Introduction To Microservices

Building Cloud Native Applications

Pivotal

Limitations of traditional apps

Pivotal

The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

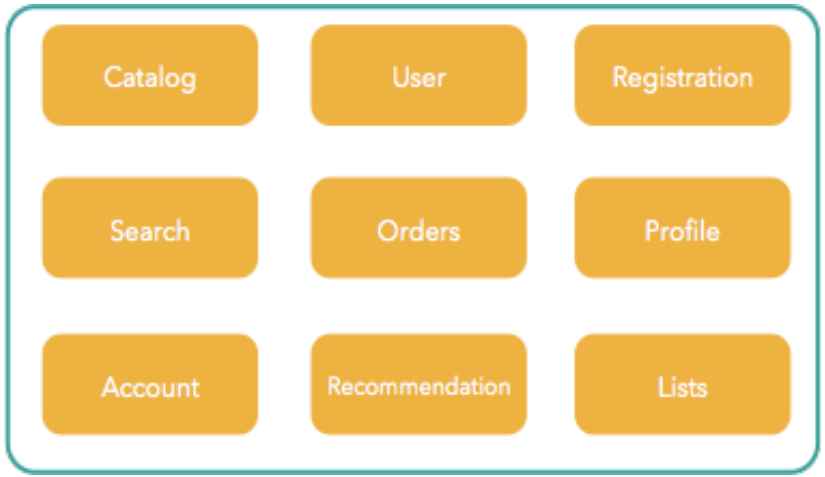
Resiliency

Traceability / Logging

Pivotal

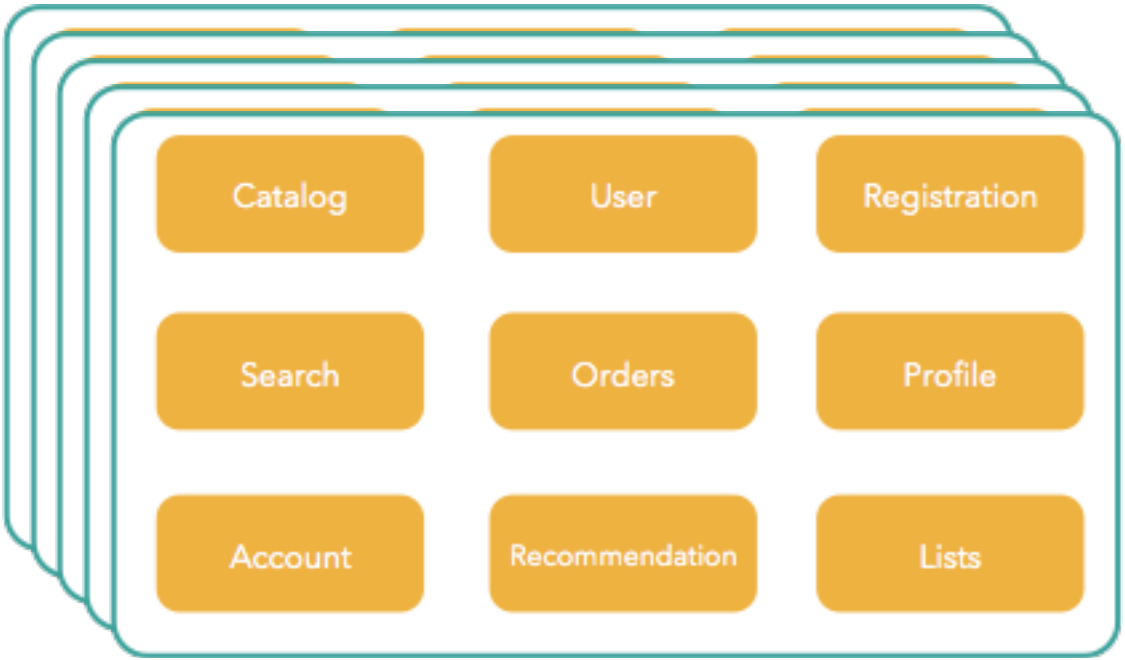
Scalability

- Traditional applications are deployed as a single artifact hence the monolithic term
- Several services are combined into a single application (usually large)
- This model makes poor use of resources especially when it comes to scaling
- One or several components may hog all of the resources



