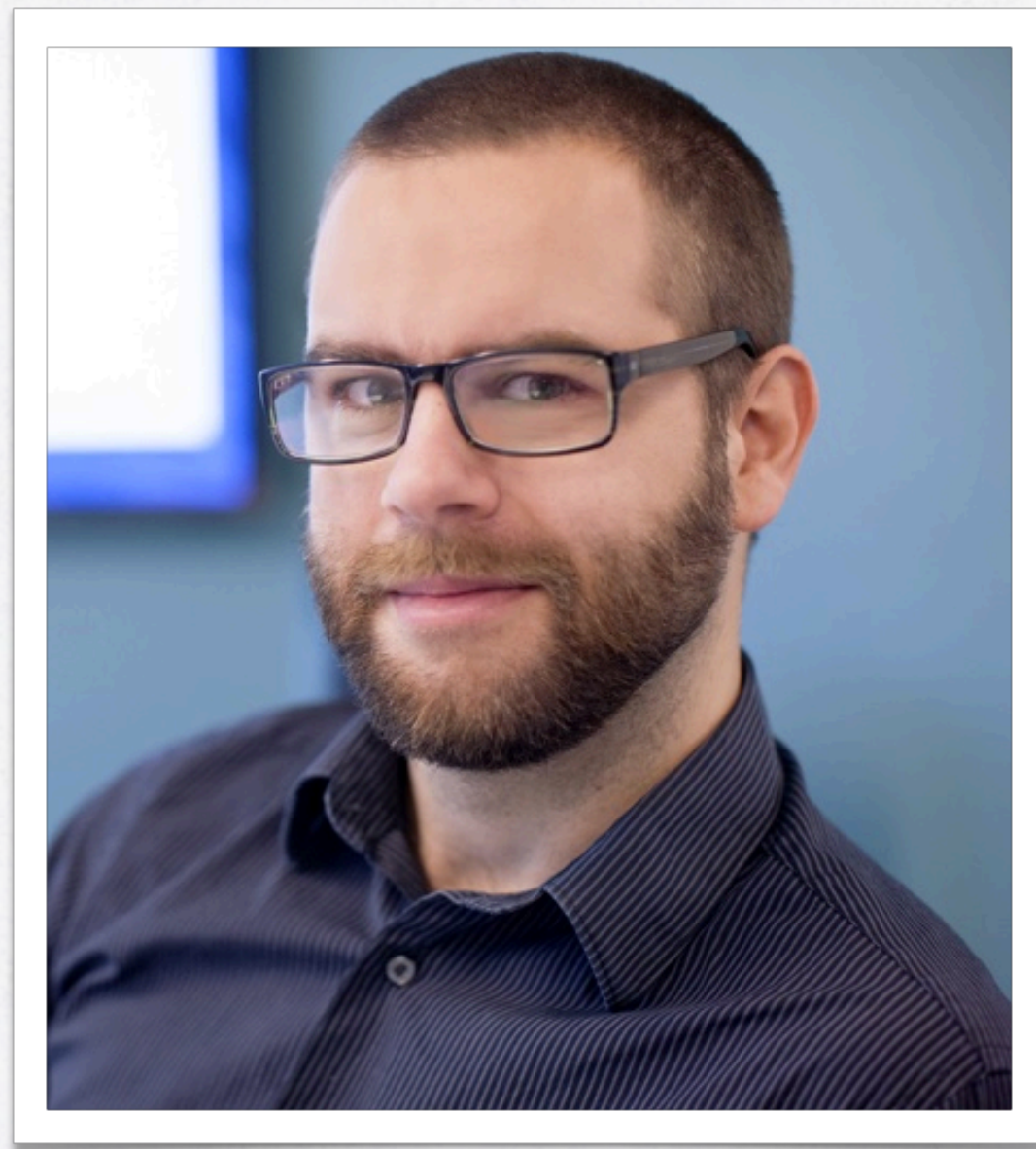
2017-11-09 | CALLISTAENTERPRISE.SE

ABOUT ME

- Erik Lupander, consultant at Callista Enterprise.
- Primarily a Java dude.
- "Discovered" Go about 2 years ago.

Love at first sight!

Go

ON THE AGENDA…

- Background: The footprint problem.
- The Go programming language.
- Go in the context of:
  - Microservices
  - Spring Cloud/Netflix OSS
  - Docker Swarm mode.
- Demos!

CALLISTA
— ENTERPRISE —

Can Go help us help us reduce the footprint of a microservice?

CALLISTA
— ENTERPRISE —

THE FOOTPRINT PROBLEM

- JVM-based solutions comes with a hefty footprint.
- If you need to run tens or even hundreds of microservice instances, cost is definitely a factor.
  - t2.micro (1GB) —> t2.small (2GB) doubles the cost / h.
- There are obviously many other alternatives for microservice development….
  - Very interesting topic… if we had all day.
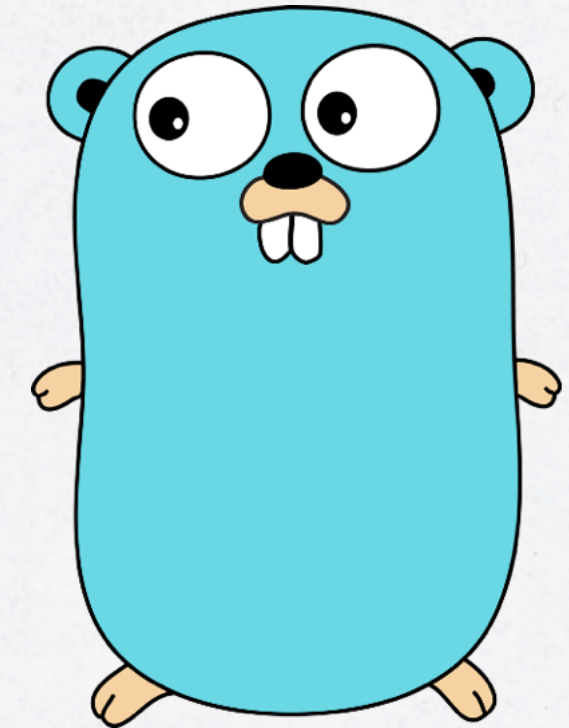
CALLISTA
— ENTERPRISE —

The Go Language

It has been stated that the reason the three authors created Go was their…

CALLISTA
— ENTERPRISE —

"… shared dislike of C++'s complexity as a primary motivation for designing a new language"

CALLISTA
— ENTERPRISE —

Go was designed …

CALLISTA
— ENTERPRISE —

THE GO LANGUAGE

"… to eliminate the slowness and clumsiness of software development at Google"

Go official FAQ

CALLISTA
— ENTERPRISE —

WHAT WAS IMPROVED WITH GO?

- ~50x build time improvement over C++
  - Internal C++ application builds taking 30-75 minutes.
- Better dependency management
- Cross-platform builds
- Language level concurrency
- Readable and maintainable code
  - Even for non superstar developers

CALLISTA
— ENTERPRISE —

THE GO LANGUAGE

- Claims to be
  - efficient, scalable and productive.
- Designed
  - to improve the working environment for its designers and their coworkers.
- Is not
  - a research language.

- Go is
  - compiled, statically typed, concurrent, garbage-collected
- Has
  - structs, pointers, interfaces, closures
- But does not have
  - classes, inheritance, generics, operator overloading, pointer arithmetic

CALLISTA
— ENTERPRISE —

What does actual developers think about Go?

CALLISTA
— ENTERPRISE —

"… a disservice to intelligent programmers"

Gary Willoughby - blogger

CALLISTA
— ENTERPRISE —

"… stuck in the 70's"

Dan Given

CALLISTA
— ENTERPRISE —

*"… psuedointellectual arrogance of Rob Pike
and everything he stands for"*

Keith Wesolowski

CALLISTA
— ENTERPRISE —

But also

CALLISTA

"I like a lot of the design decisions they made in the [Go] language. Basically, I like all of them."

Martin Odersky, creator of Scala

CALLISTA
— ENTERPRISE —

"Never used a language before that empowers you to solve problems as quick as Go does"

Alexander Orlov @ Twitter

CALLISTA
— ENTERPRISE —

"Go isn't a very good language in theory, but it's a great language in practice, and practice is all I care about"

anonymous hackernews poster

CALLISTA
— ENTERPRISE —

Some pros and cons

CALLISTA
— ENTERPRISE —

DEVELOPMENT IN GOLANG - PROS

- Easy to learn, readable, productive and pretty powerful.
- The built-in concurrency is awesome.
- Cross-platform.
- Rich standard APIs and vibrant open source community.
- Quick turnaround and decent IDE support (getting better!)
- Nice bundled tools.
  - Built-in unit testing, profiling, coverage, benchmarking, formatting, code quality…
- Strongly opinionated.
  - Code formatting, compile errors on typical warnings.

DEVELOPING IN GOLANG - SOME CONS

- Missing generics
- Dependency versioning
- Verbose syntax
  - Error checking, no autoboxing of primitive types etc.
- Unit testing and Mocking isn't very intuitive.

