Pivotal CF

Building Cloud ready apps

Pivotal

The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

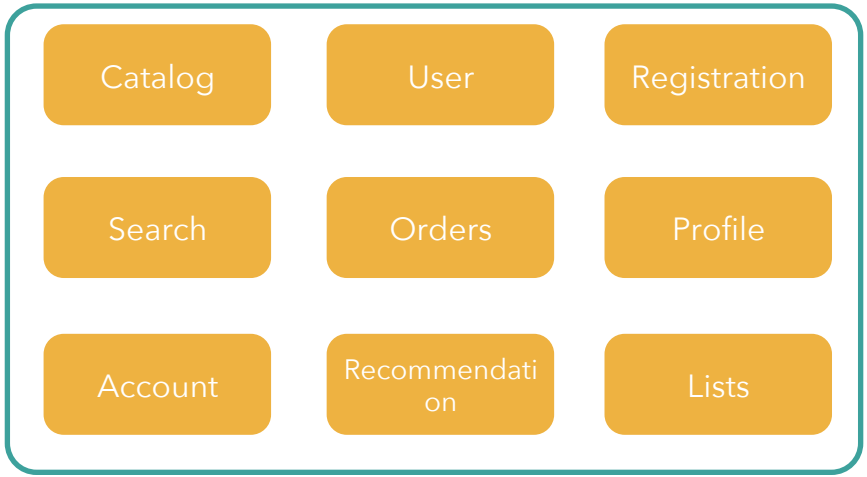
Resiliency

Traceability / Logging

Pivotal

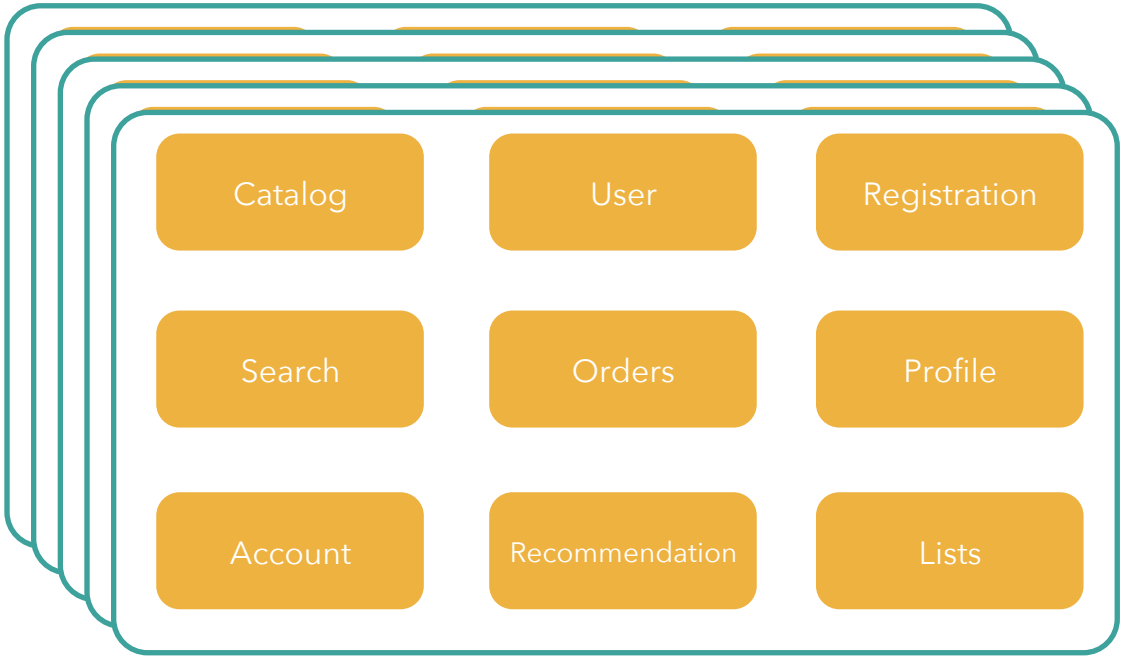
Scalability

- Traditional applications have been modeled as monolithic due an easy deployment model
- Several services are combined into one massive single application
- This model leads to a poor use of resources when it comes to scaling out your application
- Greedy components steal resources from others that are collocated with them