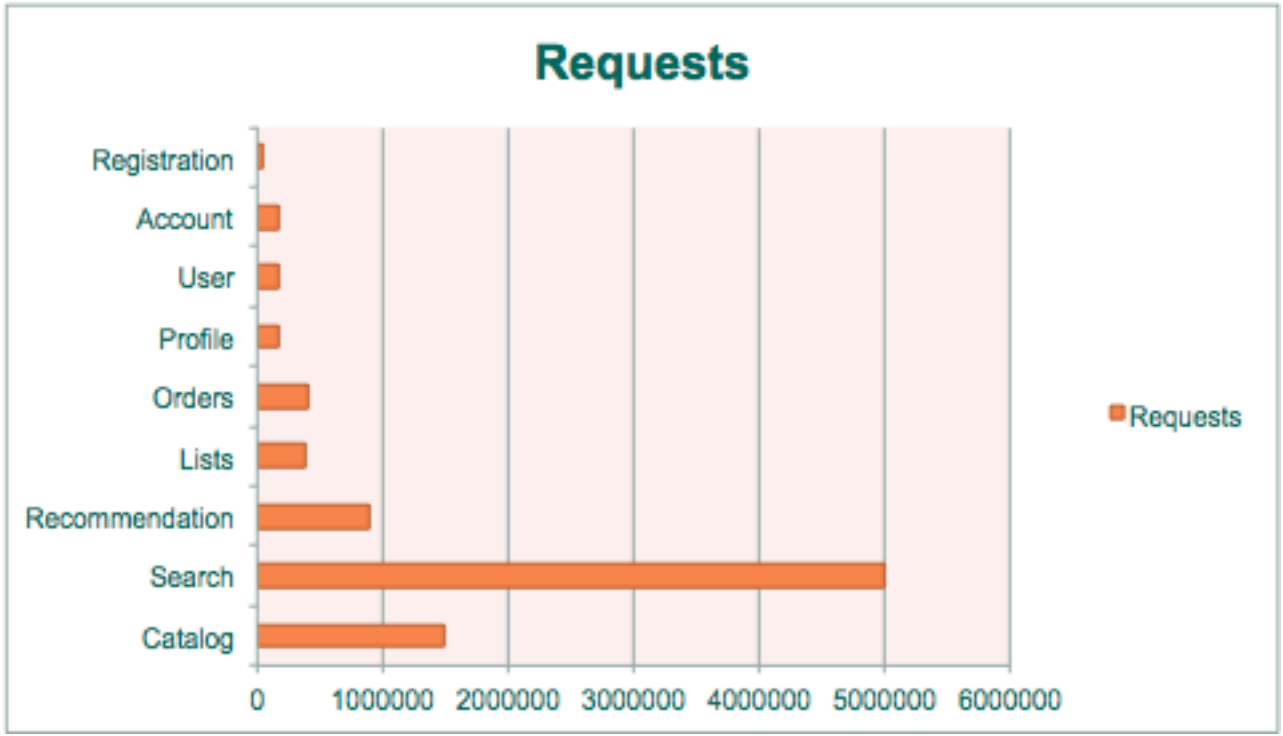
Pivotal

Scaling monolithic apps

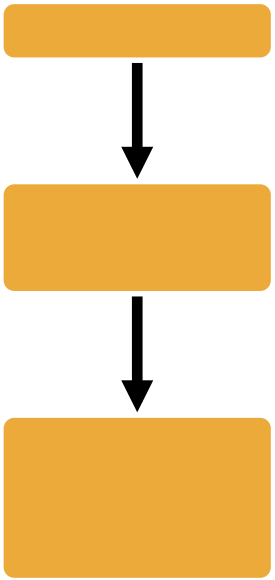


Pivotal

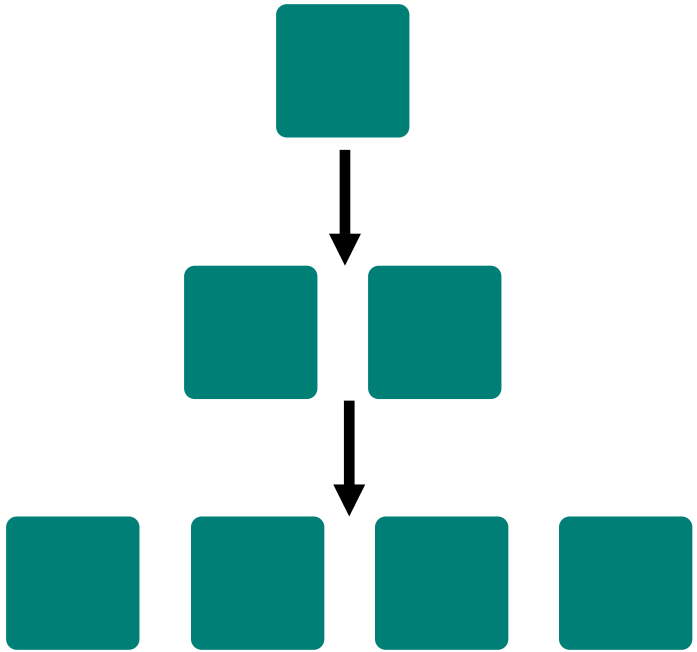
How you should scale



How you should scale



Slow/Expensive



Fast/Cheap

Pivotal

The limitations of traditional apps

Scalability

Inter-Dependency

Platform Specificity

Location Specificity

Resiliency

Traceability / Logging

Pivotal

Inter dependency

- Monolithic applications: component dependency as strong bindings at the code level
- Hard / impossible to get independent scalable model
- External dependencies are often also strongly bound

