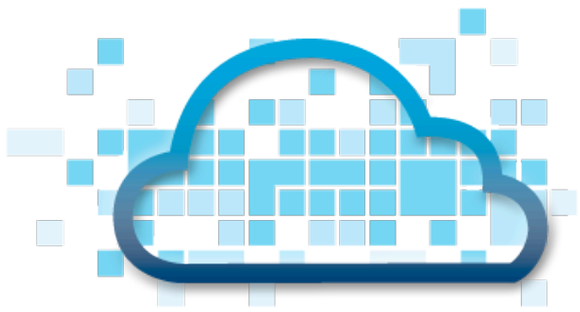


CLOUD FOUNDRY™

Design and Architecture

Derek Collison

What is Cloud Foundry?

The Open Platform as a Service

What is PaaS?

Or more specifically, aPaaS?

aPaaS

- Application Platform as a Service
- Applications and Services

aPaaS

- Application Platform as a Service
- Applications and Services
- Not
  - VMs
  - Memory
  - Storage
  - Networks
  - CPU

What is OpenPaaS?

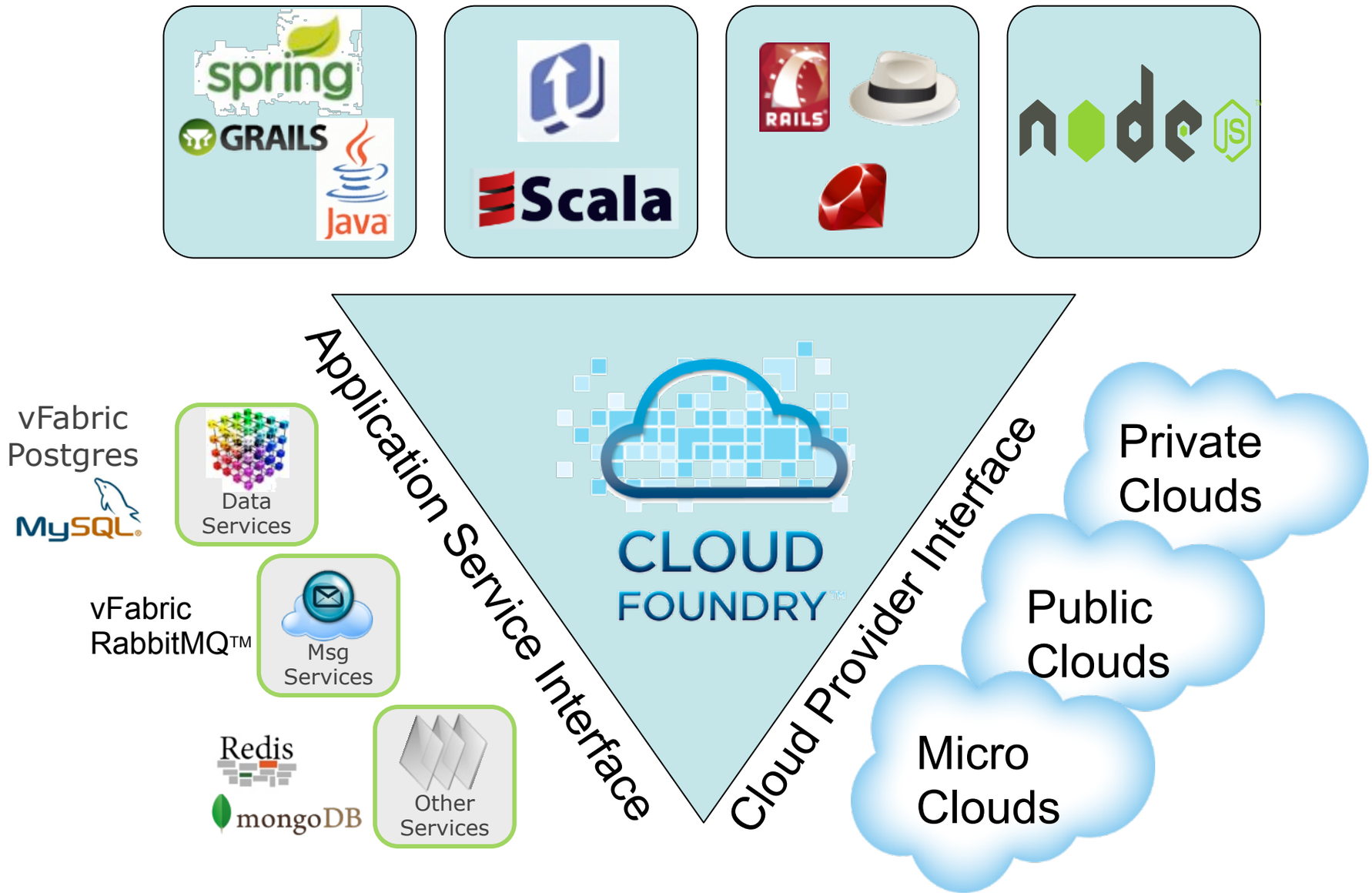
OpenPaaS

- Multi-Language
- Multi-Framework
- Multi-Services
- Multi-Cloud, Multi-IaaS
- Hybrid - Public or Private or Both
- OpenSource

OpenPaaS

- Multi-Language
  - Ruby, Java, Scala, Node.js, Erlang, Python, PHP..
- Multi-Framework
  - Rails, Sinatra, Spring, Grails, Express, Lift
- Multi-Services
  - MySQL, Postgres, MongoDB, Redis, RabbitMQ
- Multi-Cloud, Multi-IaaS
  - vSphere, MicroCloud, OpenStack, AWS

The Open PaaS



What is our Goal?

What was our Goal?

Raise the unit of currency to be the application and its associated services, not the infrastructure

What was our Goal?

Best of breed delivery platform for all modern applications and frameworks

What was our Goal?

Favor Choice

and

Openness

How was it Built?

How was it Built?