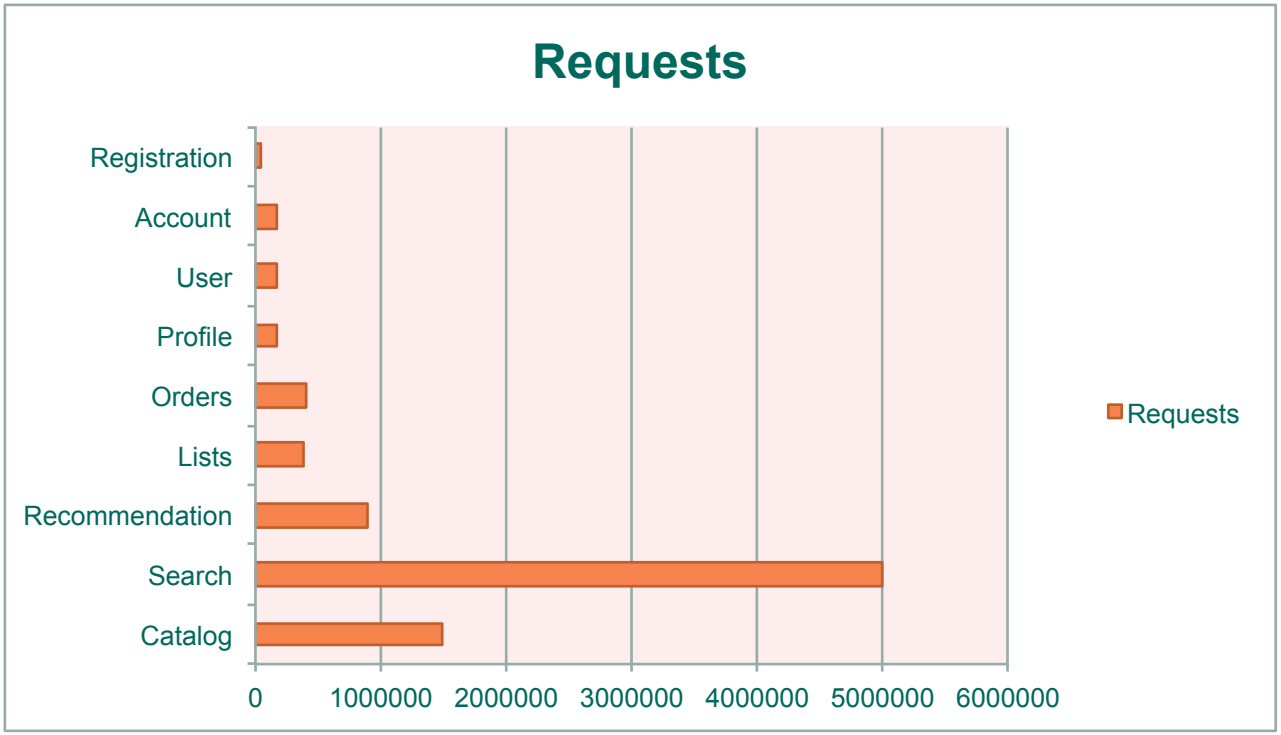
Pivotal

Scaling monolithic apps



Pivotal

How you should scale



The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

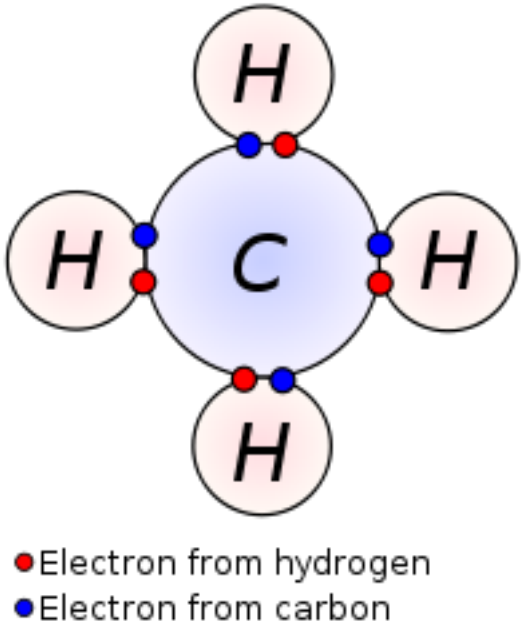
Resiliency

Traceability / Logging

Pivotal

Inter dependency

- Monolithic applications represent component dependency as strong bindings at the code level
- It makes harder to get a independent scalable model promoted by service oriented architectures



Pivotal

The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

Resiliency

Traceability / Logging

Pivotal

Platform specificity

- Hard dependencies on the runtime environment make applications not portable
- At code level with runtime dependencies
- At OS level (relying on cron as a scheduler)

```
import javax.servlet.http.HttpServlet;
```

```
import com.ibm.servlet.engine.webapp.*;
```

Pivotal

The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

Resiliency

Traceability / Logging

Pivotal

Location Specificity : Writing to disk

- Applications often need to write to disk, this includes form uploads with binary data, or content in most CMS systems
- Containers are short lived and not guaranteed to be executed on the same hardware every time they are needed to restart. Depending on a local file system is a big lock dependency some applications impose on the runtime
- This is one of the first issues to question or address when starting a conversation around new applications or selecting candidates to execute in PCF
- Some CMS vendors support usage of a service such as S3 to be the persistent mechanism of choice

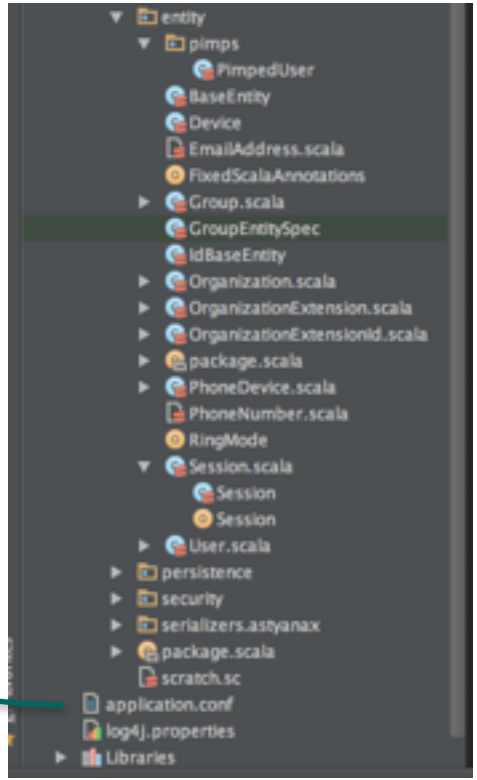
Pivotal

Location Specificity : service locations

```
Cache.hosts=10.68.27.41,10.68.27
.42

#naming does not help either
Cache.hosts=cacheserver1,
cacheserver2
```