CALLISTA
— ENTERPRISE —

- Some well-known software built entirely in golang
  - Docker
  - Kubernetes
  - etcd
  - influxdb (time series database)
  - cockroachdb (spanner-like database)

CALLISTA
— ENTERPRISE —

Two code samples

CALLISTA
— ENTERPRISE —

SAMPLE CODE 1 - HELLO WORLD

```go
package main

import "fmt"

func main() {
        fmt.Println("Hello world!")
}
```

```go
func main() {
        responseChannel := make(chan []byte)

        urls := []string{"http://gp.se", "http://dn.se"}
        for _, url := range urls {
                go doReq(url, responseChannel)   // Make async HTTP call
        }


        for i := 0; i < len(urls); i++ {
                data := <- responseChannel        // Blocks here
                fmt.Println(string(data))
        }
}

func doReq(url string, responseChannel chan []byte) {
        resp, _ := http.Get(url)
        body, _ := ioutil.ReadAll(resp.Body)
        responseChannel <- body                   // Pass result to channel
}
```

CALLISTA
— ENTERPRISE —

Go microservices

CALLISTA
— ENTERPRISE —

GO MICROSERVICE IMPLEMENTATION - CONSIDERATIONS

- When implementing microservices, we need working, mature and stable libraries for things such as:
  - HTTP / REST / RPC APIs
  - Data serializers / deserializers (json, xml etc.)
  - Messaging APIs
  - Persistence APIs
  - Logging
  - Testability

The demo application

CALLISTA
— ENTERPRISE —

ARCHITECTURAL OVERVIEW

Curl

Docker Swarm cluster

HTTPS

Edge server **(Netflix Zuul)**

OAuth token relay

CB/TA

HTTP

Security API (Go)

OAuth Res

CB / TA

OAuth
Authorization
Server
(spring-security)

Monitor
Dashboard
(Hystrix Dashboard)

HTTP

Account Composite (Go)

CB / TA

AMQP

VipService (Go)

Configuration
Server
(spring-cloud-config)

Hystrix Stream
aggregation
(Modified Netflix
Turbine)

HTTP

HTTP

AMQP
Messaging
(RabbitMQ)

TA

Images (Go)

Quotes-Service
(Spring Boot)

Trace
Analysis
(Zipkin)

WHY GO - RUNTIME CHARACTERISTICS

- Low memory usage
- Typically executes at least as fast as Java
- Fast startup
- Highly concurrent
- Garbage Collector geared for very short GC pauses

CALLISTA
— ENTERPRISE —

GO MICROSERVICES - STATICALLY LINKED BINARIES

- Statically linked binary produces an executable without external dependencies.
  - No jar- or dll-hell
  - No requirement on the OS having a JRE / CLR / NodeJS or other libraries
    - (except libc)
- Small executable size

CALLISTA
— ENTERPRISE —

DOCKER CONTAINERS & STATICALLY LINKED BINARIES

- In the context of Docker Containers, the statically linked binary allows use of very bare parent images.
- I'm using *iron/base* which is ~6 mb, *alpine* is another popular choice.

```
FROM iron/base

EXPOSE 6868
ADD vipservice-linux-amd64 /
ADD healthcheck-linux-amd64 /

HEALTHCHECK CMD ["./healthcheck-linux-amd64", "-port=6868"]

ENTRYPOINT ["./vipservice-linux-amd64", "-profile=test"]
```

Demo 1
Footprint @ Docker Swarm Mode

CALLISTA
— ENTERPRISE —

"what is hard in Microservices is all the things around them"

Jonas Bonér - author of Akka

CALLISTA
— ENTERPRISE —

Consider:

CALLISTA
— ENTERPRISE —

MICROSERVICE CONSIDERATIONS

- Centralized configuration
- Service Discovery
- Centralized Logging
- Distributed Tracing
- Circuit Breaking
- Load balancing
- Edge server / Reverse proxy
- Monitoring
- Security

CALLISTA
— ENTERPRISE —

ARCHITECTURAL OVERVIEW

Curl

Legend
- CB = Circuit Breaker (Go Hystrix)
- TA = Correlated tracing (Opentracing API / Zipkin)

Docker Swarm cluster

HTTPS

Edge server (Netflix Zuul)

OAuth token relay

CB/TA

HTTP

Security API (Go)

OAuth Res

CB / TA

HTTP

Account Composite (Go)

CB / TA

AMQP

VipService (Go)

TA

HTTP

HTTP

Images (Go)

TA

Quotes-Service (Spring Boot)
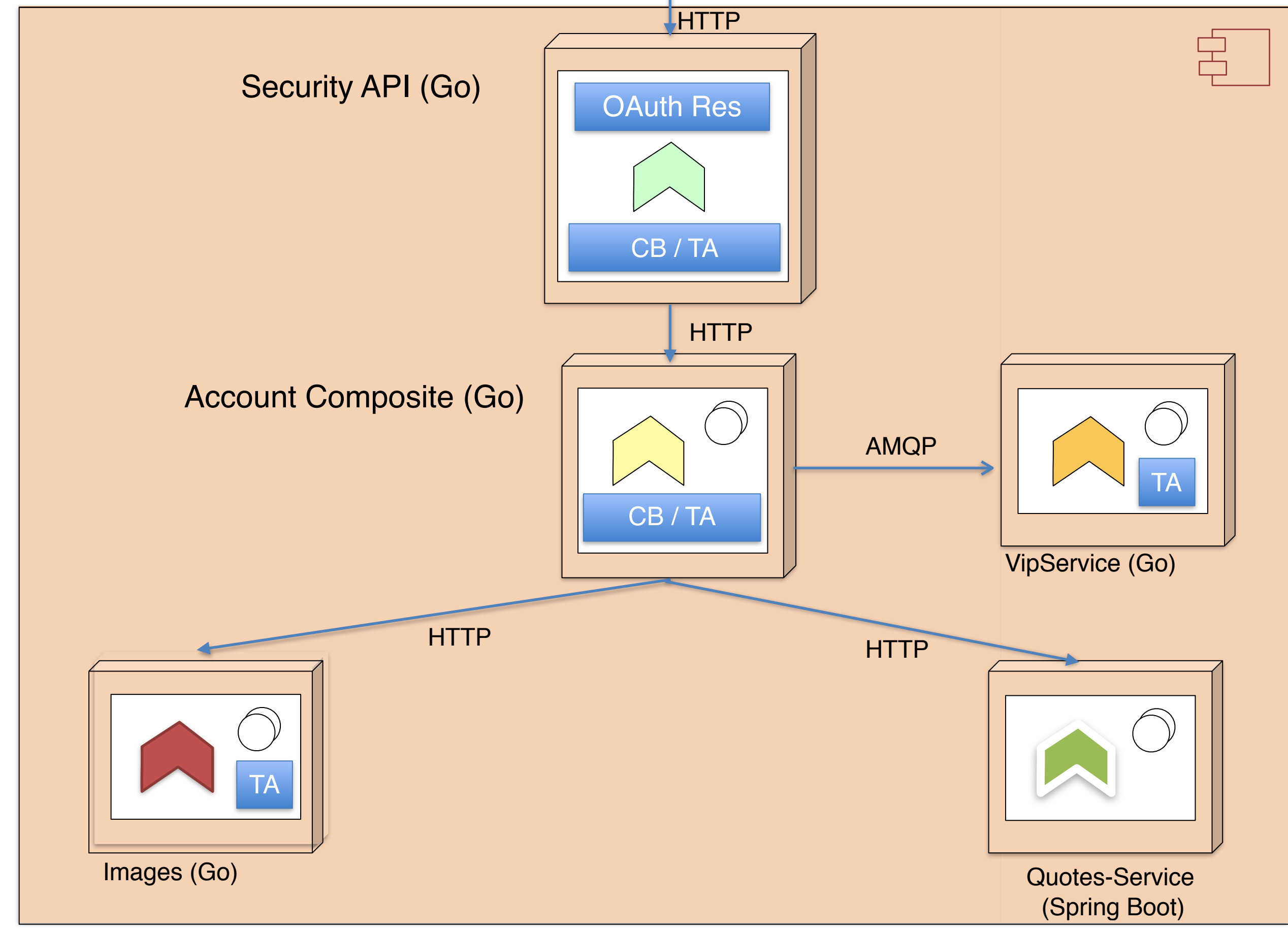
OAuth Authorization Server (spring-security)

Configuration Server (spring-cloud-config)

AMQP Messaging (RabbitMQ)

Monitor Dashboard (Hystrix Dashboard)

Hystrix Stream aggregation (Modified Netflix Turbine)

Trace Analysis (Zipkin)

Things not really Go-related…

CALLISTA
— ENTERPRISE —

EDGE SERVER

- Our Go services doesn't care about the EDGE / reverse-proxy
- Netflix Zuul, Nginx, HAProxy …
- Or use solution provided by container orchestrator
  - Ingress Routing mesh (Docker Swarm mode)
  - Ingress controller (K8S)
  - Routes (OpenShift)
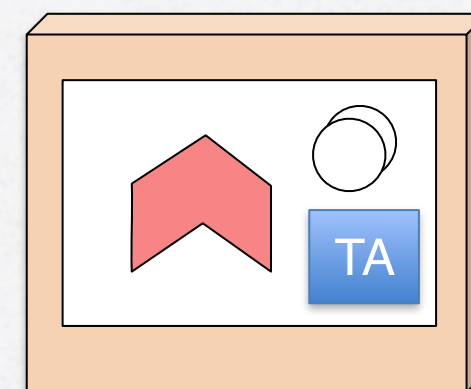- Must forward HTTP headers.
  - Security