- Kernel (CloudFoundry OSS)
  - Core PaaS System
- Kernel and Orchestrator Shells
  - Layered on top of IaaS
- Orchestrator
  - IaaS creation, management and orchestration

High Level

Clients (VMC, STS, Browser)

CF Kernel

Orchestrator

IaaS

Hardware - CPU/Memory/Disk/Network

Basic Premises

- Fail Fast
- Self Healing
- Horizontally Scalable Components
- Distributed State
- No Single Point of Failure
- Should be as simple as possible

Basic Patterns

- Event-Driven
- Asynchronous
- Non-blocking
- Independent, Idempotent
- Message Passing
- Eventually Consistent

Basic Design

- All components loosely coupled
  - Few “Classes”, many “Instances”
- Messaging as foundation
  - Addressing and Component Discovery
  - Command and Control
- JSON payloads
- HTTP or File/Blob for data transport

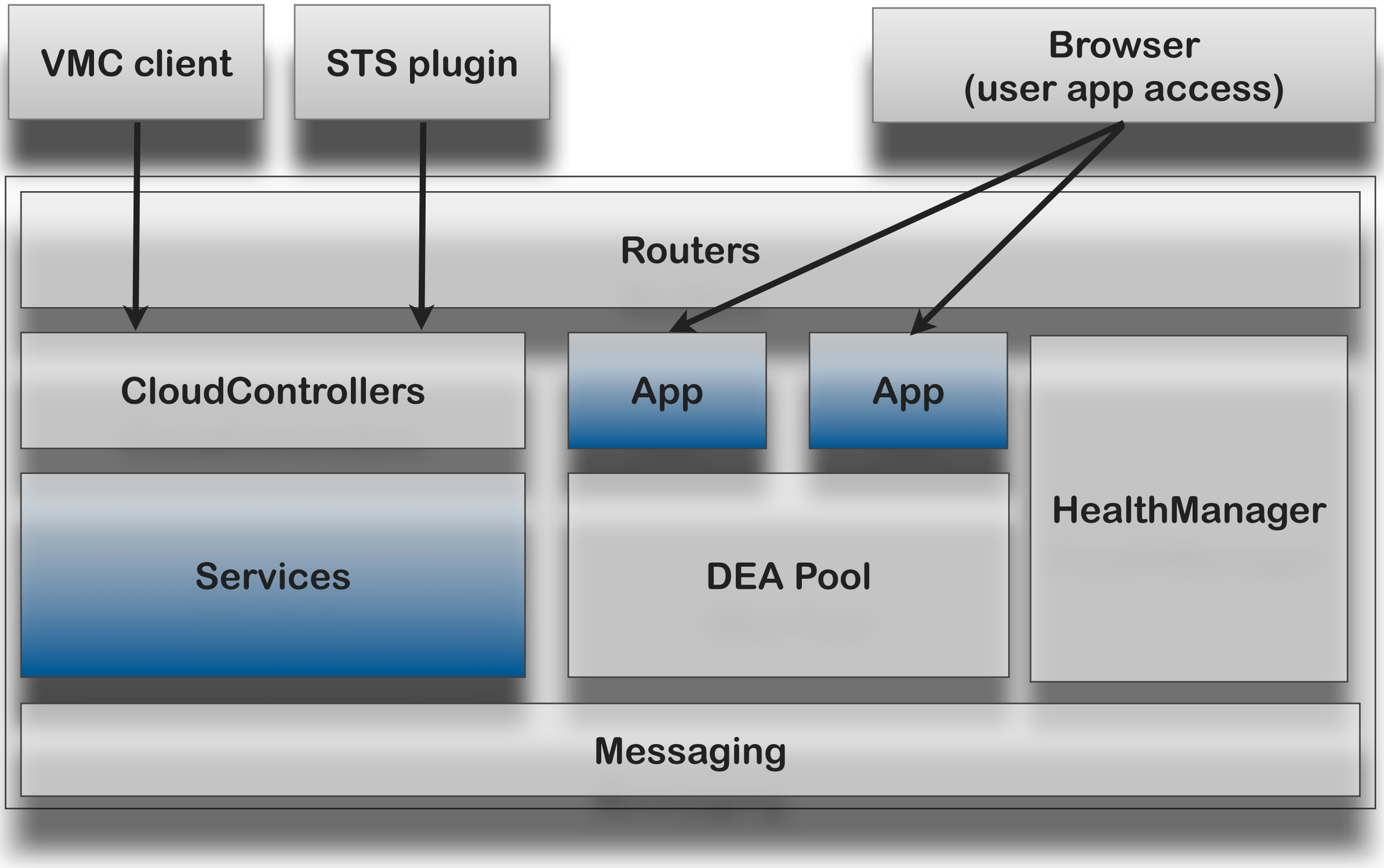
Kernel Components

- All dynamically discoverable
- Launch and scale in any order
- Can come and go as needed
- Monitor via HTTP and JSON
- Location independent