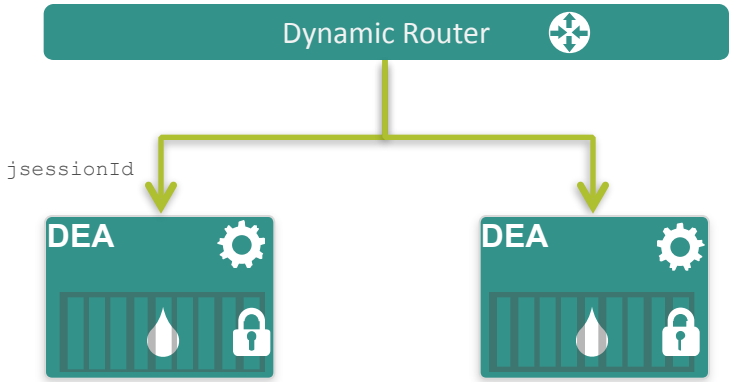
Properties file deployed
With application



Pivotal

Location Specificity : Sticky sessions

- Applications that rely on session stickiness are subject to lead to unexpected behavior in case of failure of the node that contains that data
- Most web applications that use sessions do not have a replication strategy in place
- Your application availability is subject to a single node (even if you LB the load over other nodes)
- Some app servers use P2P session replication which can be very costly on large deployments



Pivotal

The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

Resiliency

Traceability / Logging

Pivotal

Resilience

- No graceful shutdown in case of catastrophic events
- No recovery when the process executing the application dies
- No horizontal and automatic scale under heavy load
- No fallback mechanisms when dependent services are broken

Pivotal

The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

Resiliency

Traceability / Logging

Pivotal

Traceability / Logging

- Traditional apps usually write logs to files on disk
- When you have a cluster composed of several instances, it becomes really hard to trace log files spread across different locations

Pivotal

What about J2EE?

Pivotal

The J2EE story

- Java enterprise beans has always been seen as a mammoth.
- It inspired Rod Johnson to create Spring Framework, and the rest is history
- The cumbersome model, coupled to fat application servers, and excessive plumbing code, led the market to embrace spring as the top application framework in the world
- EJB 3 tried to fight back by adopting a lot of spring framework's core concepts. Unfortunately by the time it was released, AWS was already a reality.

Can I run my JEE app on PCF?

- JEE is a vast spec, people usually take for granted that JEE = EJB. EJBs are not a good match for running inside cloud foundry we will cover that later
- For the majority of the spec, we actually support it inside tomcat. The main concerns are mostly around the application architecture and not the specs it uses
- There are just a few specs are really not supported, be careful when bumping on those

Pivotal

JEE 7 and CloudFoundry (Most common JSRs)

JSR	Description	Supported	Notes
JSR 338	Java Persistence API 2.1	yes	Spring data JPA has better support than this JSR
JSR 340	Java Servlets 3.1	yes	Tomcat 8 support servlets 3.1
JSR 339	JAX-RS 2.0 Restfull web services	yes	Jersey and resteasy can be used instead of SpringMVC
JSR 344	JSF 2.2 Java server faces	yes	Supported on tomcat containers
JSR 349	Bean Validation 1.1	yes	Spring has support for bean validation
JSR 245	Java server pages 2.3	yes	Supported on tomcat containers
JSR 919	Java mail 1.5	yes	Spring has extensive support for java mail senders
JSR 343	JMS 2.0	partial	No support for MDBs, but JMS client is fully supported
JSR 345	Enterprise Java Beans 2.0	No	
JSR 322	Java Connector Architecture 1.7	No	
JNDI	Java Naming Directory Interface	No	Actually part of JSE, but its used by most containers

Pivotal

JNDI

- JNDI is not JEE it's actually part of JSE, but it's quite often used by containers to store objects such as datasources or remote clients to EJBs
- JNDI was never truly portable, across different vendors naming is different for same type of objects, stress that with your customer
- Tomcat offers a read-only JNDI, so although possible to run applications that rely on datasource JNDI objects, its really not recommended, sometimes its not that simple to port them.

EJB: Stateless Session Beans

- SLSB could potentially be executed on a supported container on PCF, but if the application only use them as a service bridge between the local web application (WAR) and business code, spring is a much lighter weight approach for that.
- The problem on running SLSBs on PCF its really the fact that EJBs are located using JNDI on a remote port (1099 for example) and RPC are made towards another port (4445 for example). Warden only exposes one port for your application
- So although an application using a web front end and a SLSB packaged on a single WAR could potentially be executed on PCF, there could never be an external route to the EJBs

Pivotal

EJB: Stateful Session Beans

- SFSBs on the other hand are not really supported. The main issue is how replication works, and this is usually locked by vendor, and it requires multiple ports opened, which Warden would not allow. It could also be implemented using multicast (Jboss) and most cloud providers ban multicasting from their networks
- SFSB makes code testing really hard. They allow state on the class level, and wrap calls on proxies to prevent racing conditions. This also makes the code practically impossible to be ported

EJB: Message Driven Beans

- MDBs could be supported if running a JEE container inside PCF. MDBs are just message listeners to JMS that runs inside a transactional container.
- Spring Messaging offers a very similar model (including transaction support) without the coupling of the container.
- If message driven architecture is a requirement for the application, show how simple it is to implement a message driven pojo with spring as opposed to a message driven bean