Pivotal

Inter dependency

- Change to a single component impacts many others
- Integration testing is much harder and often breaks
- Slows down the development process as a whole
- Imposes big development teams: can't really have separate teams owning different pieces

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 (i.e. relying on cron as a scheduler)
- At a business level, introduces vendor lock-in

```
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 need to be restarted. 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 Pivotal Cloud Foundry
- Some CMS vendors support usage of a service such as S3 to be the persistent mechanism of choice

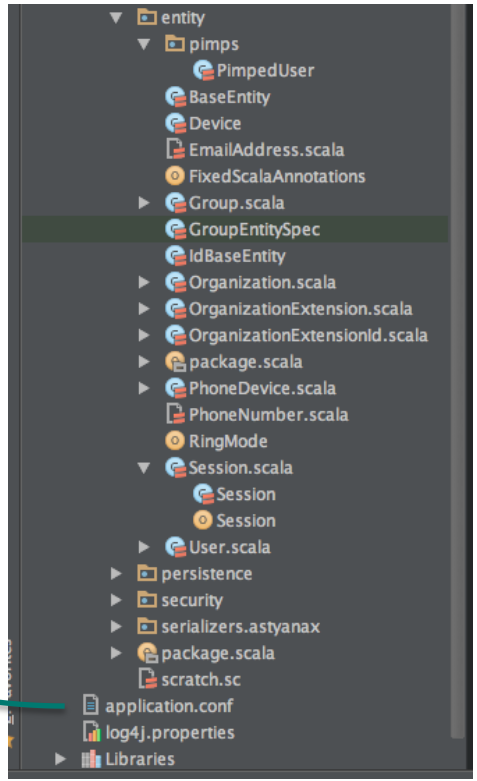
Pivotal

Location Specificity : service locations

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

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

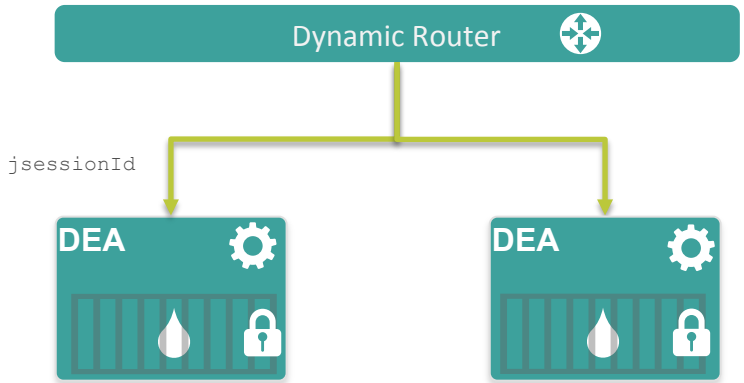
Properties file deployed
With application



Pivotal

Location Specificity : Sticky sessions

- Applications that rely on session stickiness may run into 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

12 factor apps

Pivotal

12 Factor Apps

I. Codebase

II. Dependencies

III. Configuration

IV. Backing services

V. Build, release, run

VI. Process

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/Prod parity

XI. Logs

XII. Admin Process

Pivotal

12 Factor Apps

I. Codebase

- App = Single codebase.
- Multiple codebases = distributed system
- Each codebase tracked in version control
- Single codebase, multiple deployments (dev, test, prod, vSphere, OpenStack, linux, Windows, etc)

Pivotal

12 Factor Apps

II. Dependencies

- Easily and individually packaged - Jar (Java), gems (Ruby), CPAN (Perl), etc
- Declared on a manifest for ultimate separation of concerns and abstraction. (i.e. Maven .pom file)
- No reliance on system specific tools (i.e. Linux tool not runnable on Windows)

Pivotal

12 Factor Apps