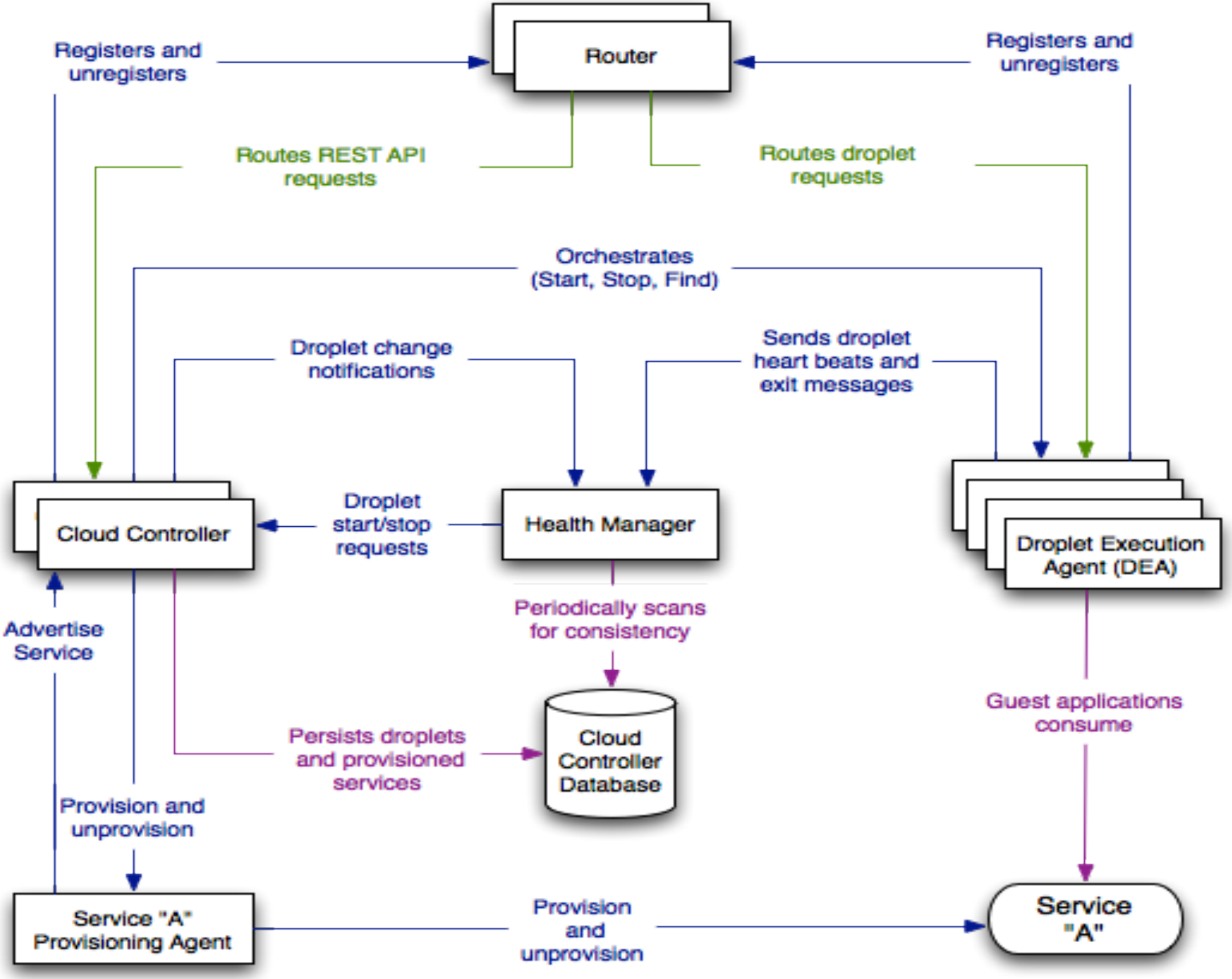
Kernel Components

- Router
- CloudController
- DEA
- HealthManager
- Service Provisioning Agent
- Messaging System

Logical View



Architecture

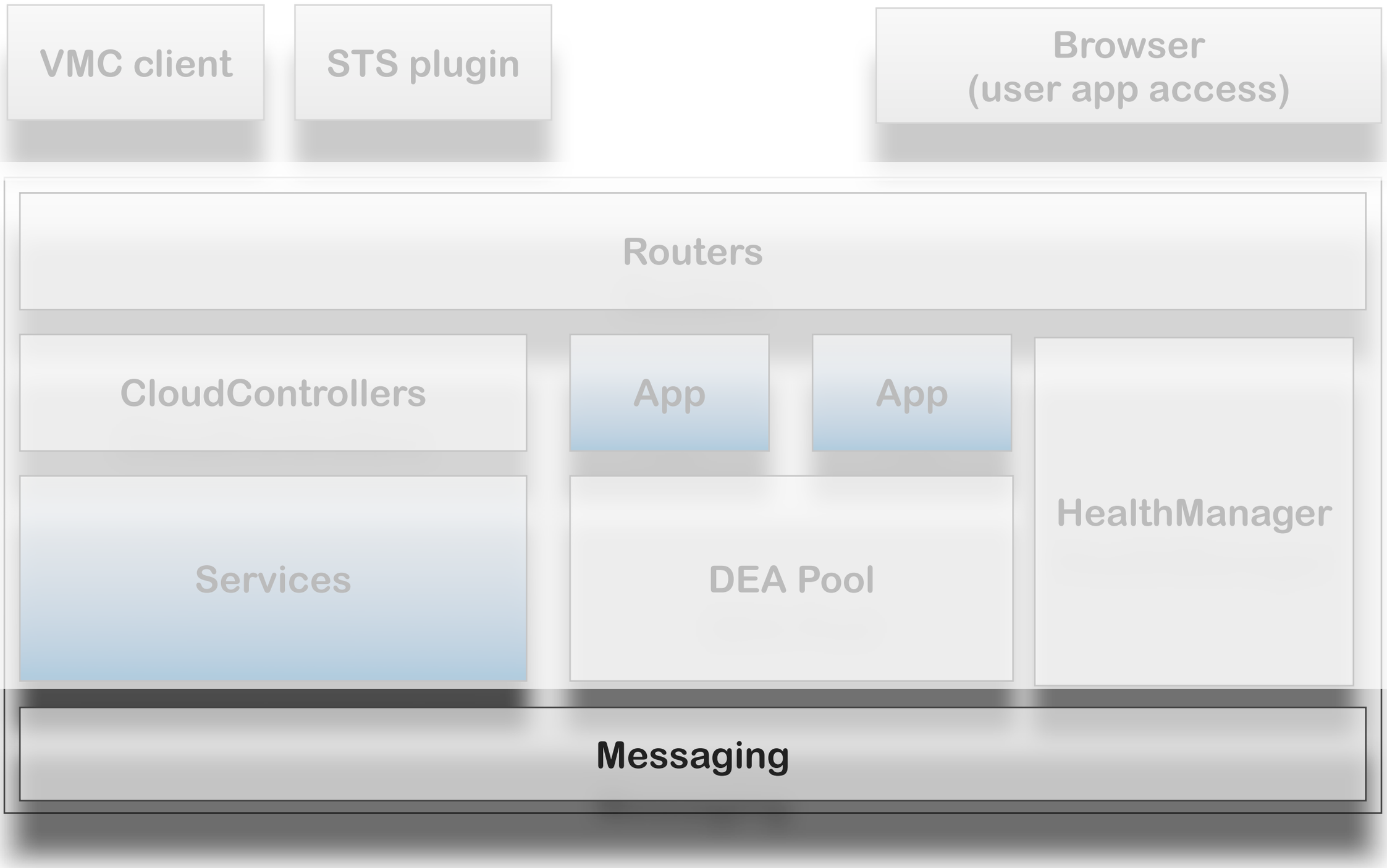


Messaging

Messaging

“The Nervous System”

Messaging


VMC client
STS plugin
Browser
(user app access)
Routers
CloudControllers
App
App
HealthManager
Services
DEA Pool
Messaging

Messaging

- Addressing and Discovery
  - No static IPs or DNS lookups req'd
  - Just Layer 4
- Command and Control
  - Central communication system
  - Dial tone, fire and forget
  - Protects *itself* at all costs
  - Idempotent semantics

Router

Router

“Traffic Cop”

Router

VMC client

STS plugin

Browser
(user app access)

Routers

CloudControllers

App

App

HealthManager

Services

DEA Pool

Messaging

Router

- Handles all HTTP traffic
- Maintains distributed routing state
- Routes URLs to applications
- Distributes load among instances
- Realtime distributed updates to routing tables from DEAs

CloudController

CloudController

“The King”

CloudController

VMC client

STS plugin

Browser
(user app access)