JTA and distributed trasactions

- If an application relies on distributed transactions across several EJB containers, that is a good indication that is really not a good fit for cloud applications.
- Distributed transactions should be avoided, they bring a lot of complexity to the architecture, with very rarely any benefit
- Spring supports 2PC using an external transaction provider (atomikos) but when talking about microservices, transactions should be a good boundary to start a conversation around the responsibility of the service

3rd Platform Strategies

Pivotal

Better 3rd Platform Strategies

Local Disk Storage

Embedded Services

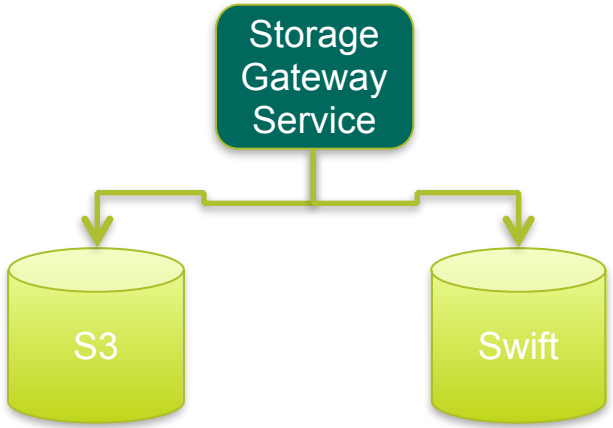
Session Replication

Logging

Pivotal

Local disk storage

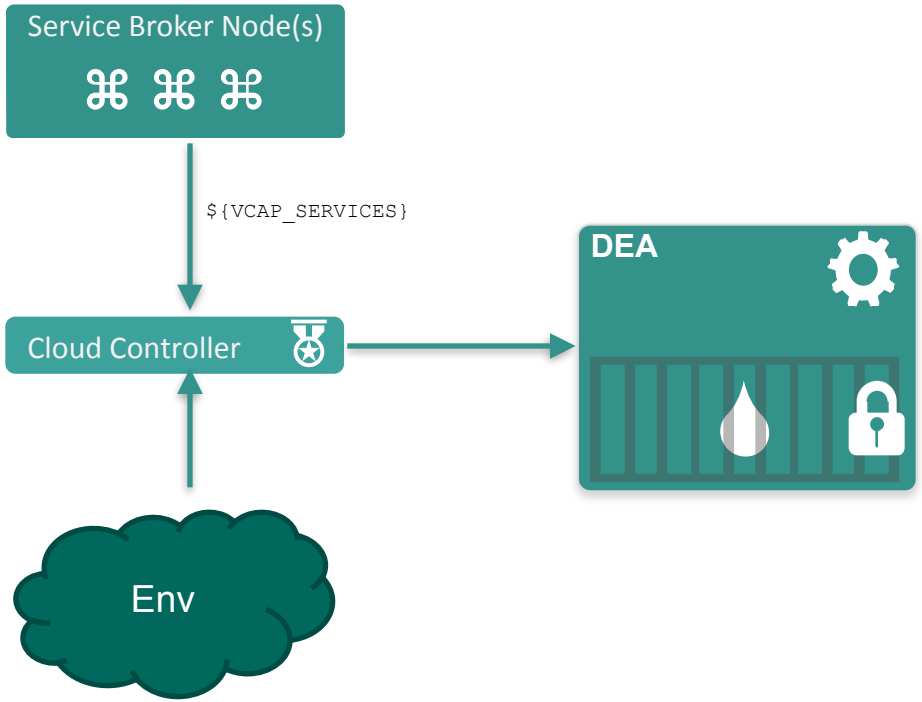
- Use an storage gateway service instead of relying on plain old file system access
- This abstracts the need of a local file system while giving your application a flexible mechanism to rely depending on it's runtime (local on your dev, S3 or swift on prod)



Pivotal

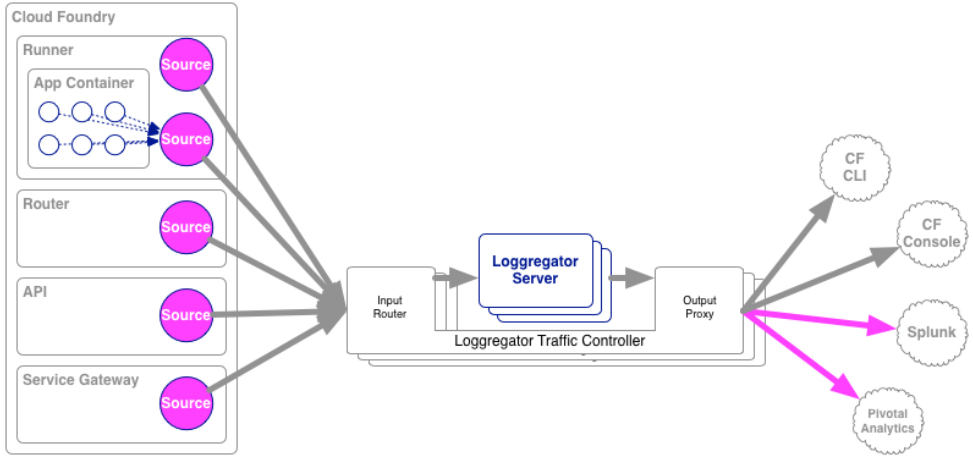
Embedded services

- Let the runtime inject any service reference through environment variables
- Reduces **location specificity** of the application
- Allows **disposability** of containers
- Runtime should **inject** any **variable** into the container