CALLISTA
— ENTERPRISE —

- Load-balancing and Service Discovery is handled by the orchestration engine.
  - E.g. the Docker Swarm or K8S / OpenShift "service" abstraction.
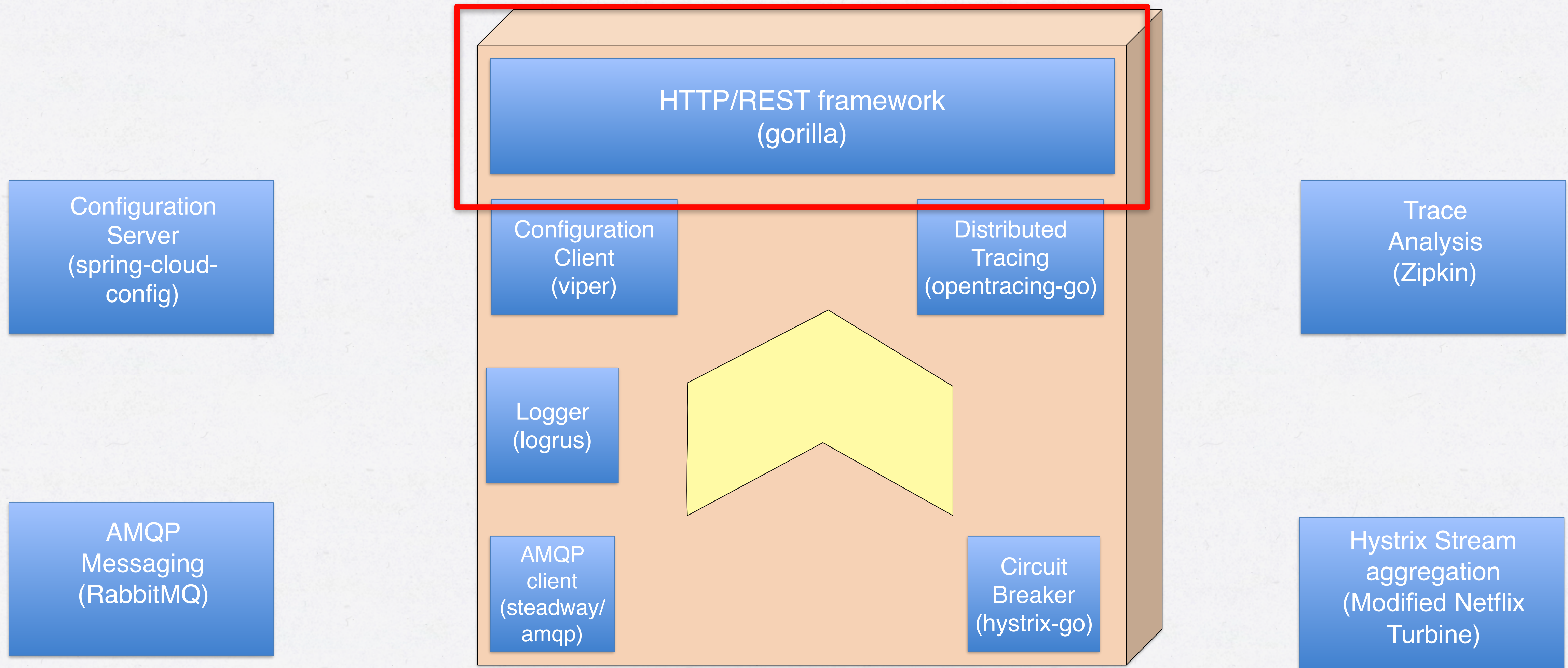- Eureka service discovery and Ribbon-like client-based load-balancing can be implemented too.

```
errors := hystrix.Go("get_account", func() error {

    req, _ := http.NewRequest("GET", "http://accountservice:7777/accounts/" + accountId, nil)
```

CALLISTA
— ENTERPRISE —

Demo 2 -
Load balancing and fast scaling
@ Docker Swarm

CALLISTA
— ENTERPRISE —

Go Microservice Anatomy

CALLISTA
— ENTERPRISE —

HTTP / REST FRAMEWORK

HTTP/REST framework
(gorilla)

Configuration
Server
(spring-cloud-
config)

Configuration
Client
(viper)

Distributed
Tracing
(opentracing-go)

Trace
Analysis
(Zipkin)

Logger
(logrus)

AMQP
Messaging
(RabbitMQ)

AMQP
client
(steadway/
amqp)

Circuit
Breaker
(hystrix-go)

Hystrix Stream
aggregation
(Modified Netflix
Turbine)

CALLISTA
— ENTERPRISE —

GO WITH OUT WITHOUT WEB FRAMEWORKS?

- Consider using the native http packages + a router package over a full-blown web framework such as gin, echo, beego.

HTTP FRAMEWORK (GORILLA)

```go
var routes = Routes{

        Route{
                "GetAccount",
                "GET",
                "/accounts/{accountId}",
                GetAccount,
        },
        Route{
                "HealthCheck",
                "GET",
                "/health",
                func(w http.ResponseWriter, r *http.Request) {
                        w.Header().Set("Content-Type", "text/plain; charset=UTF-8")
                        w.Write([]byte("OK"))
                },
        },
}
```
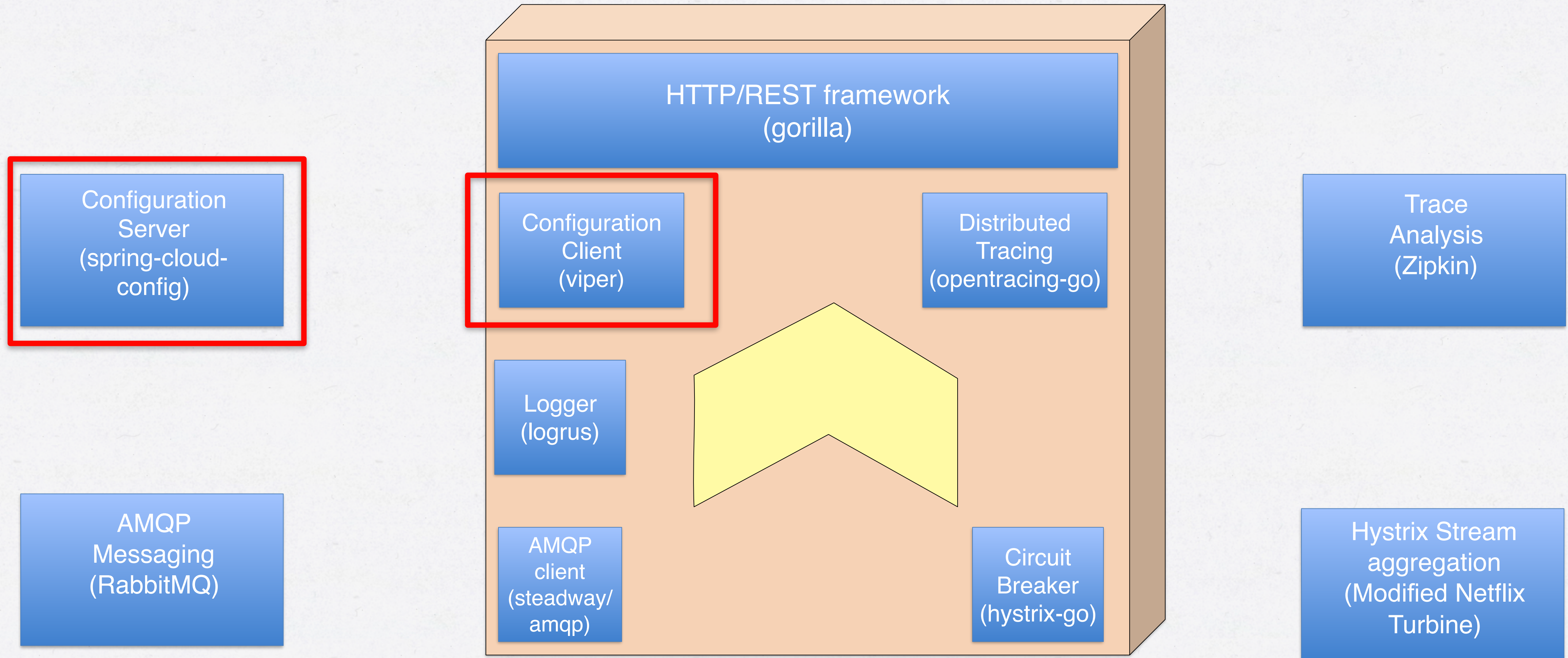
HTTP FRAMEWORK (GORILLA)

```go
func GetAccount(w http.ResponseWriter, r *http.Request) {

        var accountId = mux.Vars(r)["accountId"]
        account, _ := client.GetAccount(accountId)
        data, _ := json.Marshal(account)


        w.Header().Set("Content-Type", "application/json")
        w.WriteHeader(http.StatusOK)
        w.Write(data)

}
```
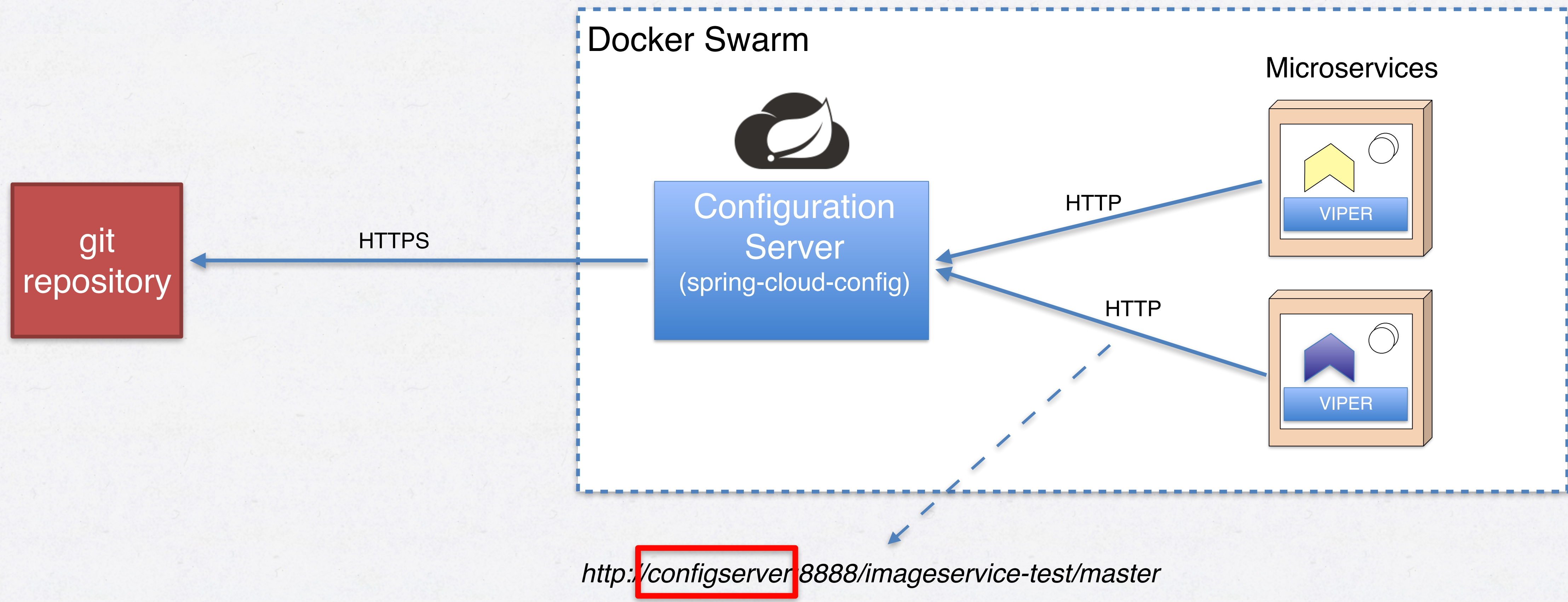
CALLISTA
— ENTERPRISE —