Routers

CloudControllers

App

App

HealthManager

Services

DEA Pool

Messaging

CloudController

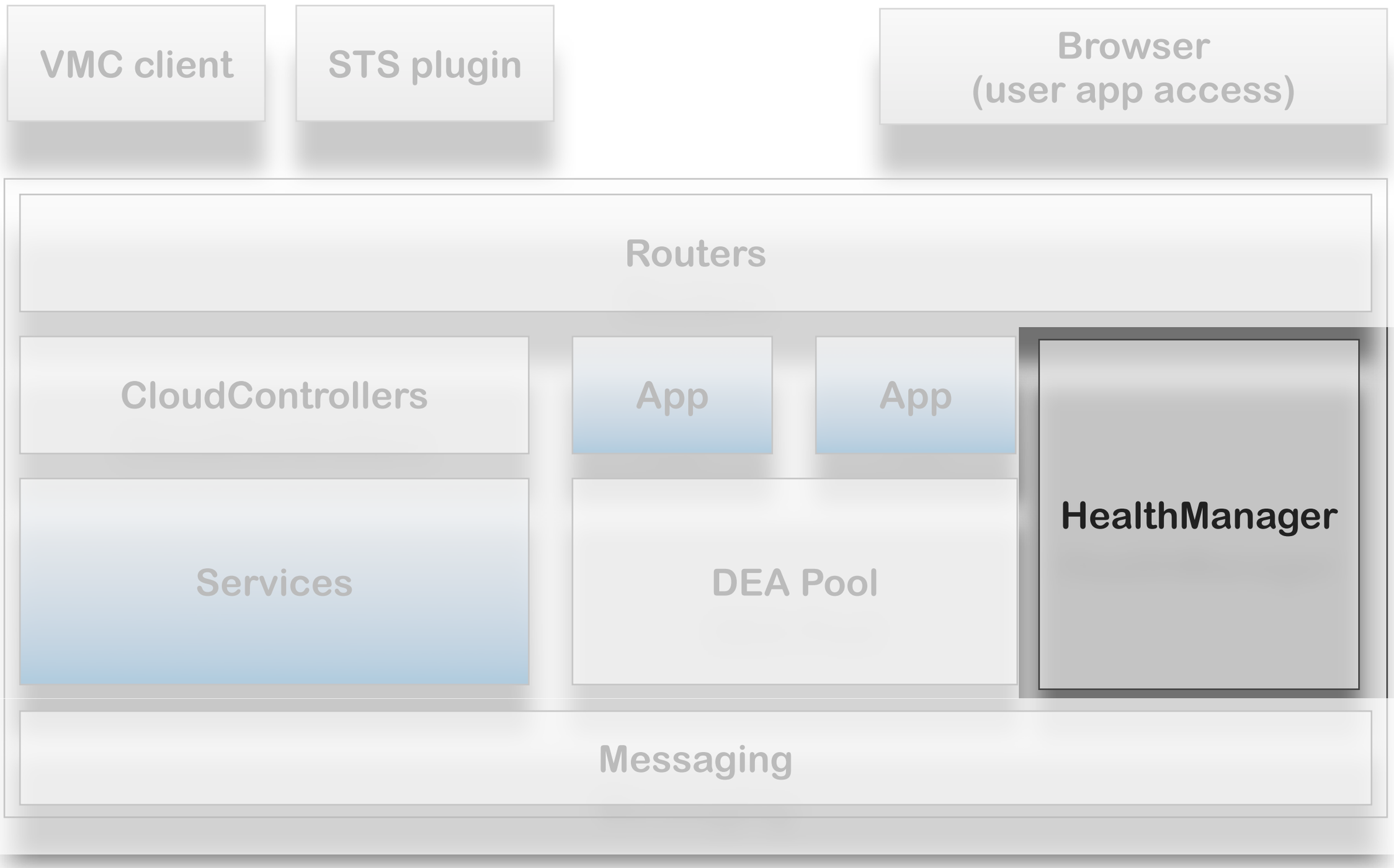
- Handles all state transitions
- Deals with users, apps, and services
- Packages and Stages applications
- Binds Services to Applications
- Presents external REST API

HealthManager

HealthManager

“Court Jester”

HealthManager



HealthManager

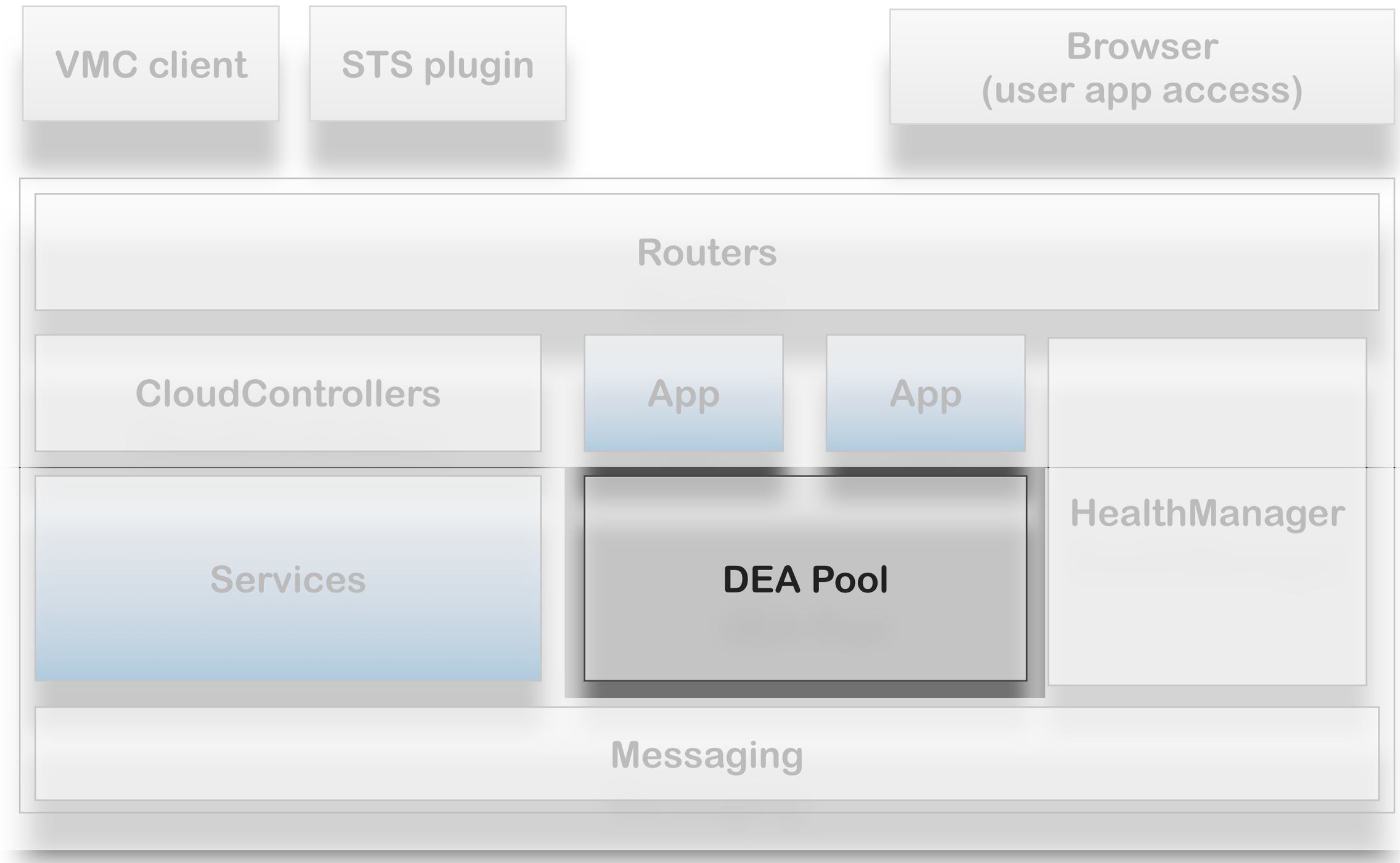
- Monitors the state of the world
- Initial value with realtime delta updates to “intended” vs “real”
- Determines drift
- Complains to the CloudControllers when something is not correct
- No power to change state itself