III. Configuration

- Config varies across deployments - source code does not.
- Any external dependency config and credential should be part of environment, not code.
- Stored as environment variables.
- Internal wiring which doesn't change between deployments is code(i.e. Spring config)

Pivotal

12 Factor Apps

IV. Backing Services

- Consumed by application as normal operations: DBs, messaging queues, SMTP servers, etc.
- May be locally or 3rd party managed
- Connected via URL / config, independent from source code.
- Should be swappable (i.e. change from local DB to 3rd party provided DB)

Pivotal

12 Factor Apps

V. Build, Release, Run

Lifecycle Stages:

- Strictly separated build and run stages.
- Build: convert codebase into build, including dependencies
- Release: build + specific config (i.e. dev release). Ready to run
- Run: Runs apps processes into the target execution environment, using the right release.

Pivotal

12 Factor Apps

VI. Process

- App is built of one of more discrete running processes
- Stateless, should rely on a shared nothing architecture
- Data needed to be shared between processes should be persisted:
  - Memory / Local storage considered volatile
  - Processes should communicate through messaging or shared external storage / Db.

Pivotal

12 Factor Apps

VII. Port Binding

- App should not need a “container". They are completely self-contained
- Web App exports HTTP as a service by binding to a port and listening requests
- One app can become the backing service to other app as providing the URL as resource to the config for consuming app.

Pivotal

12 Factor Apps

VIII. Concurrency

- Scale-out via process model
- Processes are first class citizens
- Adding more concurrency is simple and reliable.
- Must be horizontally scalable
- Should rely on platform or OS process manager to manage output streams, respond to crashes and restart/shutdowns.

Pivotal

12 Factor Apps

IX - Disposability