CENTRALIZED CONFIGURATION

Configuration Server
(spring-cloud-config)

Configuration Client
(viper)

Distributed Tracing
(opentracing-go)

Trace Analysis
(Zipkin)

HTTP/REST framework
(gorilla)

Logger
(logrus)

AMQP Messaging
(RabbitMQ)

AMQP client
(steadway/amqp)

Circuit Breaker
(hystrix-go)

Hystrix Stream aggregation
(Modified Netflix Turbine)

CALLISTA
— ENTERPRISE —

CENTRALIZED CONFIGURATION

- With possibly tens of microservices and hundreds of instances, centralized and externalized configuration is a must.
- Configuration providers:
  - Config servers
    - Spring Cloud Config, etcd …
  - Container orchestrator mechanisms
    - K8S and OpenShift has "config maps" and "secrets" in order to mount configuration files, certificates etc. into containers at startup.

CALLISTA
— ENTERPRISE —

Docker Swarm

Microservices

Configuration Server
(spring-cloud-config)

VIPER

VIPER

HTTP

HTTP

HTTPS

git repository

http://configserver:8888/imageservice-test/master

CALLISTA
— ENTERPRISE —

CONFIGURATION - VIPER

- Viper supports YAML, .properties, JSON and Env-vars
- With a few lines of code, we can load and inject config from Spring Cloud Config into Viper

```
resp, err := http.Get("http://configserver:8888/" + appName + "-" + envProfile + "/" + envProfile)
if err != nil {
        panic("Failed to load configuration: " + err.Error())
}

body, err := ioutil.ReadAll(resp.Body)

var cloudConfig model.SpringCloudConfig
json.Unmarshal(body, &cloudConfig)

for key, value := range cloudConfig.PropertySources[0].Source {
        viper.Set(key, value)
}
viper.SetConfigType("json")
```
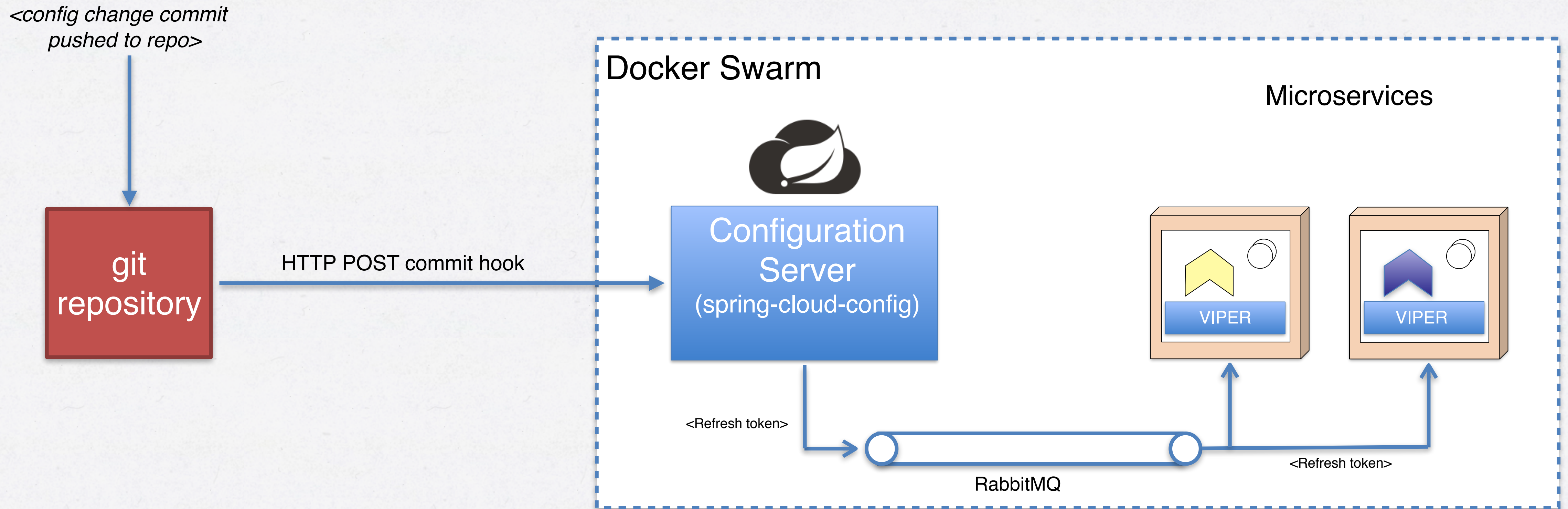
CALLISTA
— ENTERPRISE —
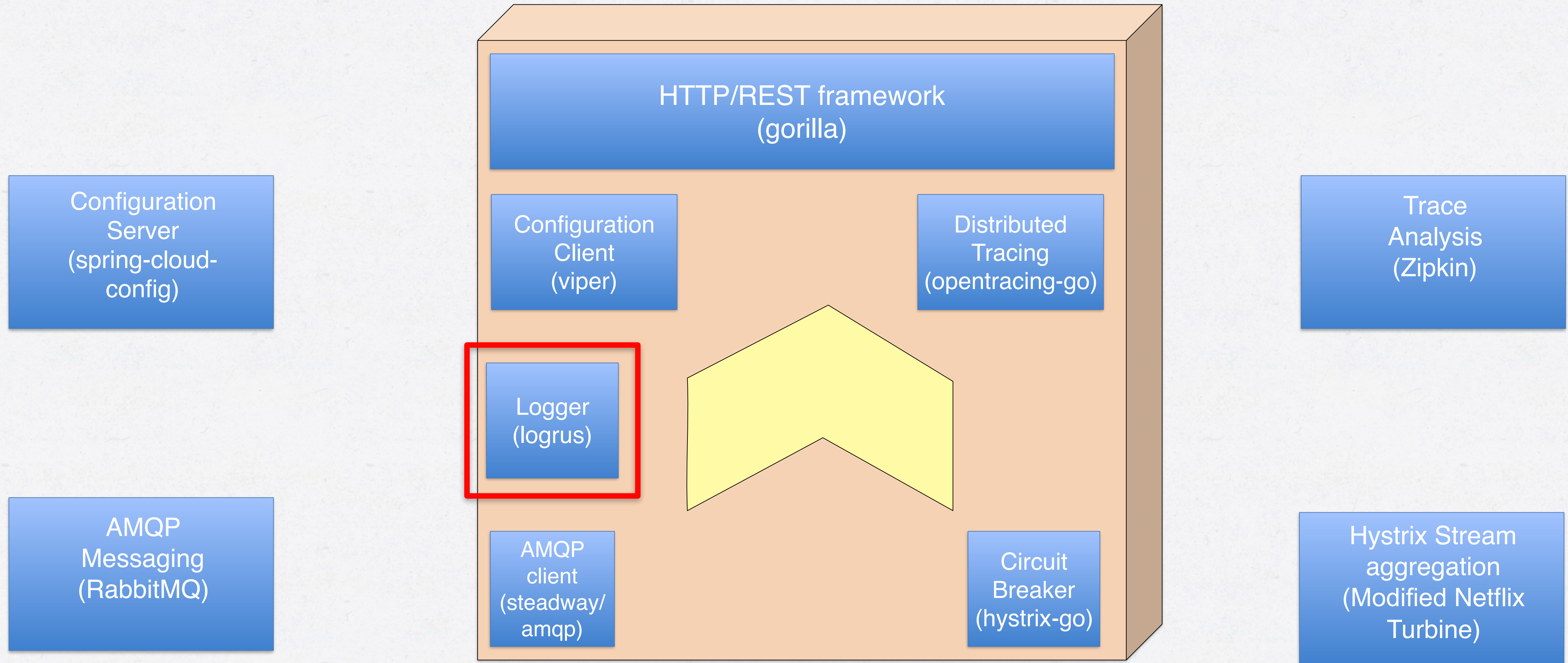
CONFIGURATION - VIPER USAGE

```go
go service.StartWebServer(viper.GetString("server_port")) // Starts HTTP service
```