DEA

DEA

“Droplet Execution Agent”

DEA


VMC client
STS plugin
Browser
(user app access)
Routers
CloudControllers
App
App
Services
DEA Pool
HealthManager
Messaging

DEA
(Droplet Execution Agent)

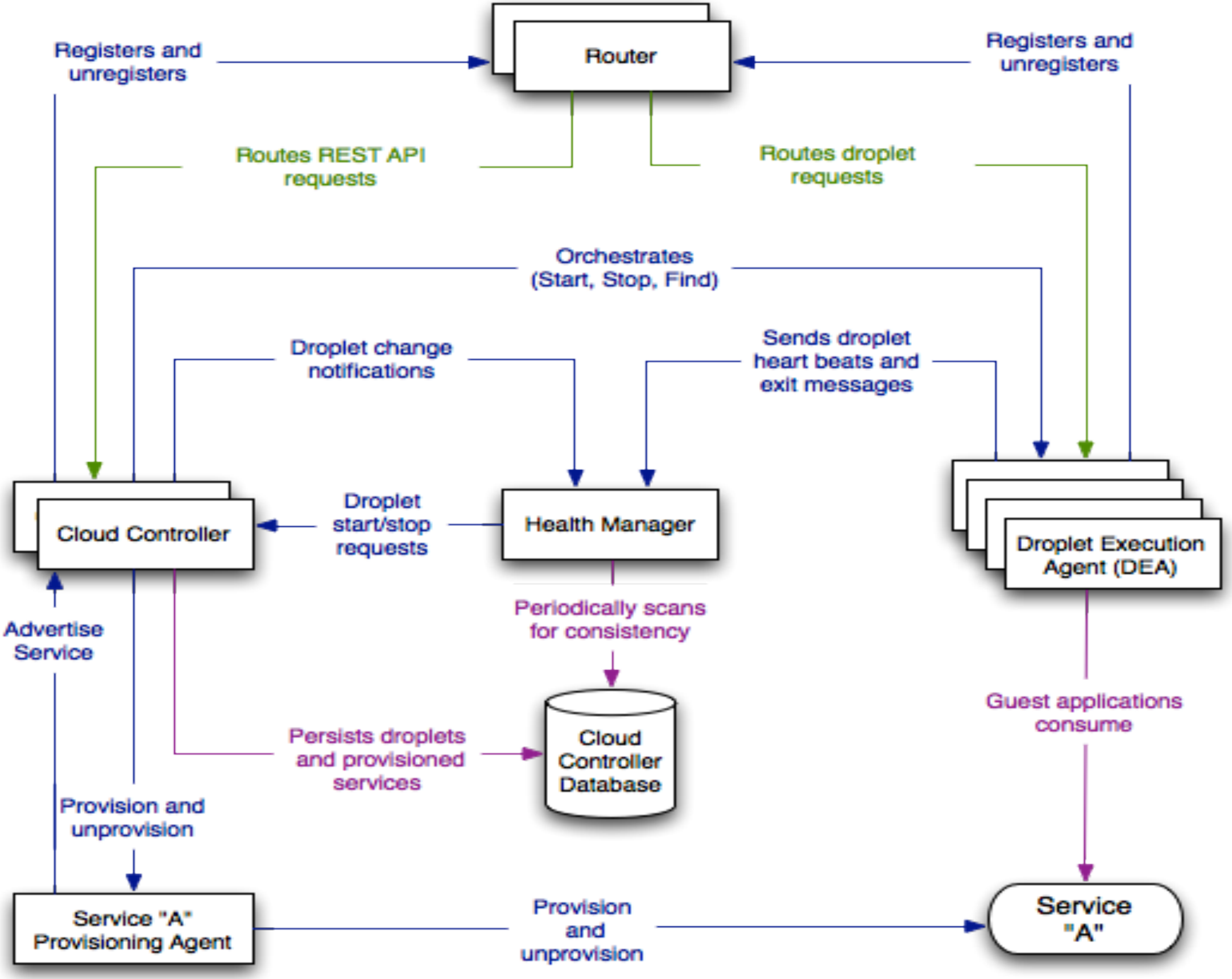
- Responsible for running all applications
- Monitors all applications
  - CPU, Mem, IO, Threads, Disk, FDs, etc
- All apps look same to DEA
  - start and stop
- Express ability and desire to run an application
  - runtimes, options, cluster avoidance, memory/cpu
- Alerts on any change in state of applications
- Provides secure/constrained OS runtime
  - Hypervisor, Unix File and User, Linux Containers*
  - Single or Multi-Tenant

How does it all Work?

Pushing an App

- Client (VMC/STS) pushes meta-data to CC
- Client optionally pushes resource signatures (diff analysis, sys wide)
- Client pushes app resources to CC
- CC puts app together
- CC stages app asynchronously
- CC binds and stages services
- Droplet ready

Architecture



Running an App

- CC asks DEAs for “help”
- First DEA back wins! Simple
- CC sends start request to selected DEA
- DEA pushes the “green” button
- DEA waits and monitors pid and ephemeral port for app to bind
- When app is healthy, sends “register” message
- Register message is seen by HM and Routers
- Routers bind URL to host:port

DEAs answer?