- Processes and instances should be disposable
- Only achieved as being stateless, location agnostic.
- Quick to start and stop.
- Exit gracefully, finish current requests or idempotent / reentrant
    - Return current job to working queue
    - Enhanced scalability and fault-tolerance
    - Designed crash-only software

Pivotal

12 Factor Apps

X - Dev/Prod Parity

- Designed for Continuous Deployment, minimizing gaps.
- Time gap: build/release/deploy cycles short
- Personnel gap: devs are closely involved on deployment and watching production.
- Tools gap: keep dev and prod as similar as possible.
- Keep save services - even though code might make it irrelevant.

Pivotal

12 Factor Apps

XI - Logs

- Logs: streams of time-ordered, aggregated events
- Apps should not be concerned about storing or routing logging: like write to sysout.
- Platform should take care of logging depending on the environment. Diff environments have diff needs.
- External log management specialized tools should handle indexing and analysis (i.e. Splunk)

Pivotal

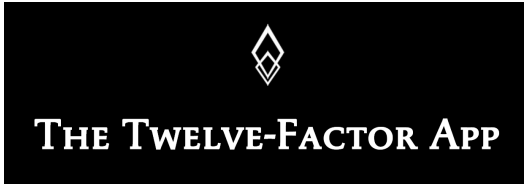
12 Factor Apps

XII - Admin Process

- Admin / Management processes run as One-Off processes
- Db provisioning, maintenance. (e.g. Service Broker registration in CF)
- Use same environment / platform as the applications.
- Same dependency isolation and other characteristics apply.

Pivotal

Twelve Factors



- One Codebase/Many Deploys
- Explicit Isolated Dependencies
- Config via Environment
- Attached Backing Services
- Separate Build/Release/Run
- Stateless Processes
- Export Services via Port
- Bindings
- Scale Out via Processes
- Disposable Instances
- Dev/Prod Parity
- Logs == Event Streams
- Admin Tasks == Processes

http://12factor.net

Pivotal

Cloud Native Design

- Evaluate components for 12-factor compliance
- Evaluate components and architecture for Cloud Native Design
- Standalone services
- Common services

Pivotal