<config change commit pushed to repo>

Docker Swarm

Microservices

git repository

HTTP POST commit hook

Configuration Server (spring-cloud-config)

VIPER

VIPER

<Refresh token>

RabbitMQ

<Refresh token>

Demo 3 - Configuration Push

CALLISTA
— ENTERPRISE —

CENTRALIZED LOGGING

HTTP/REST framework
(gorilla)

Configuration
Server
(spring-cloud-
config)

Configuration
Client
(viper)

Distributed
Tracing
(opentracing-go)

Trace
Analysis
(Zipkin)

Logger
(logrus)

AMQP
Messaging
(RabbitMQ)

AMQP
client
(steadway/
amqp)

Circuit
Breaker
(hystrix-go)

Hystrix Stream
aggregation
(Modified Netflix
Turbine)

CALLISTA
— ENTERPRISE —

LOGGING - LOGRUS

- Applications needs structured logging
  - slf4j, log4j, logback…
- Logrus is a similar API for Go
- Supports levels, fields, formatters, hooks

CALLISTA
— ENTERPRISE —

```go
func init() {
    profile := flag.String("profile", "test", "Environment profile")
    if *profile != "dev" {
        logrus.SetFormatter(&logrus.JSONFormatter{})
    } else {
        logrus.SetFormatter(&logrus.TextFormatter{
            TimestampFormat: "2006-01-02T15:04:05.000",
            FullTimestamp: true,
        })
    }
    logrus.Infof("Successfully initialized %v\n", appName)
}
```
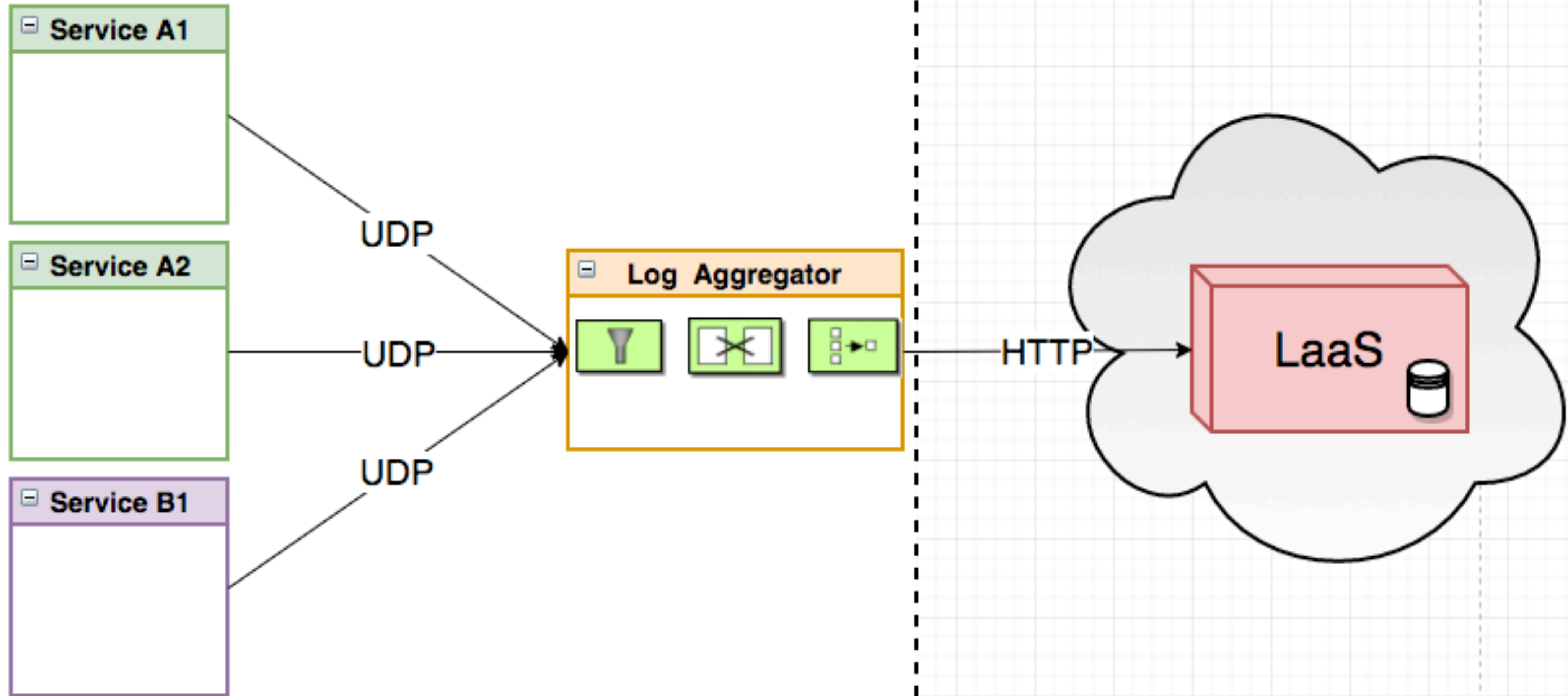
60

CALLISTA
— ENTERPRISE —

CENTRALIZING LOGS

- In a Docker context, we configure a logging driver when declaring our "service".
  - The logging driver adds lots of nice container metadata.
- Logs are sent to an aggregation service (typically something like logstash)
- The log aggregation service may perform some filtering, transforming etc. before storing logs to a storage backend or sending them to a LaaS provider.