- DEAs first determine YES or NO
  - correct runtime, options, memory, etc
- Then calculate a Delay Taint
  - SHA hash of application
  - memory
  - cpu
- Taint allows balancing and selection

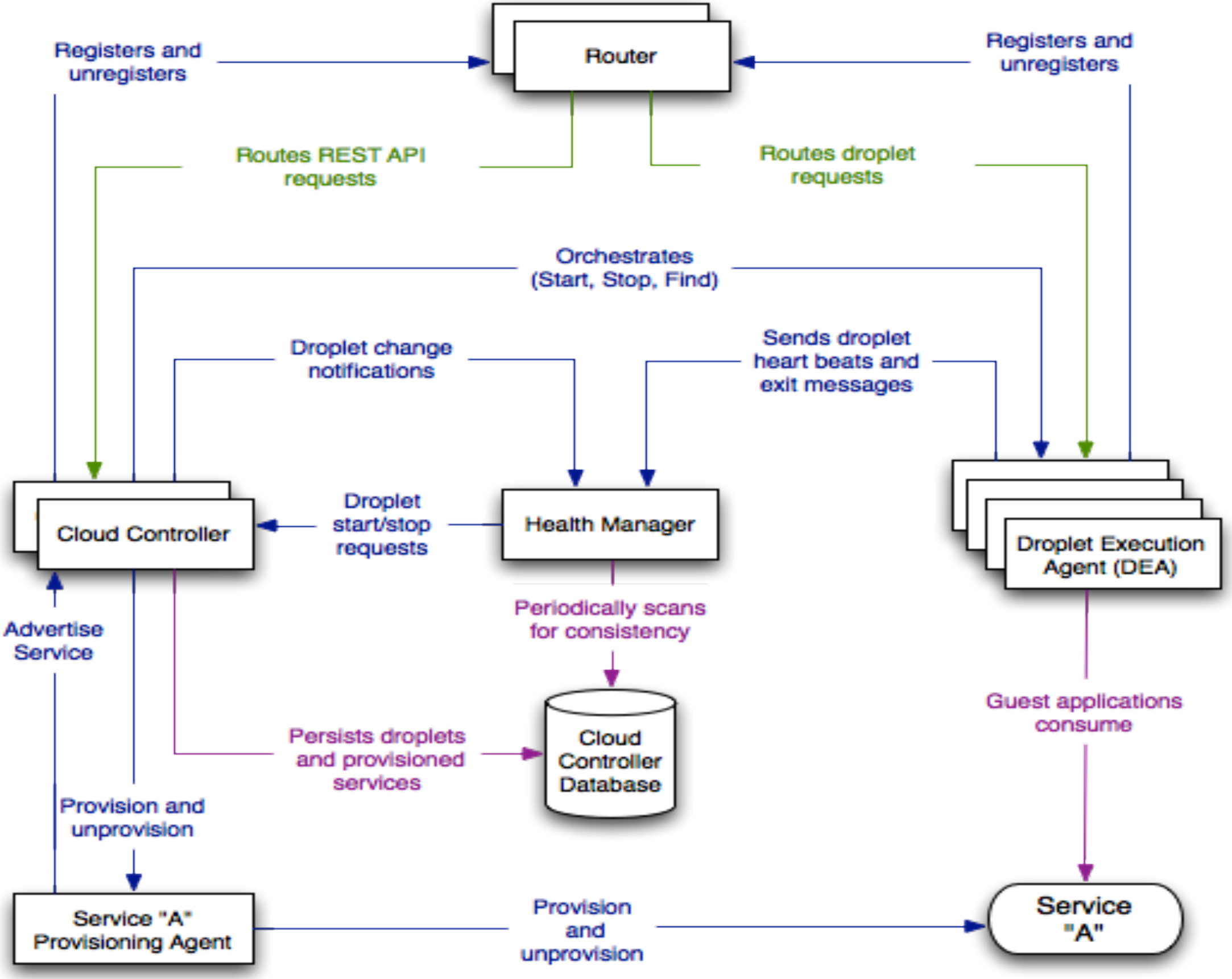
Scale up & down?

- Exact steps as running the app the first time
- SHA1 taint helps avoid clustering
- memory/cpu taint helps distribute as evenly as possible
- Nothing pre-computed
- Nothing assumed

Crashes?

- If your app stops and we did not tell it to, that is a crash
- Crashed apps are immediately detected by DEA and messaged
- Routers disconnect route instantly
- HM will signal CC
  - something is wrong
- CC will issue run sequence again

Architecture

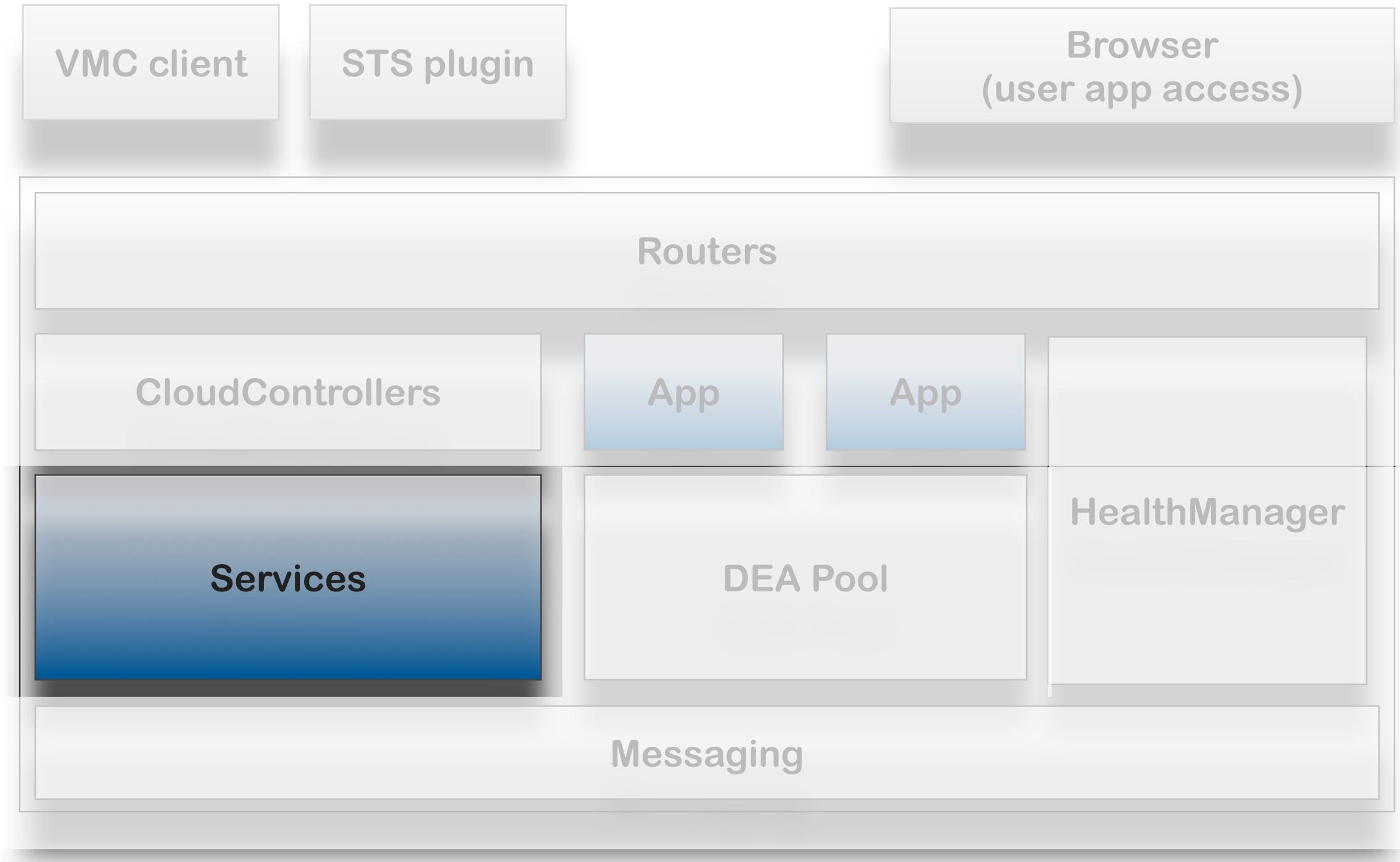


Access to my App?