Evaluate for the Cloud

Cloud Native

- Micro-service architecture and principles
- API first design

Cloud Resilient

- Design for failure
- Apps are unaffected by dependent service failure
- Proactive testing for failure
- Metrics and monitoring baked in
- Cloud agnostic runtime implementation

Cloud Friendly

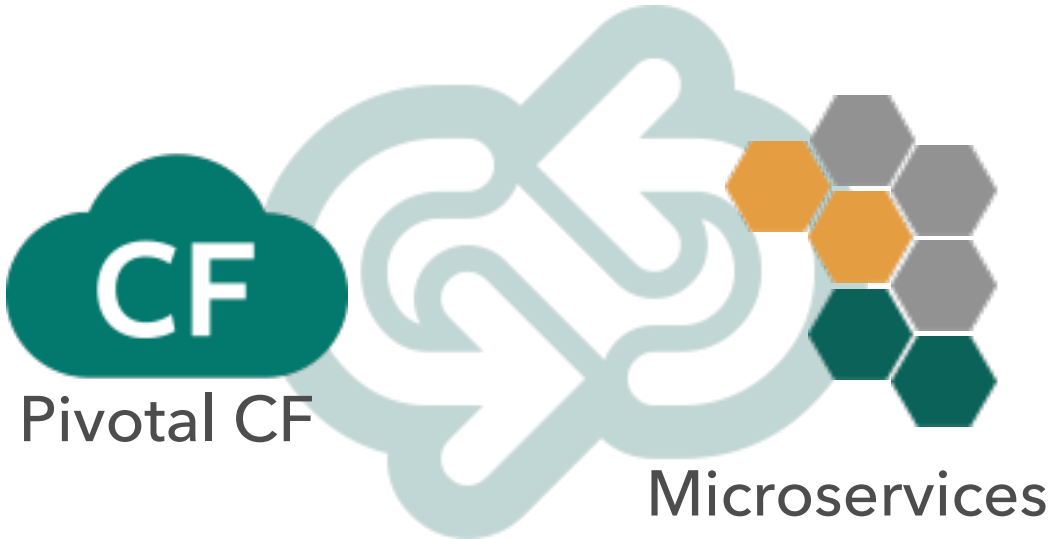
- Twelve factor applications
- Horizontally scalable
- Leverage platform for HA

Cloud Ready

- No file-system requirements or uses S3 API
- Self-contained application
- Platform managed ports and addressing
- Consume off platform services using platform semantics

Pivotal

Made for each other



Continuous Delivery

Software Factories

Feedback → Rapid Iteration

Horizontal Scale

Diversity of Clients

Physical/Virtual

Pivotal CF

Infrastructure

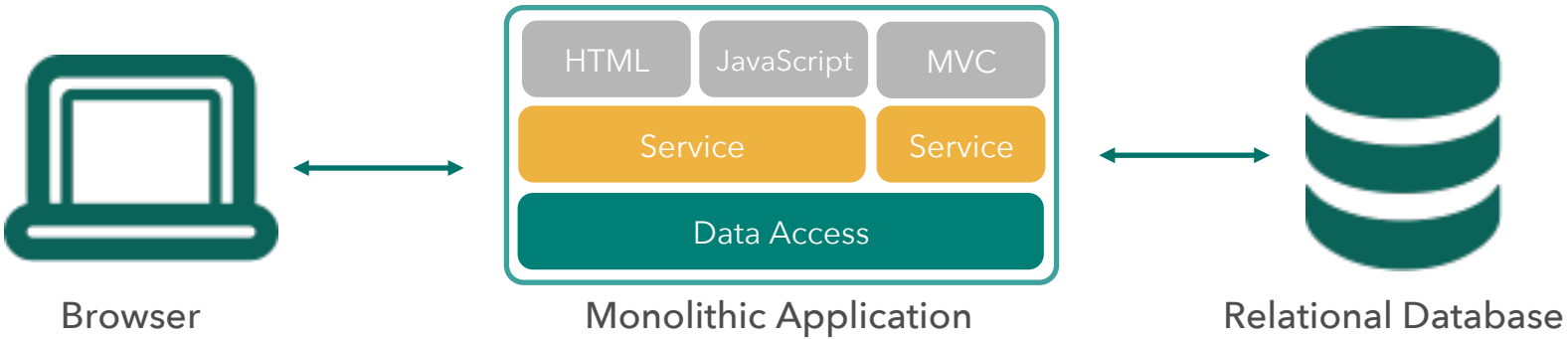
Applications

Monoliths

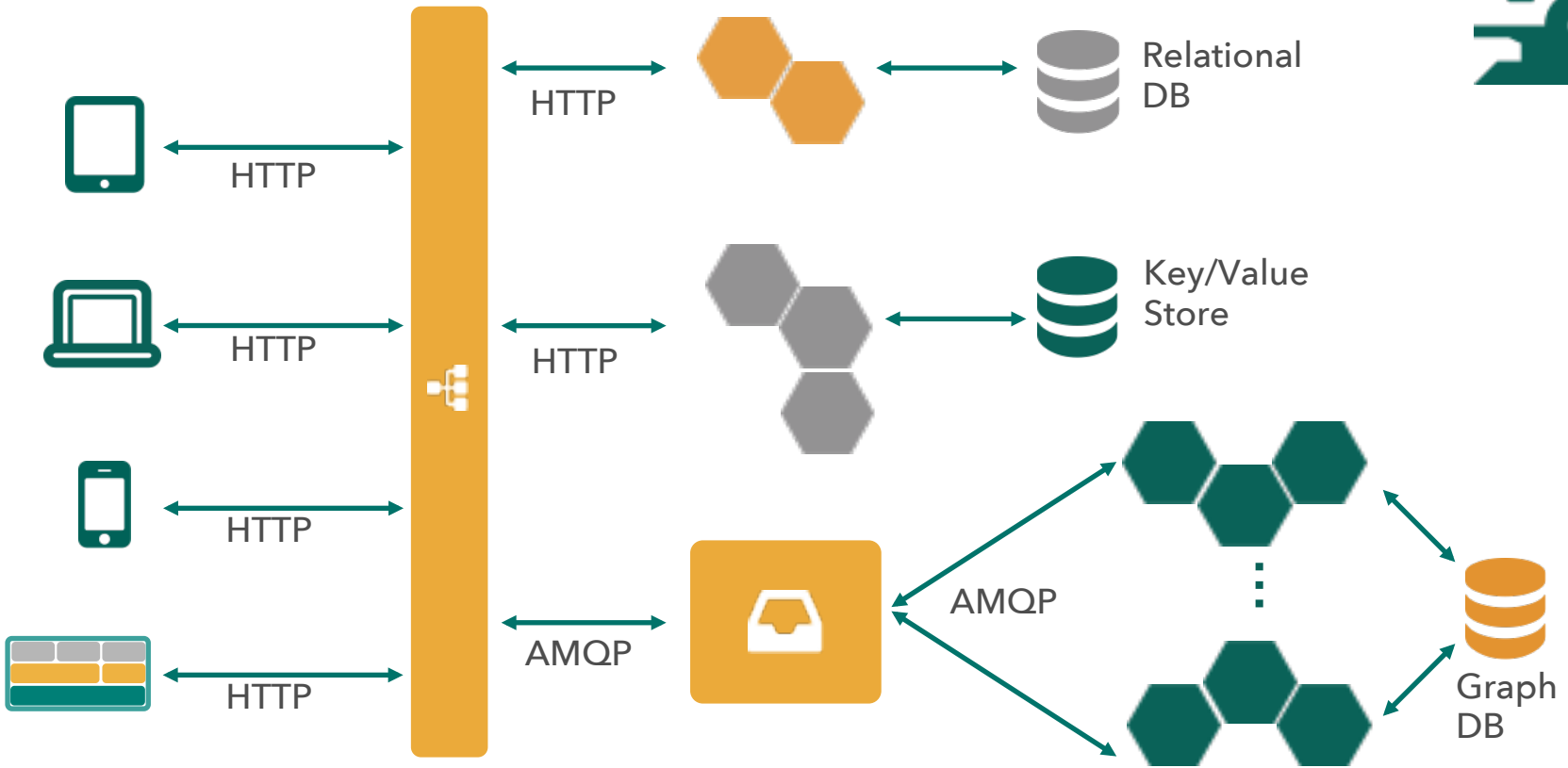
Microservices

Pivotal

Monolithic Architecture



Microservice Architecture

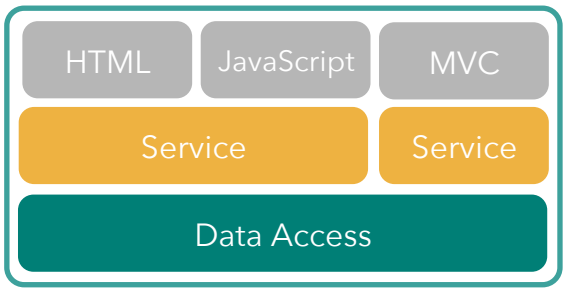


Pivotal

Organize Around Business Capabilities

Siloed Functional Teams

Siloed Application Architectures



Cross-functional Teams

Microservice Architectures

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

Pivotal

High Scalability

Building bigger, faster, more reliable websites.

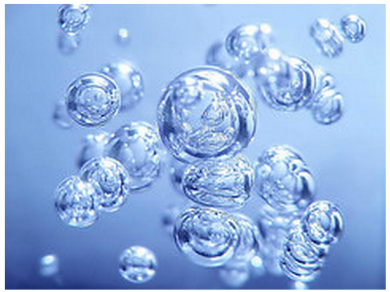
Home Real Life Architectures Strategies All Posts Advertising Book Store Start Here Contact
All Time Favorites RSS Twitter Facebook G+

« Paper: Scalable Atomic Visibility with RAMP Transactions - Scale Linearly to 100 Servers | Main | Google Finds: Centralized Control, Distributed Data Architectures Work Better than Fully Decentralized Architectures »

Microservices - Not A Free Lunch!

TUESDAY, APRIL 8, 2014 AT 8:54AM

This is a guest post by Benjamin Wootton, CTO of Contino, a London based consultancy specialising in applying DevOps and Continuous Delivery to software delivery projects.