CALLISTA
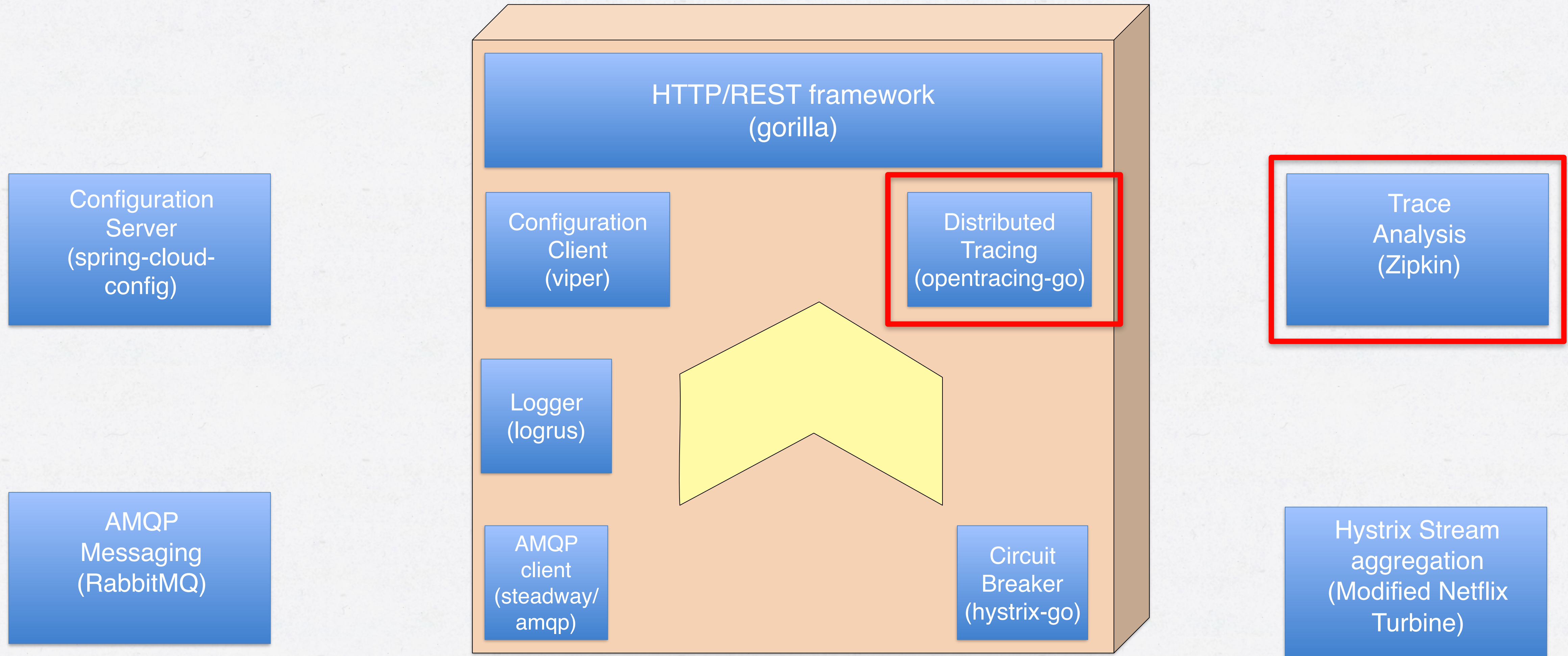— ENTERPRISE —

LOGGING WITH CONTAINER METADATA (GELF)

```
{
    "version":"1.1",
    "host":"swarm-manager-0",
    "short_message":{
        "level":"info",
        "msg":"Successfully initialized service",
        "time":"2017-07-17T16:03:35+02:00"
    },
    "timestamp":1.487625824614e+09,
    "level":6,
    "_command":"./vipservice-linux-amd64 -profile=test",
    "_container_id":"894edfe2faed131d417eebf77306a0386b430....",
    "_container_name":"vipservice.1.jgaludcy21iriskcu1fx9nx2p",
    "_created":"2017-02-20T21:23:38.877748337Z",
    "_image_id":"sha256:1df84e91e0931ec14c6fb4e55.....",
    "_image_name":"someprefix/vipservice:latest",
    "_tag":"894edfe2faed"
}
```

CALLISTA
— ENTERPRISE —

DISTRIBUTED TRACING

HTTP/REST framework
(gorilla)

Configuration
Client
(viper)

Distributed
Tracing
(opentracing-go)

Logger
(logrus)

AMQP
client
(steadway/
amqp)

Circuit
Breaker
(hystrix-go)

Configuration
Server
(spring-cloud-
config)

AMQP
Messaging
(RabbitMQ)

Trace
Analysis
(Zipkin)

Hystrix Stream
aggregation
(Modified Netflix
Turbine)

CALLISTA
— ENTERPRISE —
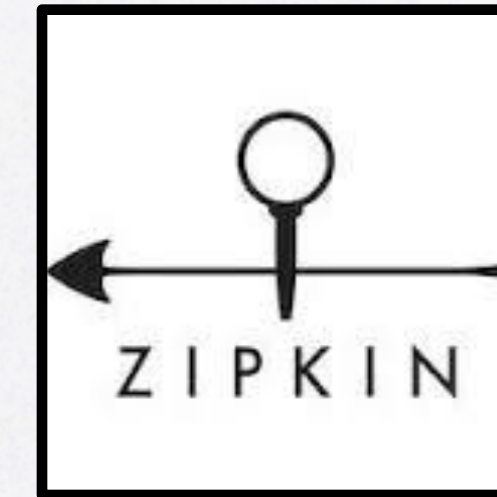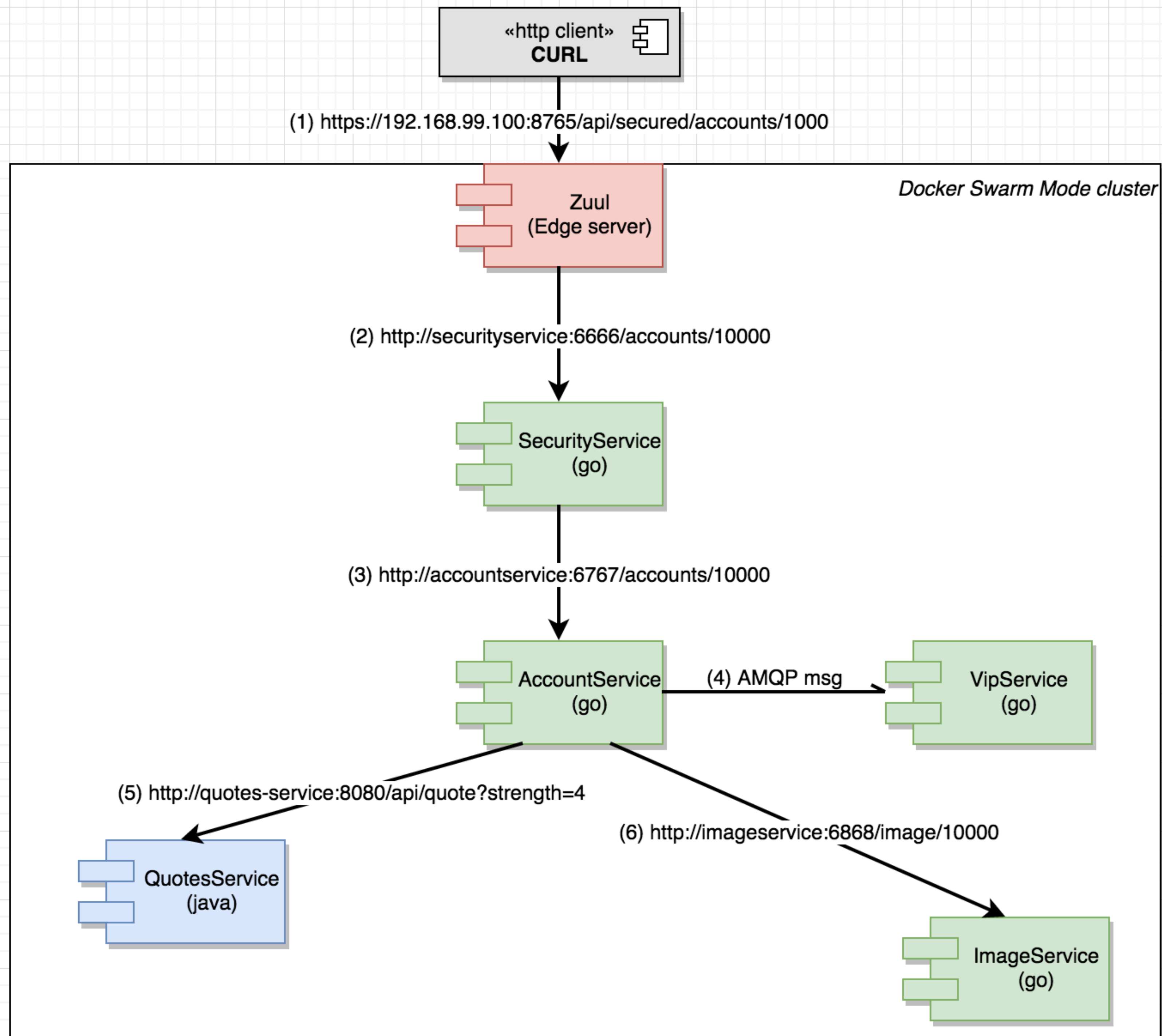
DISTRIBUTED TRACING