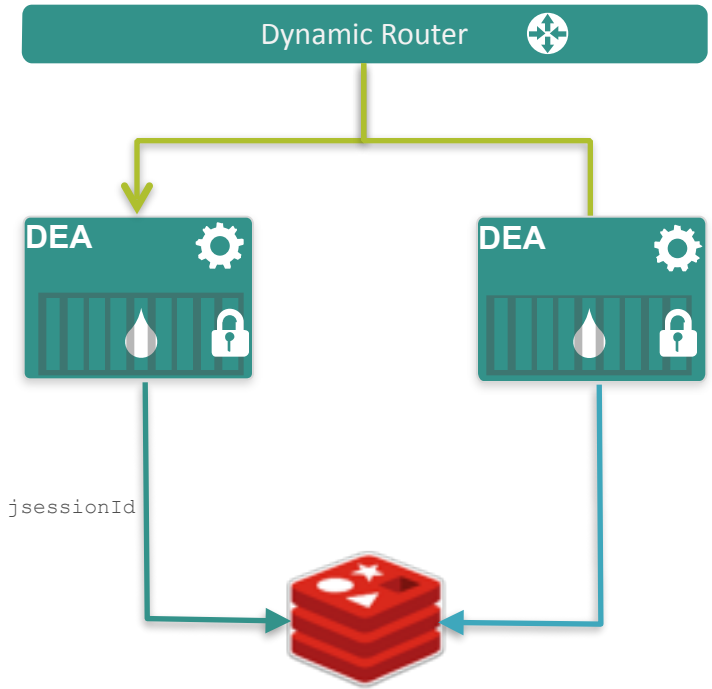
Logging

- Do not write to log files
- Output logs to console and use a syslog drain to collect all logs



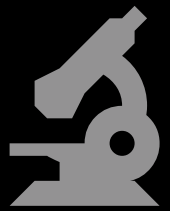
Session replication

- The java buildpack supports redis session replication
- Sessions are now stored on redis, and in case of a node failure, the client state can still be found on a second node



