

Cloud Foundry

The Building of the Open PaaS

Derek Collison
July 27, 2011

What is Cloud Foundry?

The Open Platform as a Service

What is PaaS?

**Or more
specifically, an
aPaaS?**

aPaaS

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

What is OpenPaaS?

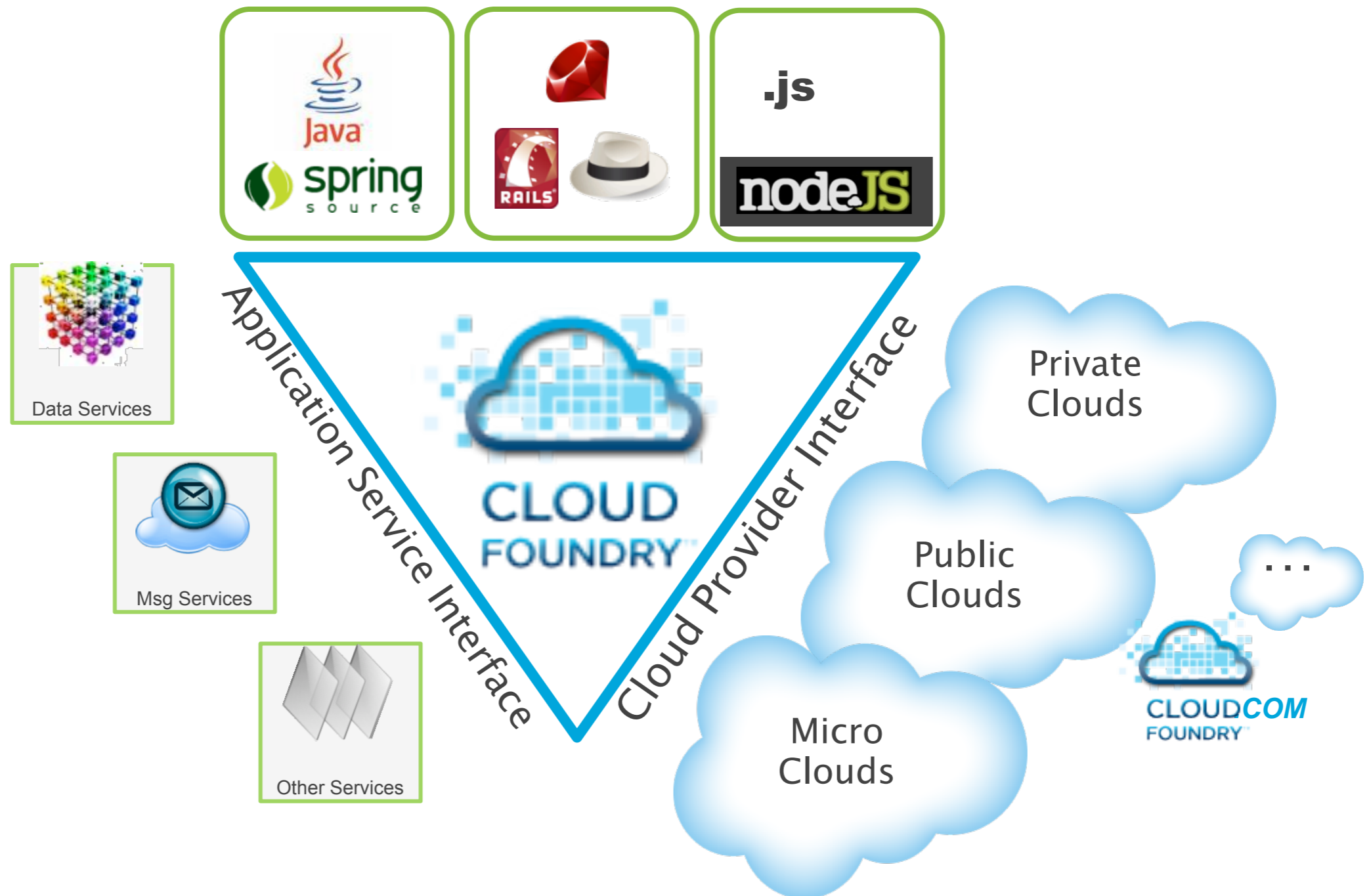
OpenPaaS

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

OpenPaaS

- **Multi-Language**
 - Java, Scala, Ruby, Node, Erlang, PHP..
- **Multi-Framework**
 - Spring, Grails, Express, Rails, Lift, MochiWeb
- **Multi-Services**
 - MySQL, Postgres, MongoDB, Redis, RabbitMQ
- **Multi-Cloud, Multi-iaaS**
 - vSphere, OpenStack, AWS, Eucalyptus

The Open PaaS



What was our Goal?