- Track a request over multiple microservices
  - Also trace within services and methods
- Invaluable for high-level profiling across the service stack.
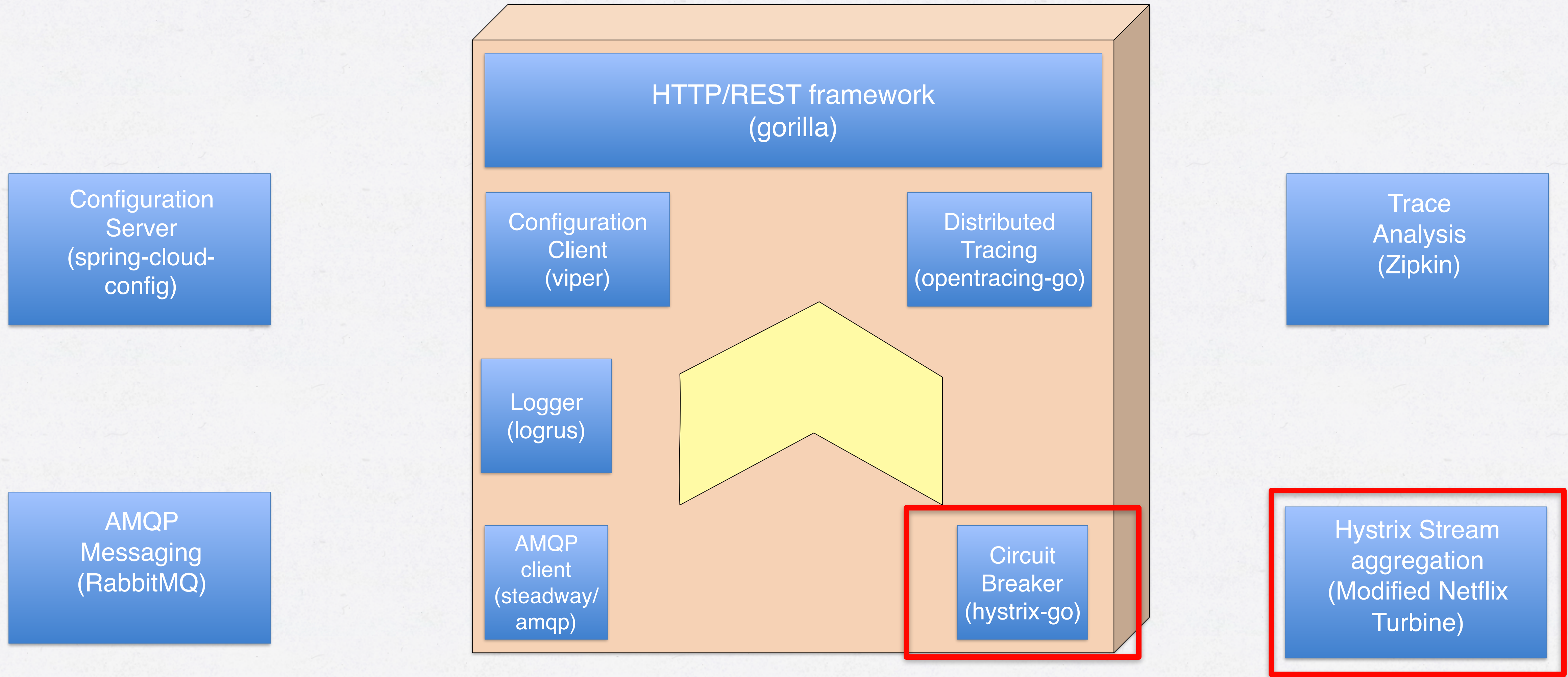- Facilitated by go-opentracing and zipkin

CALLISTA
— ENTERPRISE —

GO-OPENTRACING CODE SAMPLE

```go
// Tracing code.
span := tracing.StartChildSpanFromContext(ctx, "QueryAccount")
defer span.Finish()
```

CALLISTA
— ENTERPRISE —

Demo 4 - Distributed Tracing with Zipkin

CALLISTA
— ENTERPRISE —

CIRCUIT BREAKER

Configuration
Server
(spring-cloud-
config)

AMQP
Messaging
(RabbitMQ)

HTTP/REST framework
(gorilla)

Configuration
Client
(viper)

Distributed
Tracing
(opentracing-go)

Logger
(logrus)

AMQP
client
(steadway/
amqp)

Circuit
Breaker
(hystrix-go)

Trace
Analysis
(Zipkin)

Hystrix Stream
aggregation
(Modified Netflix
Turbine)

CALLISTA
— ENTERPRISE —

CIRCUIT BREAKING - HYSTRIX

- Mechanism to make sure a single malfunctioning microservice doesn't halt the entire service or application.
- go-hystrix (circuit breaker)
- Netflix Turbine (aggregation)
- Netflix Hystrix Dashboard (GUI)

- Programmatic hystrix configuration

```go
func configureHystrix() {

	hystrix.ConfigureCommand("get_account_image", hystrix.CommandConfig{
		Timeout:               3000,
		MaxConcurrentRequests: 100,
		ErrorPercentThreshold: 25,
	})
	hystrix.ConfigureCommand("get_account", hystrix.CommandConfig{
		Timeout:               3000,
		MaxConcurrentRequests: 100,
		ErrorPercentThreshold: 25,
	})
}
```
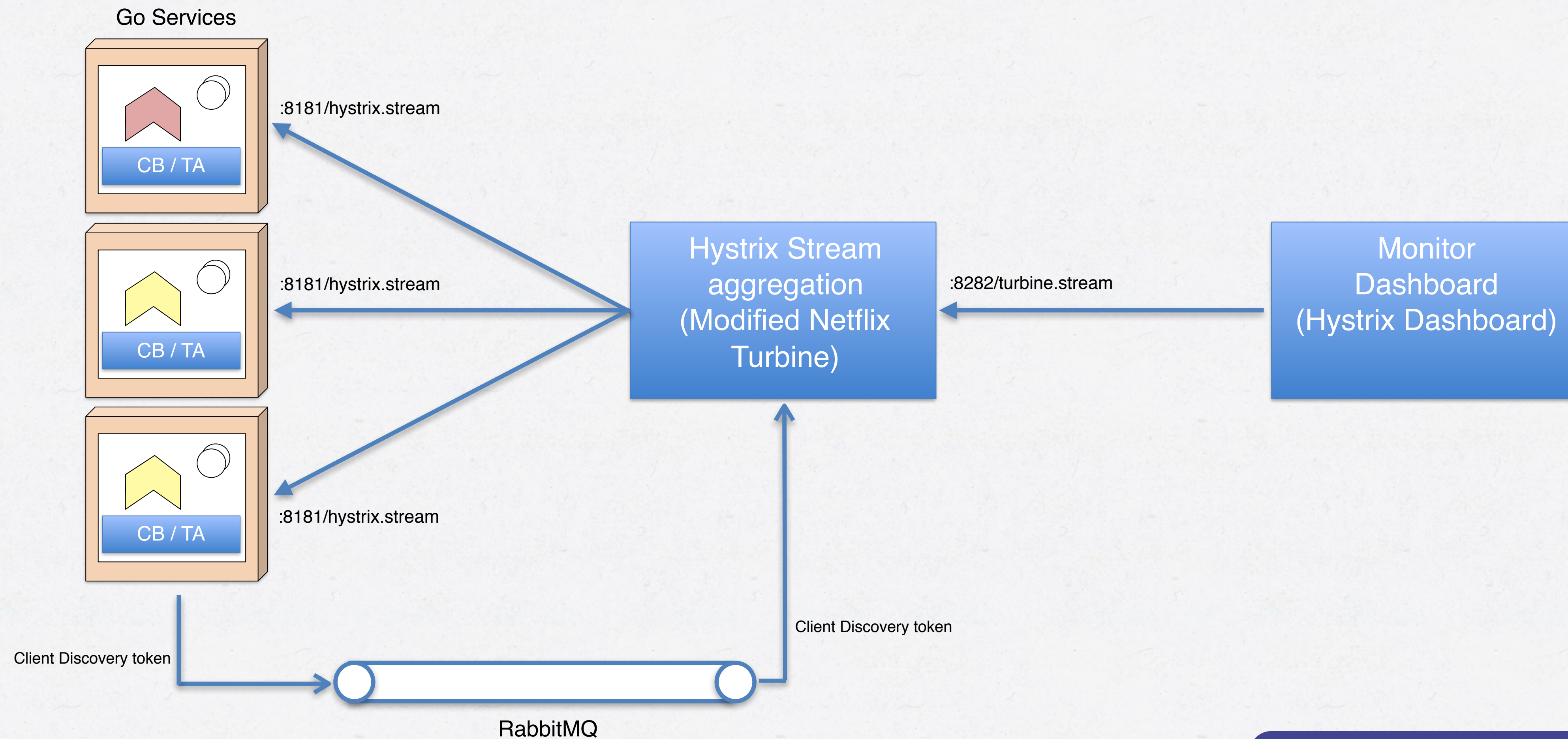
CALLISTA
— ENTERPRISE —