Pivotal

Microservices Overview



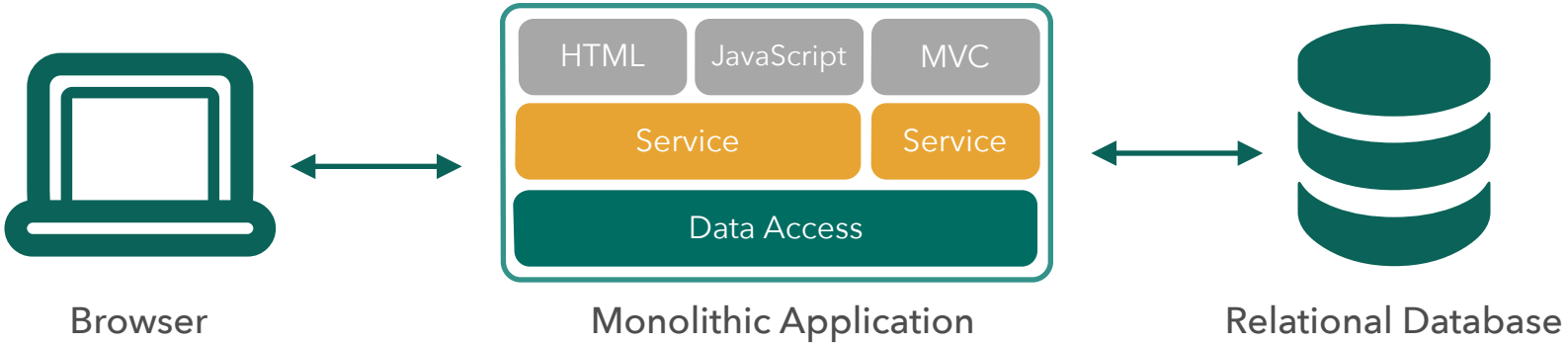
Pivotal

Simple vs. Easy

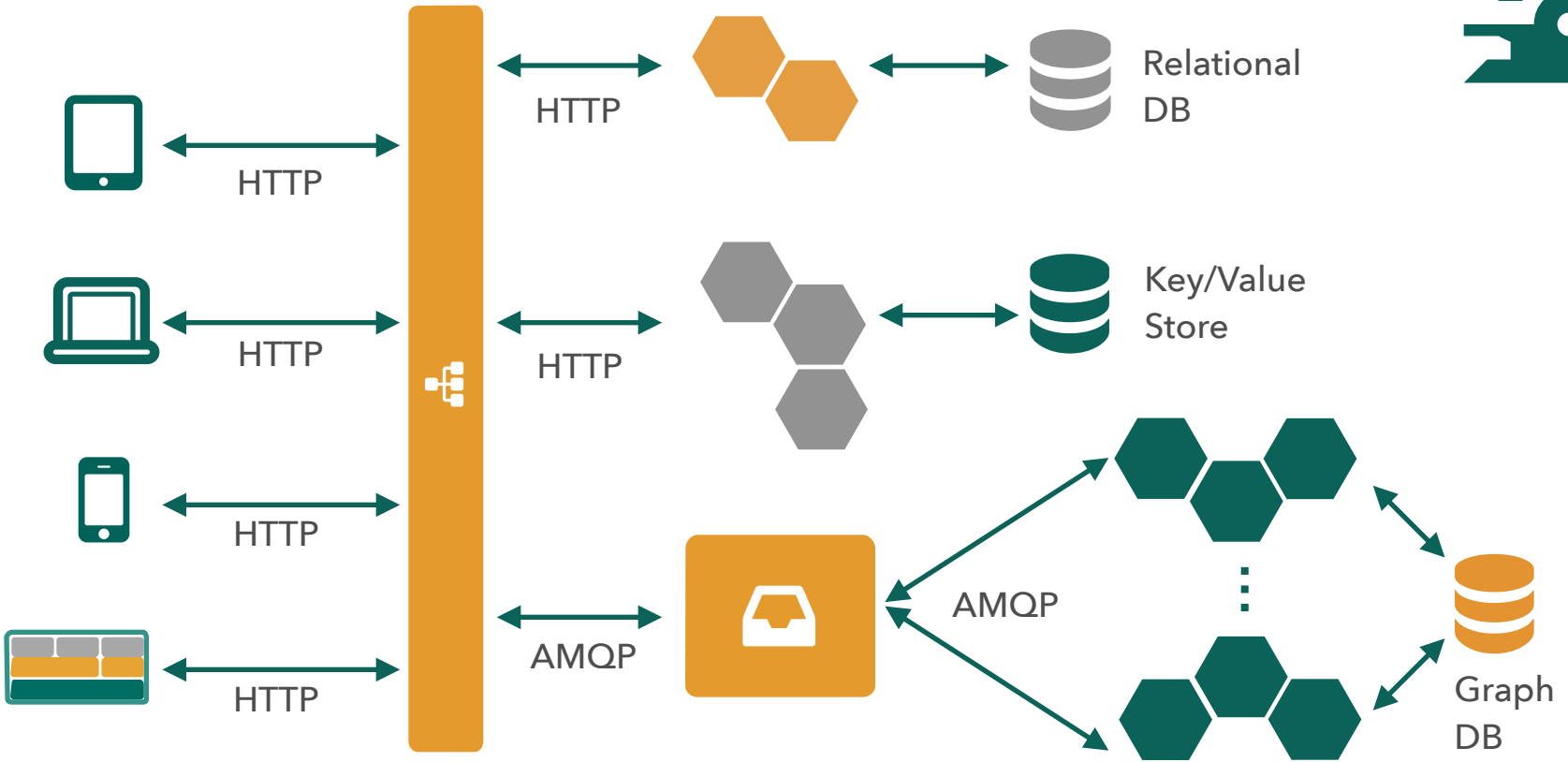
- **Simple**
  - *sim- plex*
  - one fold/braid
  - vs complex
- **Easy**
  - *ease < aise < adjacens*
  - lie near
  - vs hard

Pivotal

Monolithic Architecture



Microservice Architecture



Pivotal

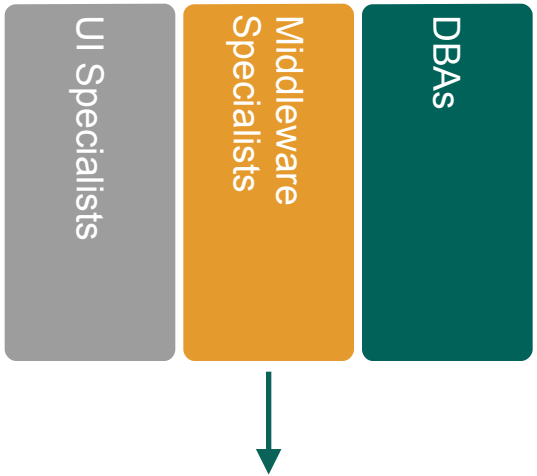
Microservice Architectures

- Simple / Hard
- Modularity Based on Component Services
- Change Cycles Decoupled / Enable Frequent Deploys
- Efficient Scaling
- Individual Components Less Intimidating to New Developers
- Enables Scaling of Development
- Eliminates Long-Term Commitment to Technical Stack

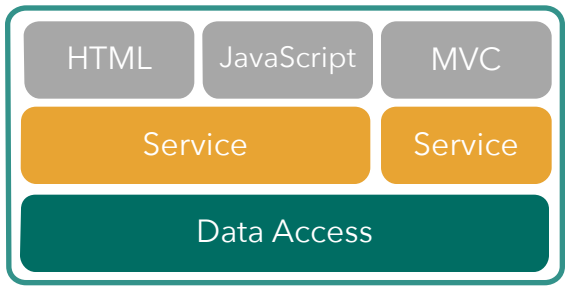
Pivotal

Organize Around Business Capabilities

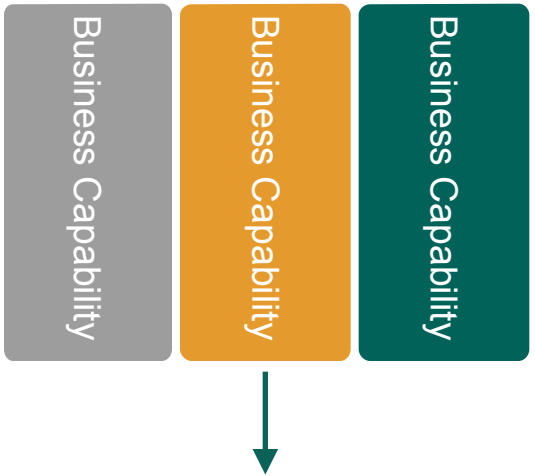
Siloed Functional Teams



Siloed Application Architectures



Cross-functional Teams



Microservice Architectures

http://martinfowler.com/articles/microservices.html#OrganizedAroundBusinessCapabilities