- All routers understand where all instances of your application are running
- Will randomly pick backend, not semantically aware.
- Will remove routes that are stale or unhealthy
- Session stickiness and replication available, but best to avoid if possible

What about Services?

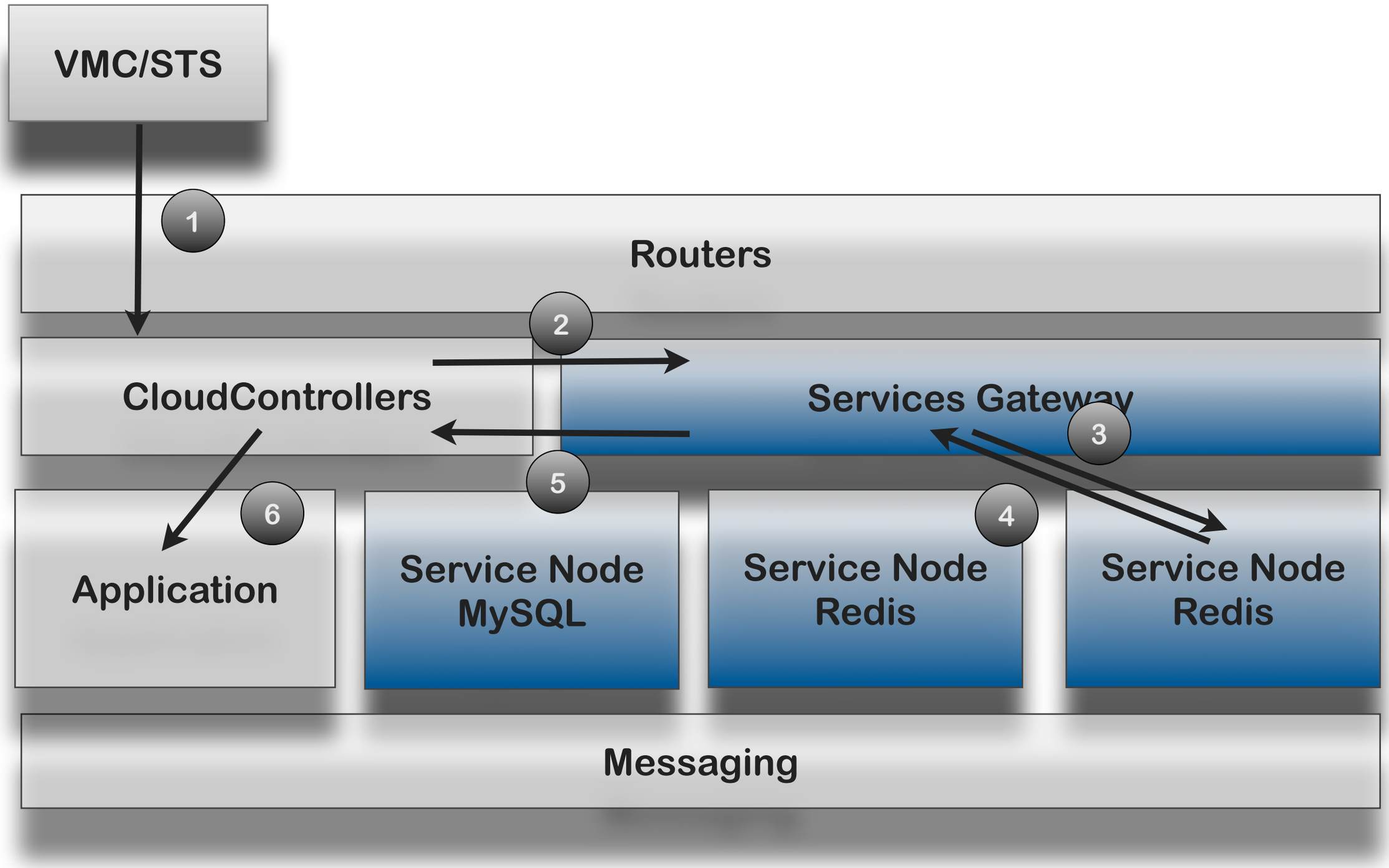
Services


VMC client
STS plugin
Browser
(user app access)
Routers
CloudControllers
App
App
Services
DEA Pool
HealthManager
Messaging