- Example go-hystrix usage, non-blocking.
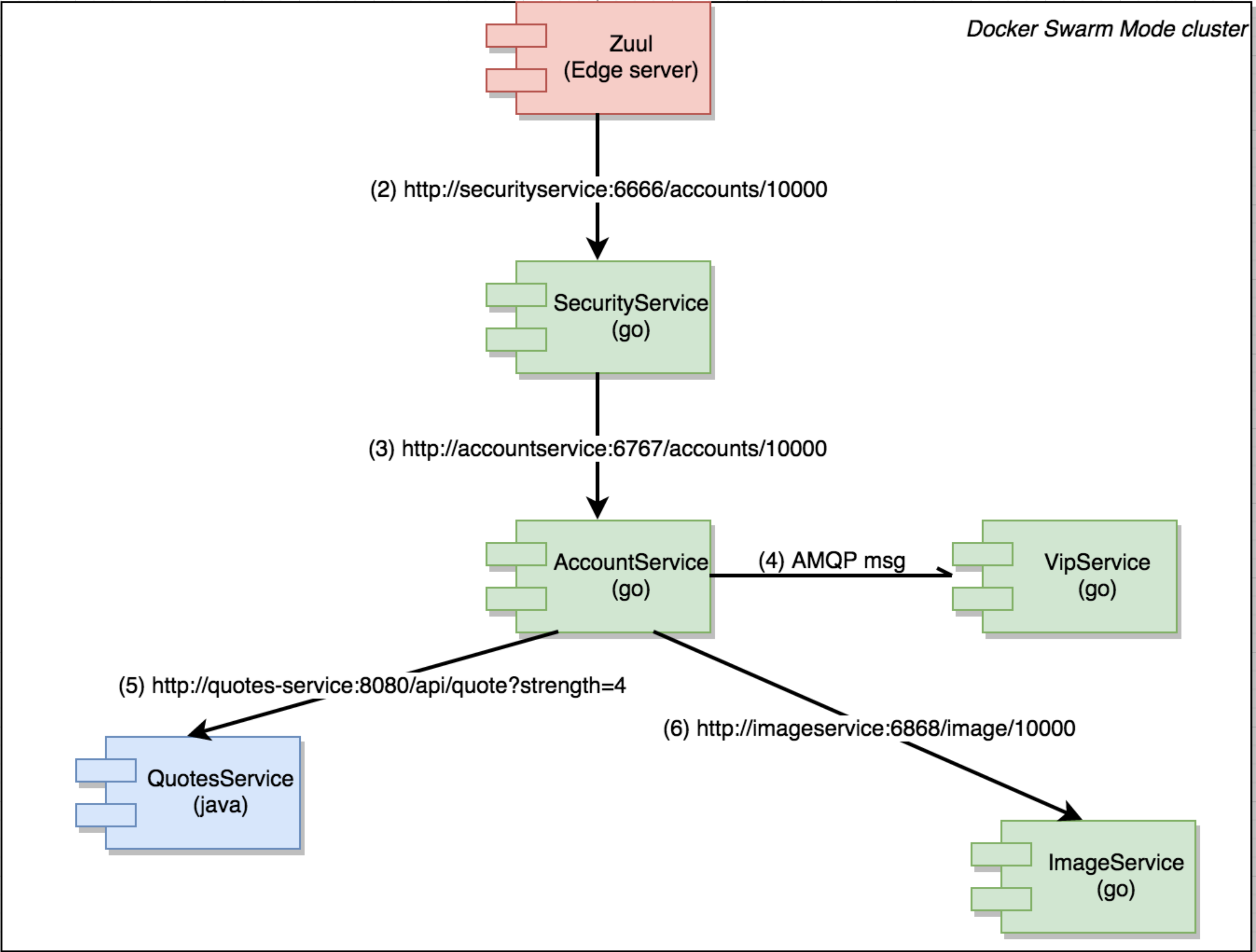
```
output := make(chan []byte, 1)
errors := hystrix.Go("get_account", func() error {
        output <- getData(accountId)
        return nil
}, func(err error) error {
        // fallback method here
        return nil
})
```

CALLISTA
— ENTERPRISE —

- Hystrix stream aggregation using customized Netflix Turbine

Go Services

:8181/hystrix.stream

CB / TA

:8181/hystrix.stream

CB / TA

:8181/hystrix.stream

CB / TA

Hystrix Stream
aggregation
(Modified Netflix
Turbine)

:8282/turbine.stream

Monitor
Dashboard
(Hystrix Dashboard)

Client Discovery token

Client Discovery token

RabbitMQ

CALLISTA
— ENTERPRISE —

Demo 5 - Hystrix Dashboard

CALLISTA
— ENTERPRISE —

«http client»
CURL

(1) https://192.168.99.100:8765/api/secured/accounts/1000

Zuul
(Edge server)

Docker Swarm Mode cluster

(2) http://securityservice:6666/accounts/10000

SecurityService
(go)

(3) http://accountservice:6767/accounts/10000

AccountService
(go)

(4) AMQP msg

VipService
(go)

(5) http://quotes-service:8080/api/quote?strength=4

QuotesService
(java)

(6) http://imageservice:6868/image/10000

ImageService
(go)

SUMMARY

- Go is an interesting option for microservices due to runtime characteristics and rather pleasant developing.
  - Although but not without it's fair share of quirks especially regarding the lack of traditional OO constructs and missing generics.
- Microservice development in Go requires a bit of work regarding integration with supporting services, but can be mitigated by using integration libraries such as go-kit or our own little toolkit.
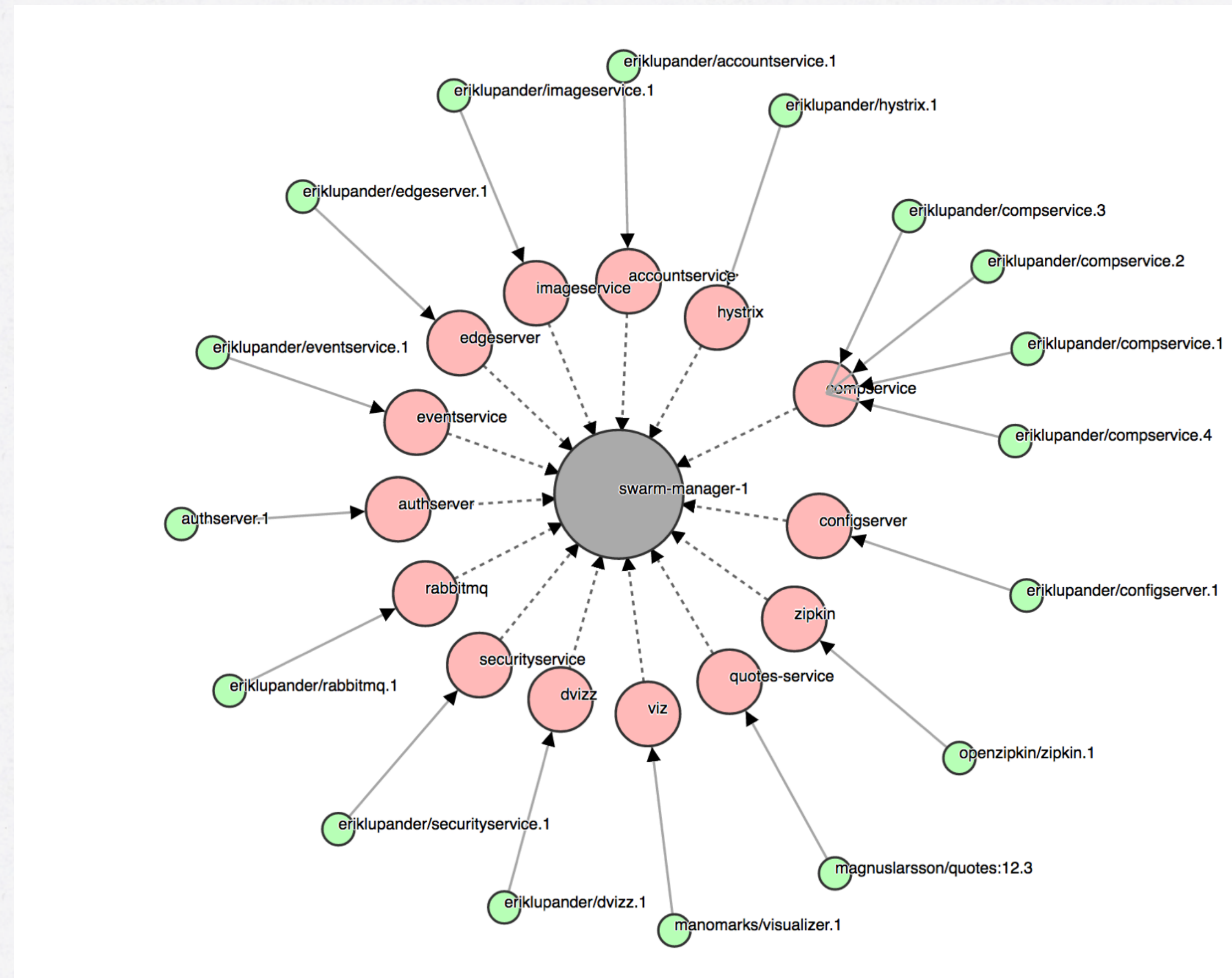 - Don't be afraid to pick your favorite libraries!

CALLISTA

WANT TO LEARN MORE?

- Nic Jackson
- July 2017 from Packt
- Technical reviewers:
  - Magnus Larsson
  - Erik Lupander

Nic Jackson

Building Microservices with Go

Build seamless, time efficient and robust microservices with Go

Packt>

CALLISTA
— ENTERPRISE —

DVIZZ - A DOCKER SWARM VISUALIZER

- https://github.com/eriklupander/dvizz
- Pull requests are more than welcome!

RESOURCES

- My 12-part blog series: http://callistaenterprise.se/blogg/teknik/2017/02/17/go-blog-series-part1/
- Demo landscape source code: https://github.com/callistaenterprise/goblog
  - Branch "nov2017"
- Spring Cloud Netflix: https://cloud.spring.io/spring-cloud-netflix/
- go-kit: https://github.com/go-kit/kit
- dvizz: https://github.com/eriklupander/dvizz
- packt book: https://www.packtpub.com/application-development/building-microservices-go

CALLISTA
— ENTERPRISE —

Questions?

CALLISTA
— ENTERPRISE —