- **Raise the unit of currency to the be the application and its associated services, not the infrastructure**
- **Best of breed delivery platform for all modern applications and frameworks**
- **Favor Choice and Openness**
- **Simplicity and Speed**

How was it Built?

- **Kernel and Orchestrator Shells**
 - Layered on top of IaaS
- **Kernel**
 - Core PaaS System
- **Orchestrator**
 - Creation, management and orchestration of the infrastructure

High Level

Clients (VMC, STS, Browser)

CF Kernel

Orchestrator

IaaS

Hardware - CPU/Memory/Disk/Network

Basic Premises

- Assume failure
- Optimize for MTTR, not MTBF
 - MTTR = Mean Time To Recovery
 - MTBF = Mean Time Between Failures
- Fail Fast
- Self Healing
- Horizontally Scalable Components
- Distributed state, No single POF
- 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**

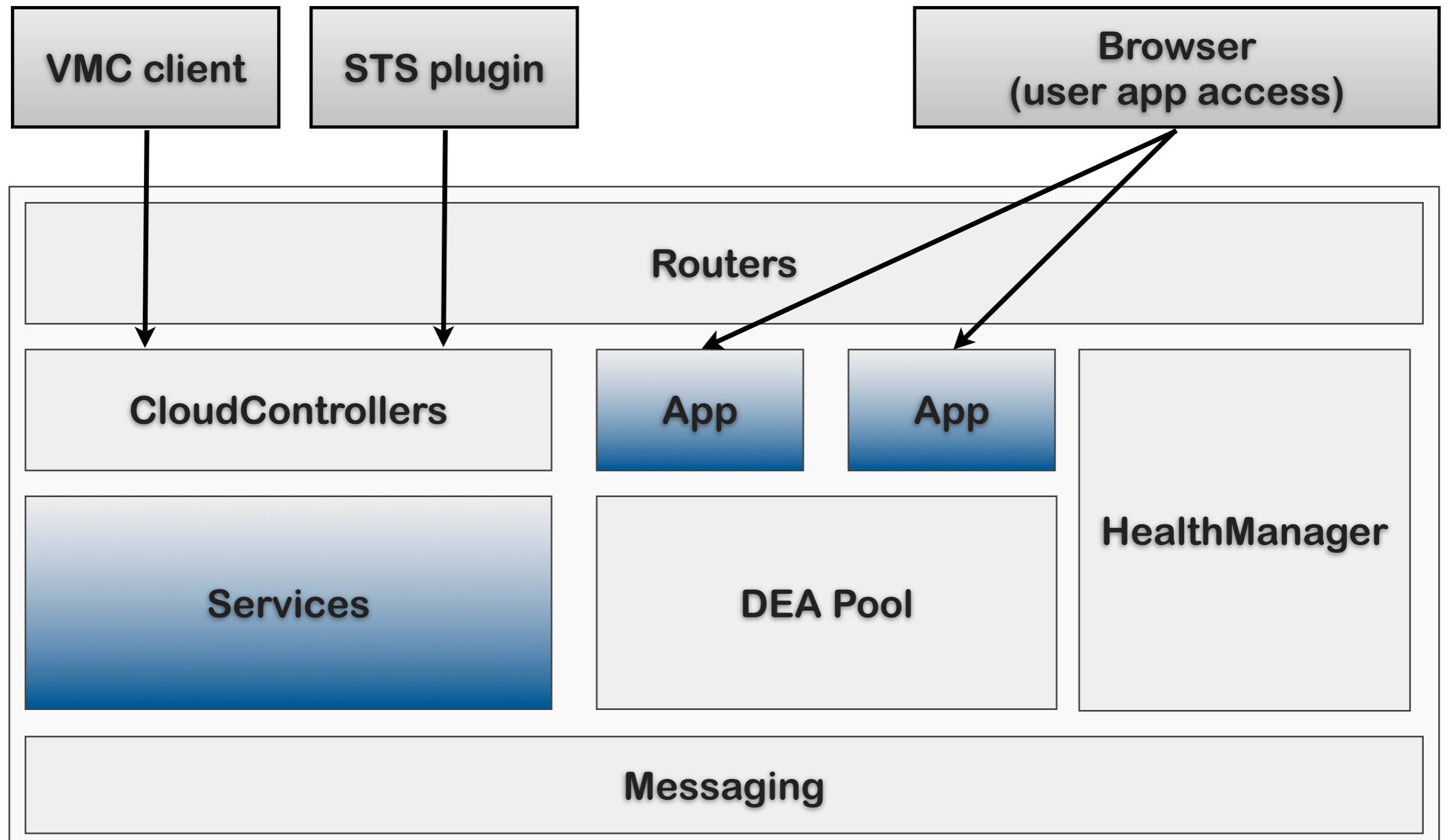
Kernel Components

- 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
- Messaging System

Logical View



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

- **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

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

HealthManager

- **Monitors the state**
- **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

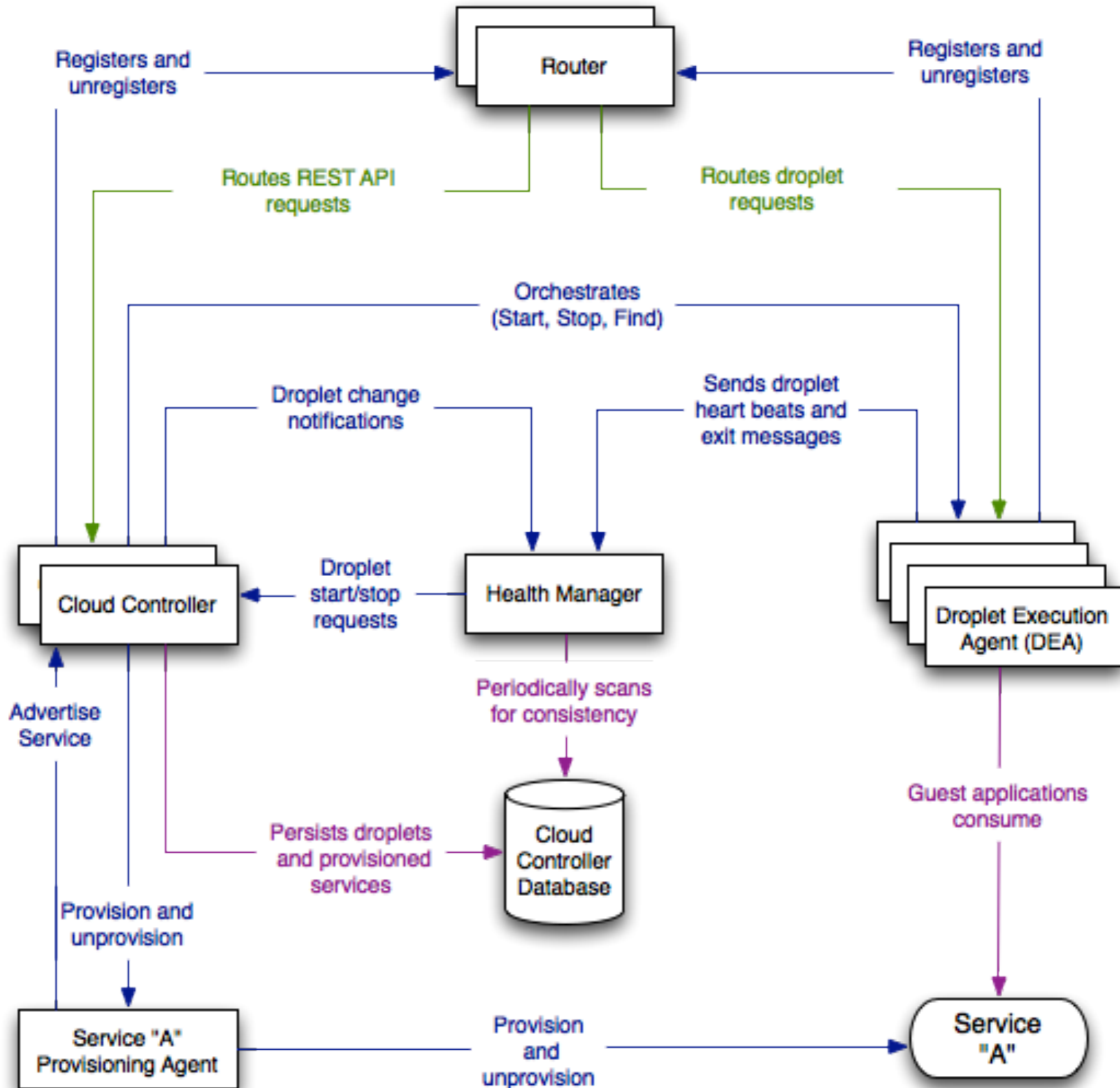
(Droplet Execution Agent)