Microservices are a style of software architecture that involves delivering systems as a set of very small, granular, independent collaborating services.

http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html

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

You must handle this to use Microservices…

- RAPID PROVISIONING
- BASIC MONITORING
- RAPID APPLICATION DEPLOYMENT
- DEVOPS CULTURE

https://www.flickr.com/photos/gusset/3723961589

http://martinfowler.com/bliki/MicroservicePrerequisites.html

Pivotal

It takes a platform...

Pivotal CF

Spring Cloud

Pivotal

Platform Features

- Environment Provisioning
- On-Demand/Automatic Scaling
- Failover/Resilience
- Routing/Load Balancing
- Data Service Operations
- Monitoring

- Distributed/Versioned Config
- Service Registration/Discovery
- Routing/Load Balancing
- Service Integration
- Fault Tolerance
- Asynchronous Messaging

Pivotal

Refactoring strategies

Pivotal

Some refactoring strategies

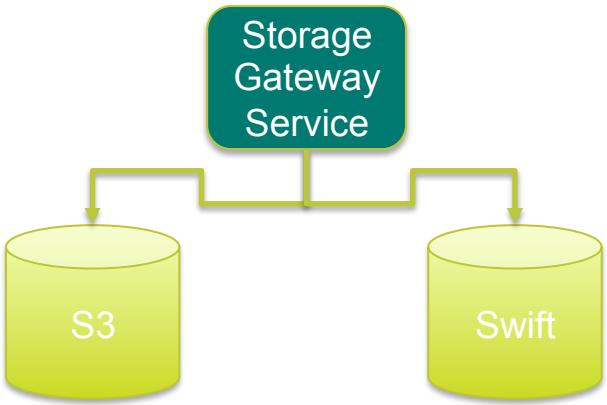
Local Disk Storage

Embedded Services

Session Replication

Logging

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



Pivotal

Some refactoring strategies

Local Disk Storage

Embedded Services

Session Replication

Logging

- 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

Pivotal

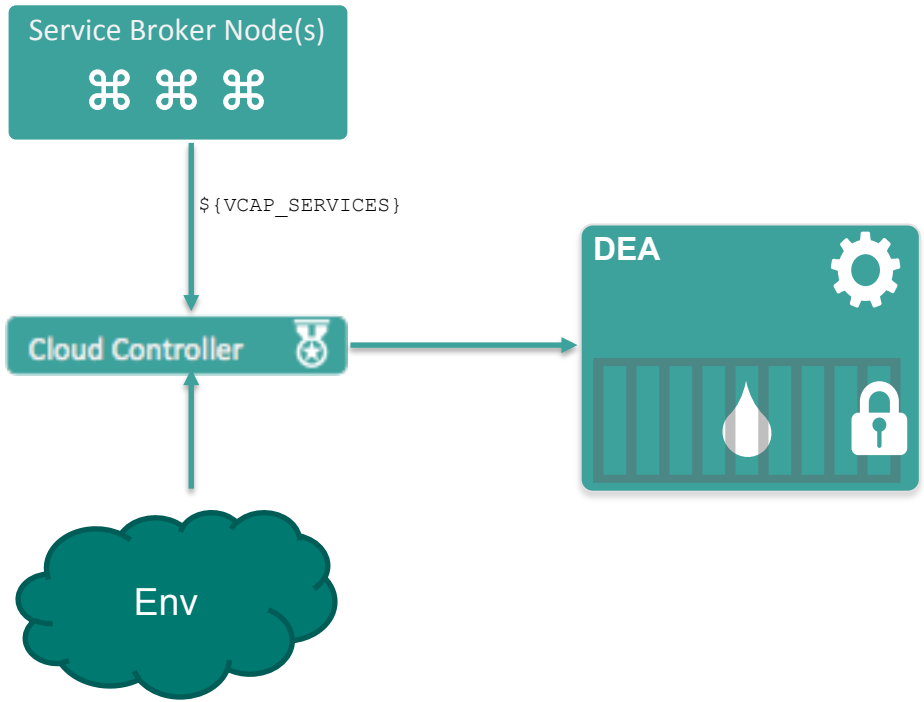
Some refactoring strategies

Local Disk Storage

Embedded Services

Session Replication

Logging



Pivotal

Some refactoring strategies

Local Disk Storage

Embedded Services

Session Replication

Logging

- When necessary, leverage an external service such as Redis or GemFire.
- Pivotal CF already provides session stickness

Pivotal

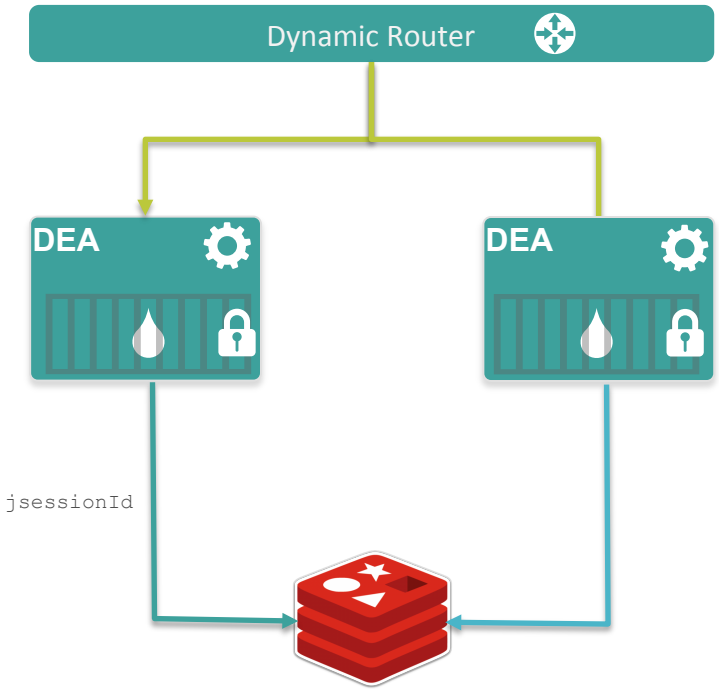
Some refactoring strategies

Local Disk Storage

Embedded Services

Session Replication

Logging



Pivotal

Some refactoring strategies

Local Disk Storage

Embedded Services

Session Replication

Logging