Pivotal

UNIX Pipes and Filters

```
cut -d" " -f1 < access.log | sort | uniq -c | sort -rn | less
```

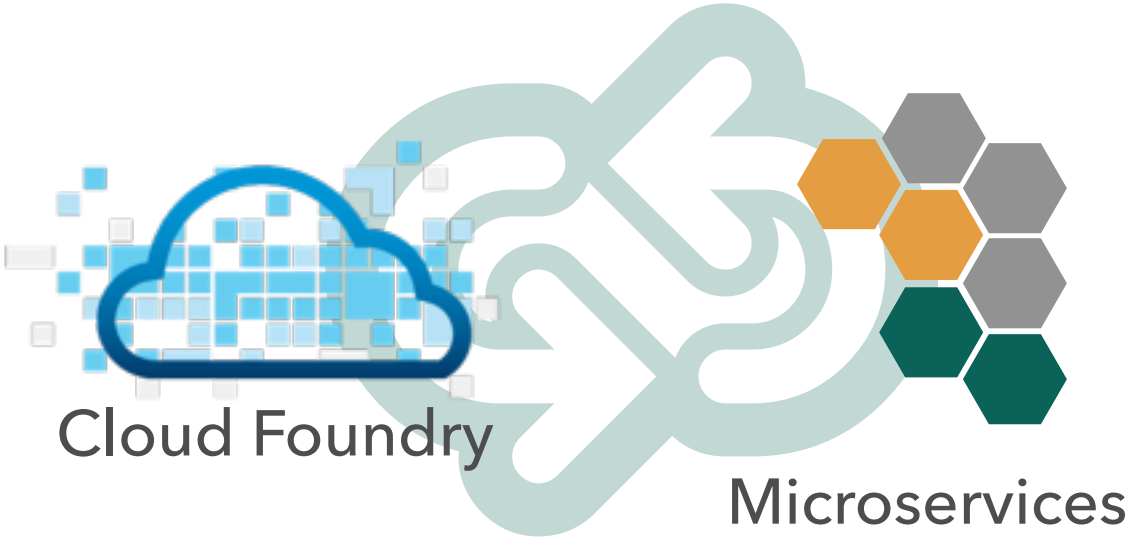
Pivotal

Choreography over Orchestration

http://martinfowler.com/articles/microservices.html#SmartEndpointsAndDumbPipes

Pivotal

A Mutualistic Symbiotic Relationship…



http://en.wikipedia.org/wiki/Symbiosis#Mutualism

Pivotal

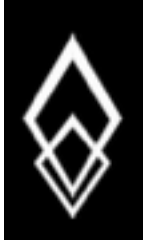
The Twelve-Factor App

Overview

http://12factor.net

Pivotal

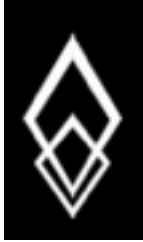
Twelve Factor + Cloud Foundry



- One codebase tracked in revision control, many deploys
  - Multiple = Distributed System
  - Consistent with CF application unit
- Explicitly declare and isolate dependencies
  - CF emphasis on deployable units (e.g. Java WAR)
  - CF Buildpacks provide runtime dependencies

Pivotal

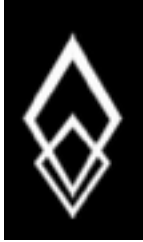
Twelve Factor + Cloud Foundry



- Store config in the environment
  - Nicely facilitated via CF

Pivotal

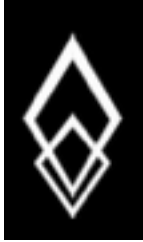
Twelve Factor + Cloud Foundry



- Store config in the environment
  - Nicely facilitated via CF
- Treat backing services as attached resources
  - cf create-service / cf bind-service

Pivotal

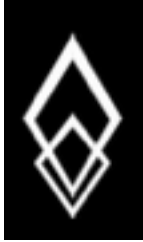
Twelve Factor + Cloud Foundry



- Store config in the environment
  - Nicely facilitated via CF
- Treat backing services as attached resources
  - cf create-service / cf bind-service
- Strictly separate build and run stages
  - CF Buildpacks + immutable Warden containers

Pivotal

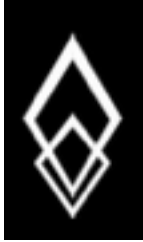
Twelve Factor + Cloud Foundry



- Execute the app as one or more stateless processes
  - CF Warden containers - no app server clustering, no shared FS.
  - Challenge for the monolith!

Pivotal

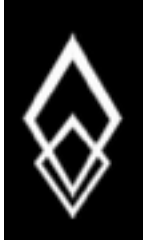
Twelve Factor + Cloud Foundry



- Execute the app as one or more stateless processes
  - CF Warden containers - no clustered memory, no shared FS.
  - Challenge for the monolith!
- Export services via port binding
  - CF provides HTTP/HTTPS today, more future (TCP?)

Pivotal

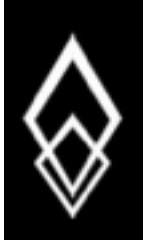
Twelve Factor + Cloud Foundry



- Execute the app as one or more stateless processes
  - CF Warden containers - no clustered memory, no shared FS.
  - Challenge for the monolith!
- Export services via port binding
  - CF provides HTTP/HTTPS today, more future (TCP?)
- Scale out via the process model
  - cf scale app -i 1000

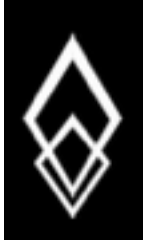
Pivotal

Twelve Factor + Cloud Foundry



- Maximize robustness with fast startup and graceful shutdown
  - CF scales quickly, but can only move as fast as your app can bootstrap (challenge for the monolith!