- Responsible for running all applications
- Monitors all applications
 - CPU, Mem, IO, Threads, Disk, FDs, etc
- All apps look same to DEA, start and stop
- “concept” of 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

Services

- **One of the extensibility planes**
- **First class citizen**
- **Bound to applications**
- **Can be shared**
- **Services API to discover, list, and provision**
- **Direct access to service after provisioned**
- **Easy way to bind any service to an app**

Architecture

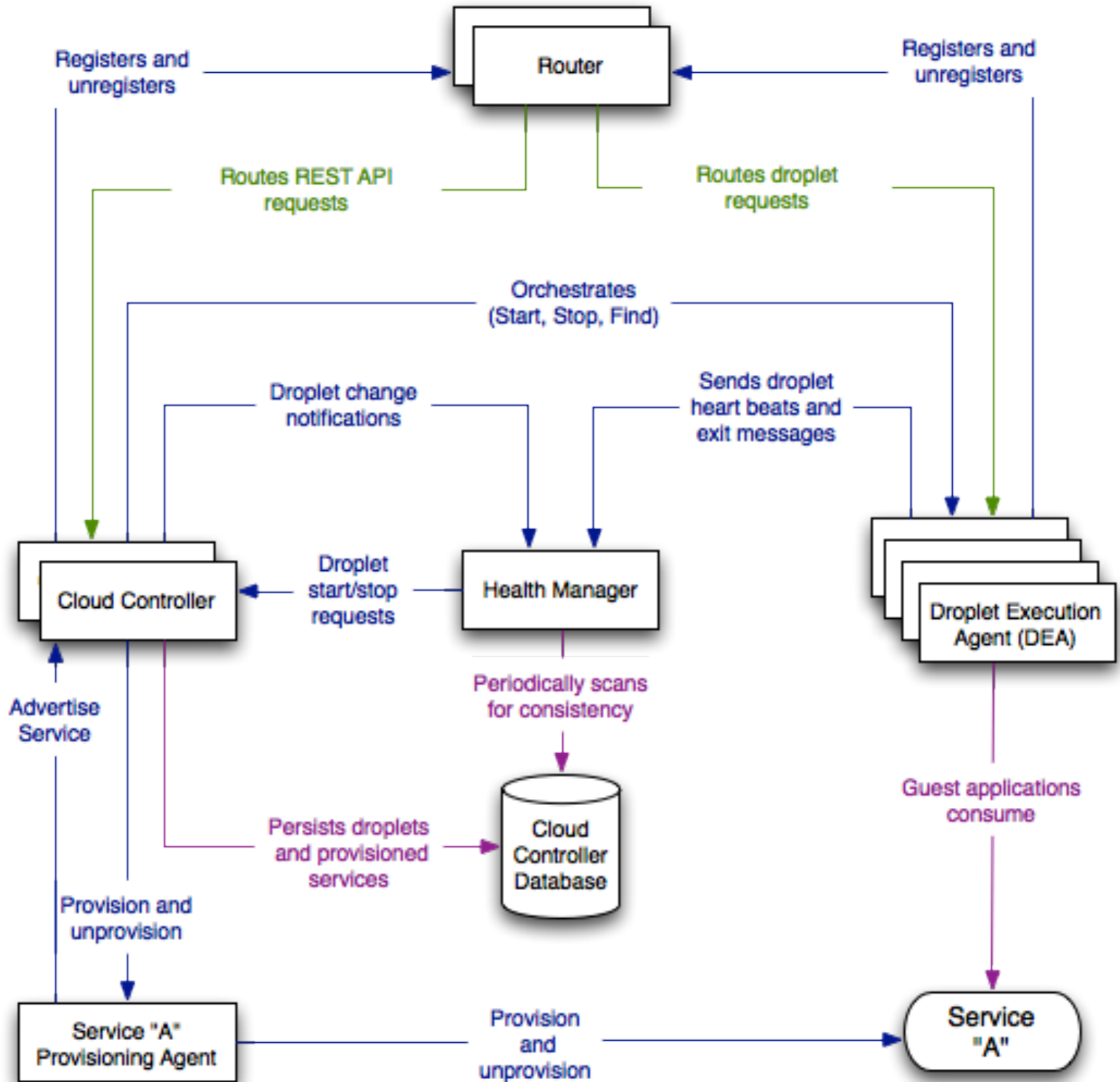


**How does it all
work?**

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**

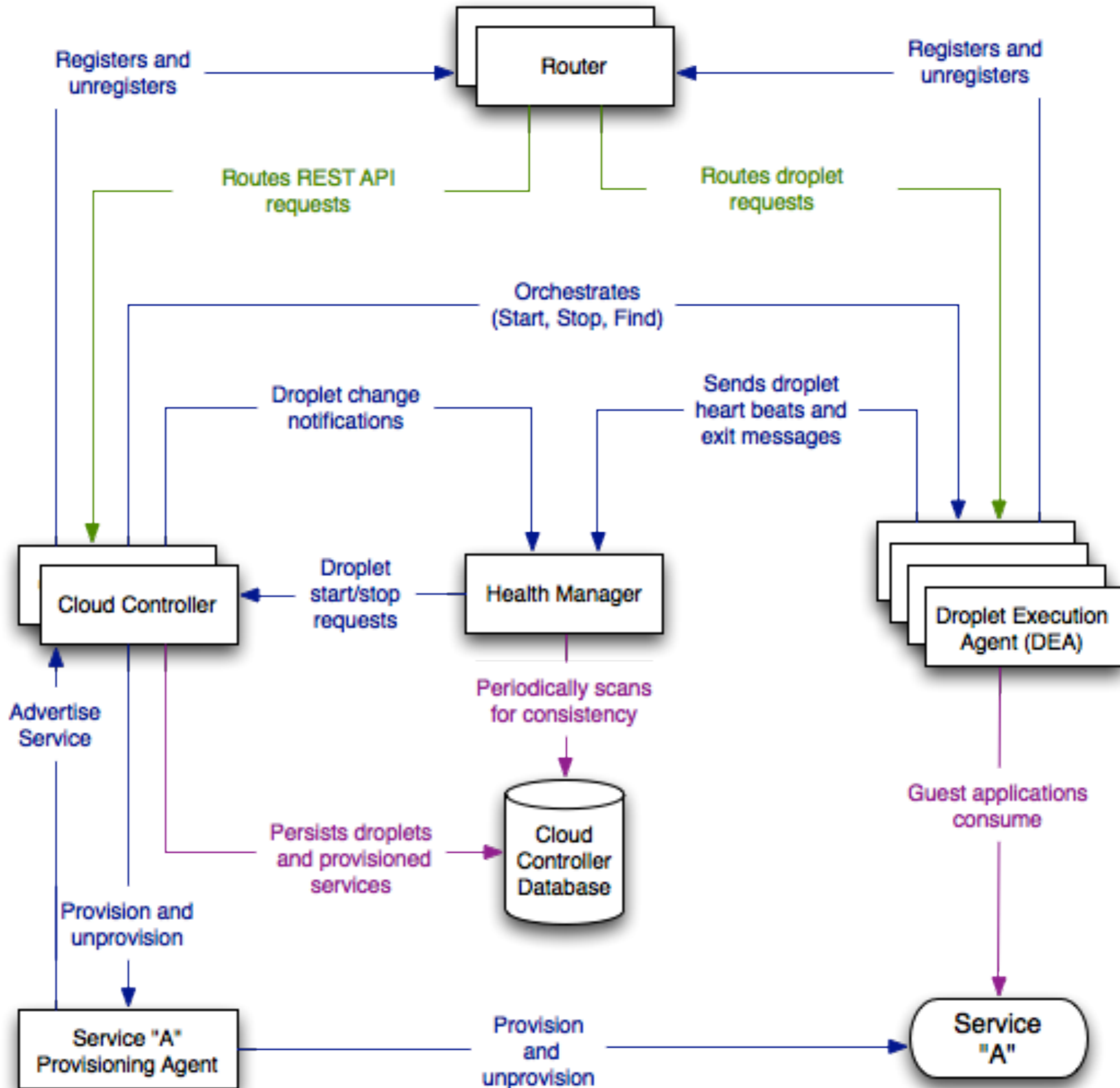
Architecture



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**

Architecture



DEAs answer?