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

Pivotal

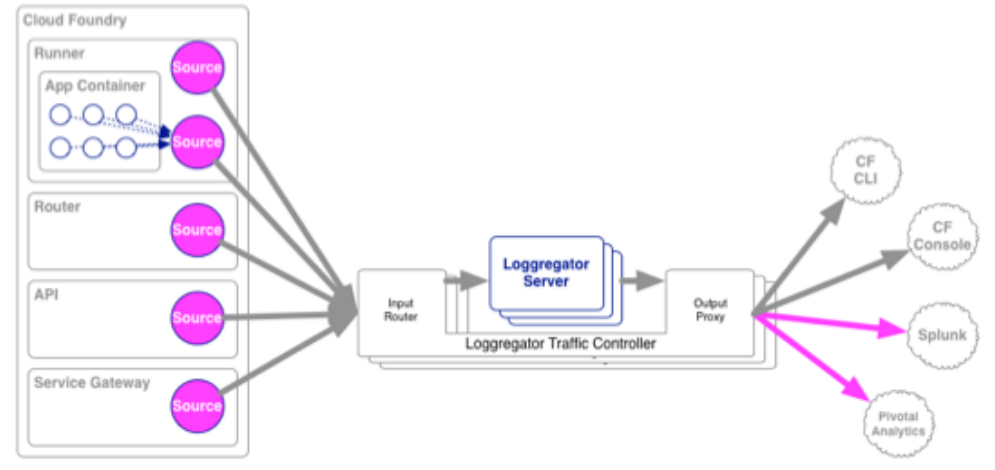
Some refactoring strategies

Local Disk Storage

Embedded Services

Session Replication

Logging



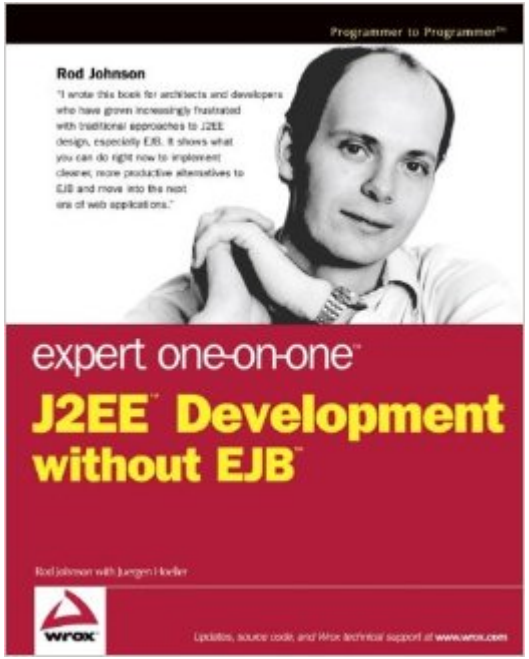
A NEW PLATFORM FOR A NEW ERA

J2EE/JEE Specifics

Pivotal

The JEE Story

- EJBs were seen as a mammoth spec
- It inspired Rod Johnson to create the Spring Framework
- The cumbersome model, coupled to fat application servers, and excessive plumbing code, led the market to embrace Spring as a top application framework in the world
- EJB 3 adopted 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
- For the majority of the spec, it is actually supported in web containers like tomcat. The main concerns are mostly around the application architecture and not the specs it uses
- There are just a few specs that are really not supported

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 it's 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 the same type of objects…
- Tomcat offers a read-only JNDI, so although it’s possible to run applications that rely on datasource JNDI objects, it's really not recommended, sometimes it's not that simple to port them.
- The service broker model and the environment variable system can offer the same functionality as a JNDI to store objects.

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). PCF 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 vendor specific, and it requires multiple ports opened, which is not feasible in PCF. 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, it is simple to implement a message driven pojo with spring as opposed to a message driven bean

Pivotal

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 (e.g. atomikos) but when talking about microservices, transactions should be a good boundary to start a conversation around the responsibility of the service

Pivotal

A NEW PLATFORM FOR A NEW ERA