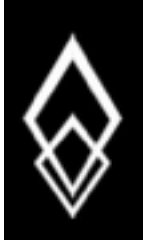
Twelve Factor + Cloud Foundry



- Maximize robustness with fast startup and graceful shutdown
  - CF scales quickly, but can only move as fast as your app can bootstrap (challenge for the monolith!)
- Keep development, staging, and production as similar as possible
  - CF is CF! Spaces provide separation of concerns without technical differences.

Pivotal

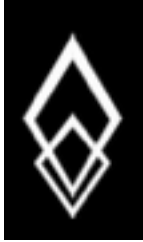
Twelve Factor + Cloud Foundry



- Treat logs as event streams
  - CF Loggregator!
- Run admin/management tasks as one-off processes
  - Still a challenge to be addressed…

Pivotal

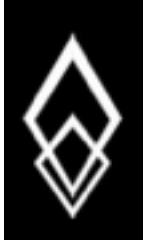
Twelve Factor + Microservices



- Fully compatible architectural style
- Frameworks tend to optimize around same ideas
- Examples:
  - Spring Boot + Cloud
    - http://projects.spring.io/spring-boot
    - http://projects.spring.io/spring-cloud
  - Dropwizard (https://dropwizard.github.io/dropwizard)

Pivotal

Twelve Factor + Microservices



- Fully compatible architectural style
- Frameworks tend to optimize around same ideas
- Examples:
  - Spring Boot + Cloud
    - http://projects.spring.io/spring-boot
    - http://projects.spring.io/spring-cloud
  - Dropwizard (https://dropwizard.github.io/dropwizard)

Pivotal

3rd Platform Realities

Pivotal

Paying for your lunch…

- Significant Operations Overhead
- Substantial DevOps Skills Required
- Implicit Interfaces
- Duplication of Effort
- Distributed System Complexity
- Asynchronicity is Difficult!
- Testability Challenges

Pivotal

Paying for your lunch…

- **Significant Operations Overhead**
- **Substantial DevOps Skills Required**
- Implicit Interfaces
- Duplication of Effort
- **Distributed System Complexity**
- Asynchronicity is Difficult!
- **Testability Challenges**

Pivotal

Significant Operations Overhead

- Mitigate polyglot language/environment provisioning complexity via CF Buildpacks
- Mitigate failover and resilience concerns via CF Scale, CF Health Monitor, and future CF App AZ's (http://blog.gopivotal.com/cloud-foundry-pivotal/products/the-four-levels-of-ha-in-pivotal-cf)

Pivotal

Significant Operations Overhead

- Mitigate polyglot language/environment provisioning complexity via CF Buildpacks
- Mitigate failover and resilience concerns via CF Scale, CF Health Monitor, and future CF App AZ's (http://blog.gopivotal.com/cloud-foundry-pivotal/products/the-four-levels-of-ha-in-pivotal-cf)
- Mitigate routing/load balancing and plumbing concerns via CF Router and CF Services
- High quality monitoring = CF BP agent-based tooling, future CF metric streams
- High quality operations infrastructure = CF BOSH!

Pivotal

Significant Operations Overhead

- Mitigate polyglot language/environment provisioning complexity via CF Buildpacks
- Mitigate failover and resilience concerns via CF Scale, CF Health Monitor, and future CF App AZ's (http://blog.gopivotal.com/cloud-foundry-pivotal/products/the-four-levels-of-ha-in-pivotal-cf)
- Mitigate routing/load balancing and plumbing concerns via CF Router and CF Services
- High quality monitoring = CF BP agent-based tooling, future CF metric streams
- High quality operations infrastructure = CF BOSH!
- Robust release/deployment automation = CF API, scriptable CF CLI, Maven/Gradle Plugins, Strong Cloudbees/Jenkins partnerships

Pivotal

Substantial DevOps Skills Required

- This is a **Good Thing™** in any architecture!
- CF keeps your microservices up and available (and your monoliths too!)
- CF = development and production parity!
- Polyglot persistence without all the fuss: CF BOSH and Service Brokers

Pivotal

Substantial DevOps Skills Required

- This is a **Good Thing™** in any architecture!
- CF keeps your microservices up and available (and your monoliths too!)
- CF = development and production parity!
- Polyglot persistence without all the fuss: CF BOSH and Service Brokers

Pivotal

Distributed System Complexity

- Agreed: Microservices imply distributed systems.
- All of the CF platform features we've discussed help to mitigate these concerns:
  - latent/unreliable networks
  - fault tolerance
  - load variability

Pivotal

Testability Challenges

- With CF, it is ***NOT*** difficult to recreate environments in a consistent way for either manual or automated testing!
- Idiomatic Microservices involves placing less emphasis on testing and more on monitoring
  - Not sure where this idea comes from…
  - CF is an enabler of both!

Pivotal

Testability Challenges

- With CF, it is ***NOT*** difficult to recreate environments in a consistent way for either manual or automated testing!
- Idiomatic Microservices involves placing less emphasis on testing and more on monitoring
  - Not sure where this idea comes from…
  - CF is an enabler of both!

Pivotal

A NEW PLATFORM FOR A NEW ERA