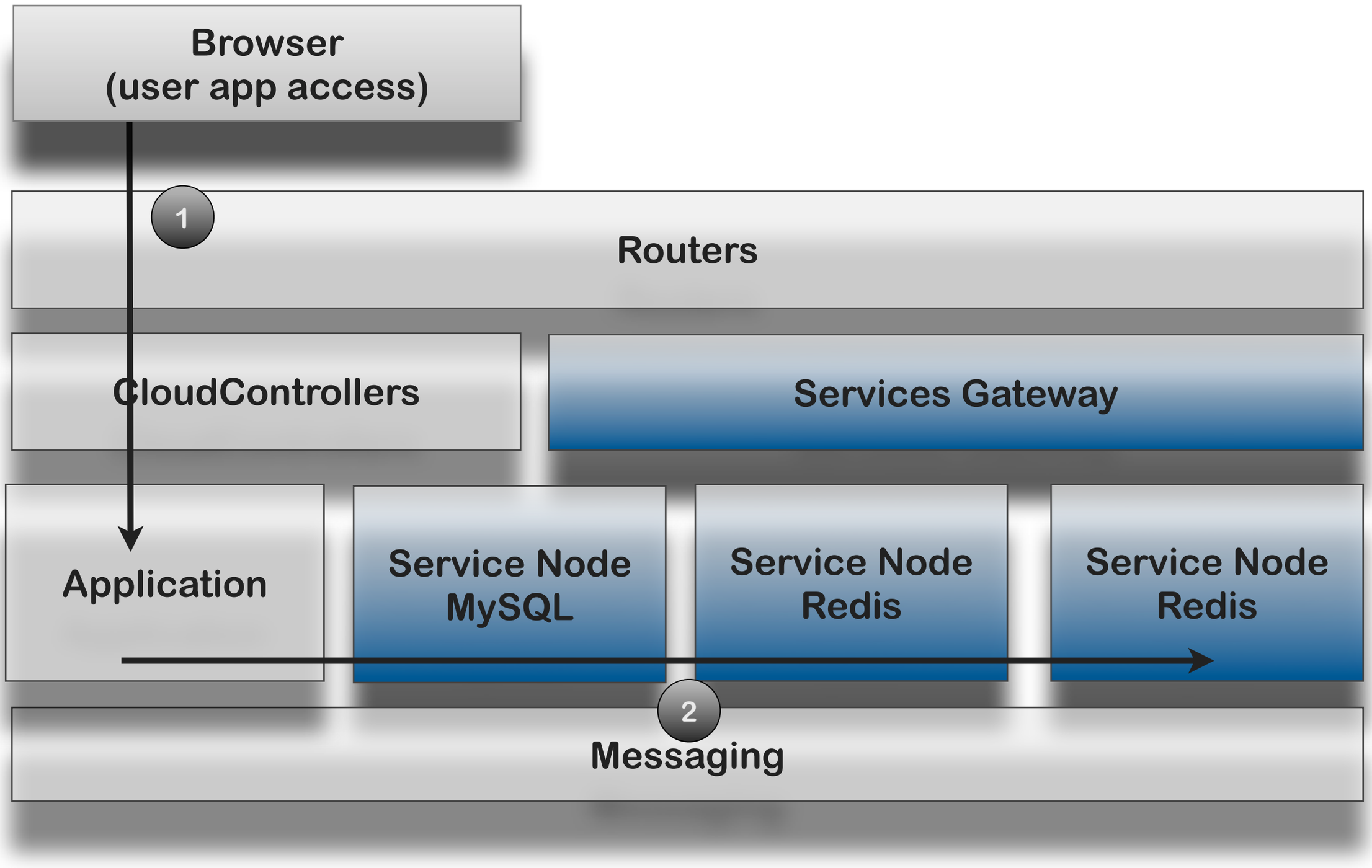
Services

- Service Advertisement
- Service Provisioning
- Gateway fronts multi-backends
- Service Nodes scale independent
- App and service talk directly
- API to register into system
- Closure for additional value

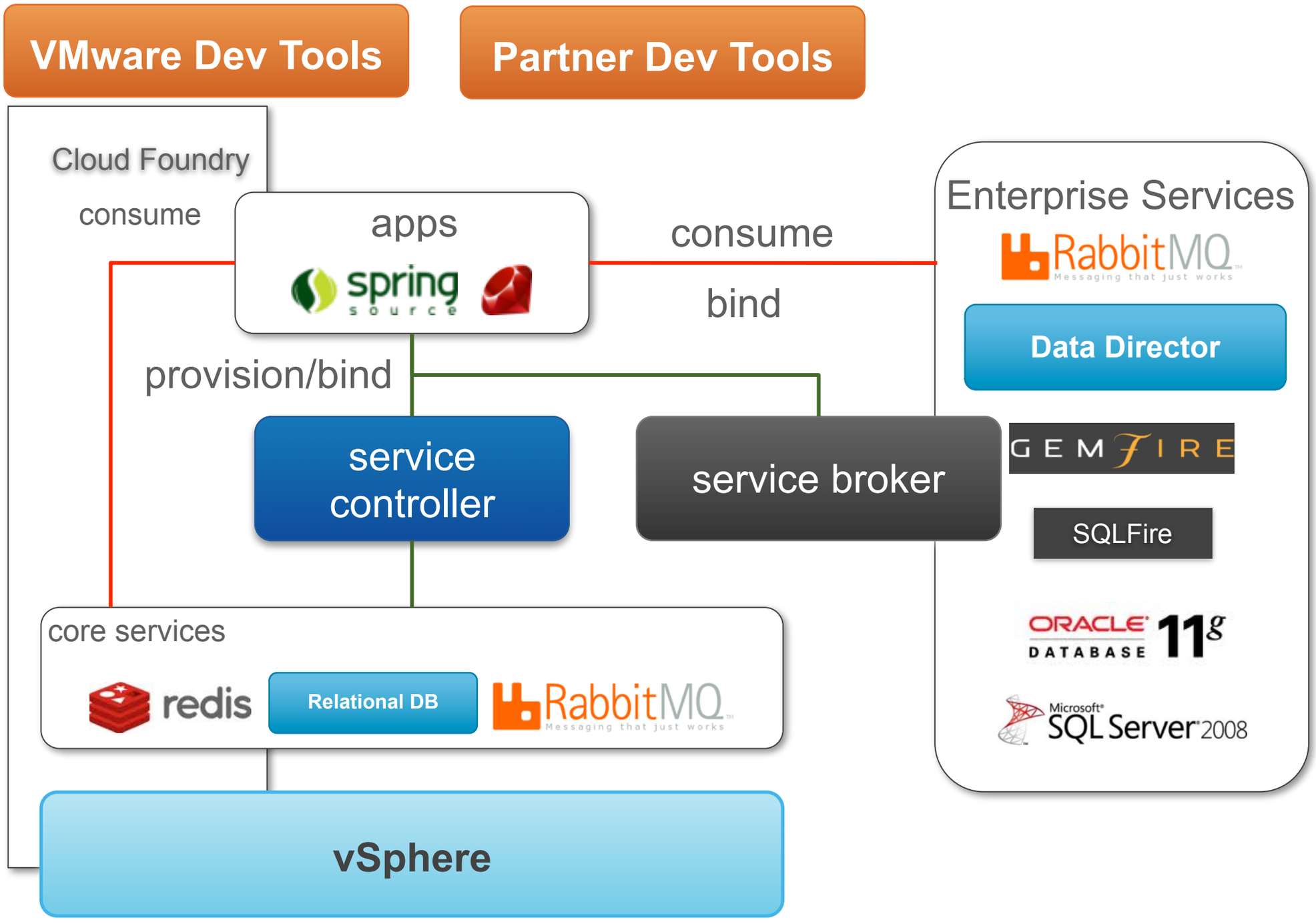
Provisioning



Access (Direct)



Services


VMware Dev Tools
Partner Dev Tools
Cloud Foundry
consume
apps
spring source
consume
bind
provision/bind
service controller
service broker
Enterprise Services
RabbitMQ
Messaging that just works
Data Director
GEMFIRE
SQLFire
ORACLE 11g DATABASE
Microsoft SQL Server 2008
core services
redis
Relational DB
RabbitMQ
Messaging that just works
vSphere

Learn more:

www.cloudfoundry.org

blog.cloudfoundry.com

support.cloudfoundry.com

Thank You

Questions?

dcollison@vmware.com

derek.collison@gmail.com

twitter: derekcollison