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

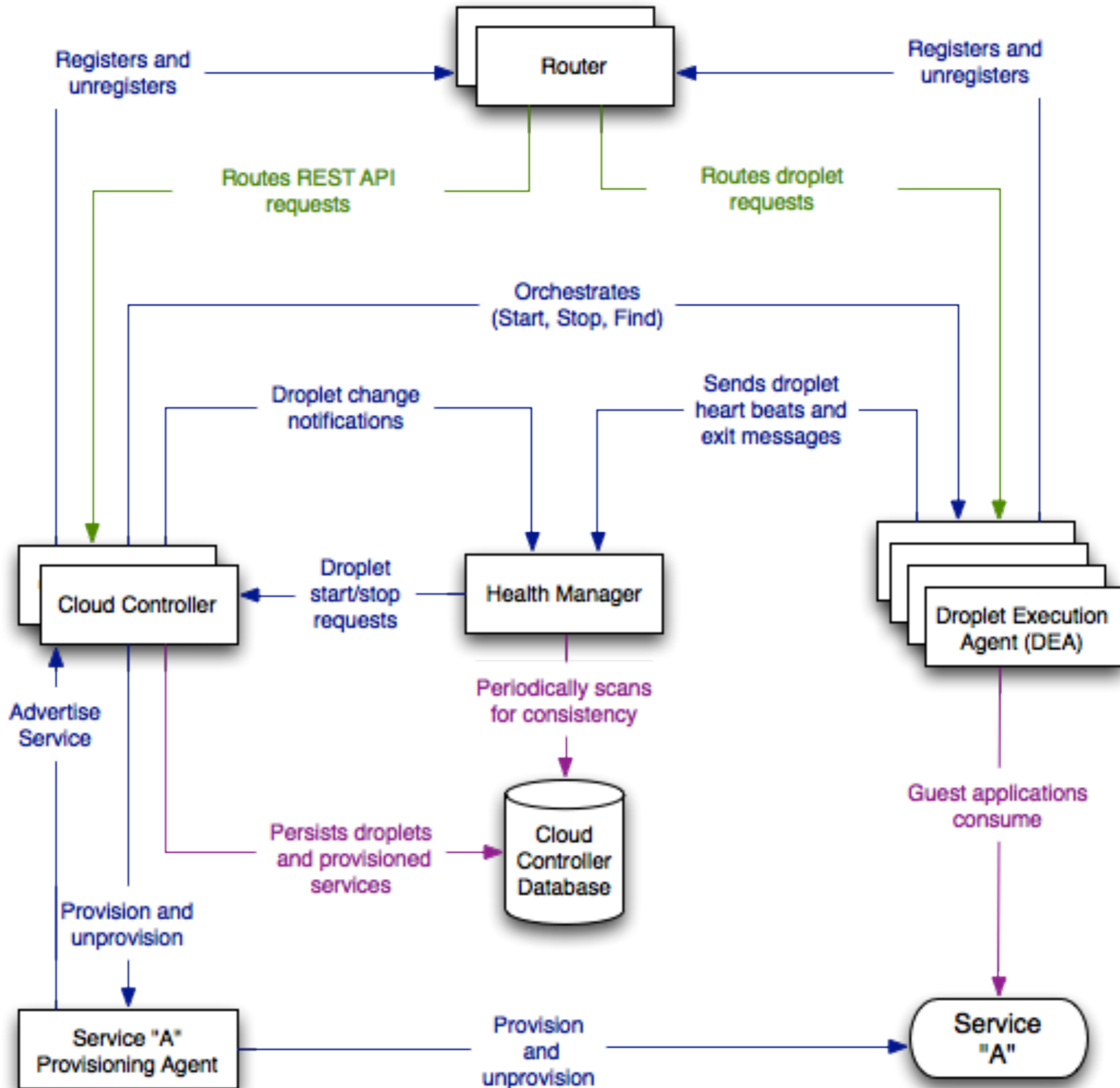
Scale up & down?

- **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 or strict**

Crashes?

- If your app stops and we did not tell it to, that is a crash
- Crashed apps are immediately detected by DEA and message sent
- Routers disconnect route
- HM will signal a CC that 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

Thank You

Questions?

dcollison@vmware.com

derek.collison@gmail.com

[@derekcollison](#)