Enable your applications to unleash the power of the cloud using Spring Framework

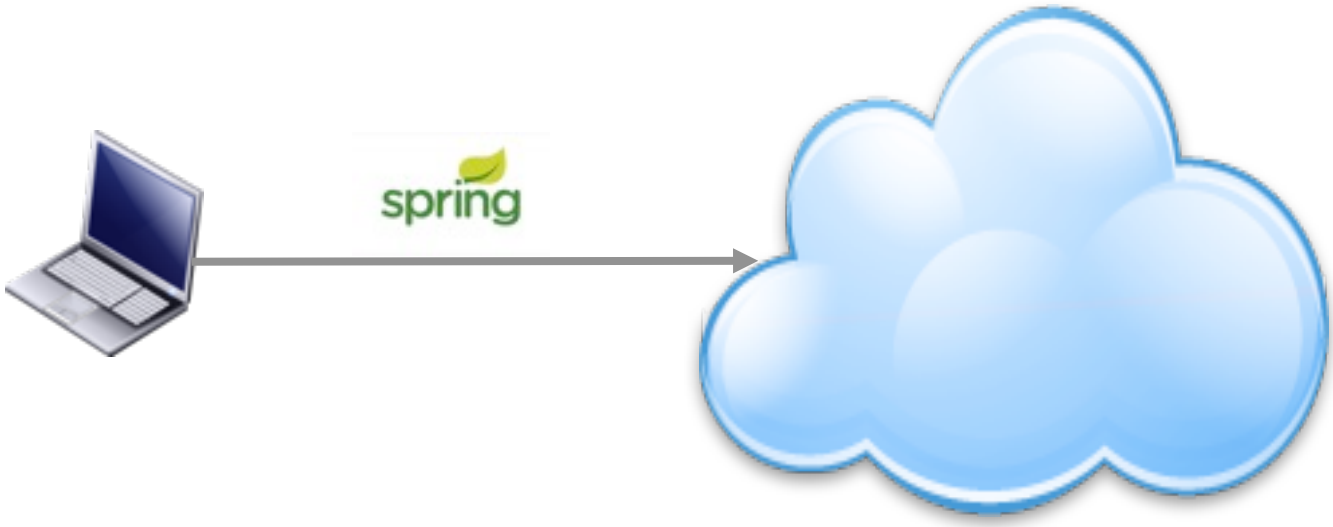
Tobias Karlsson

SpringSource a division of VMware

Who's Tobias?

- Started out as a Java consultant
- Worked 4 years with mobile services in MENA and APAC
- Currently living in Stockholm
- Currently working for SpringSource as a Sales Engineer in Northern Europe

Agenda

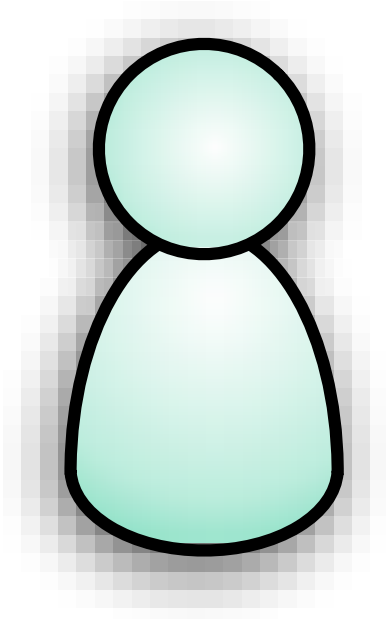



It’s a “New” World

Mobile first, mobile *only???*

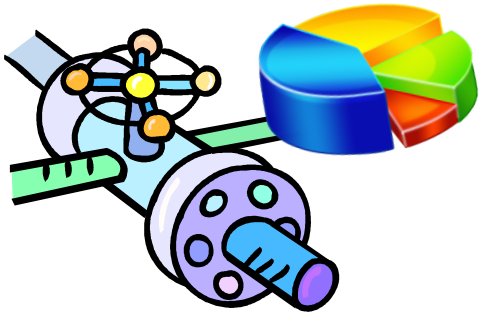
User Centric

web-app &
browser



users &
services

Real-time, contextual, social

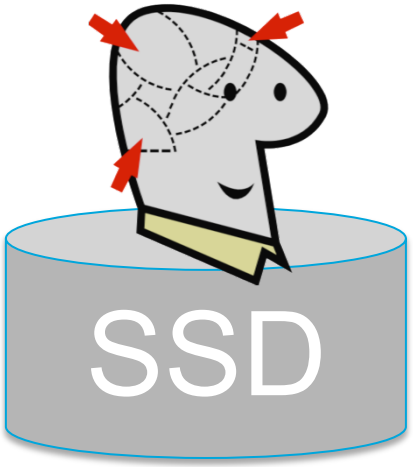


what happened last month?

what's happening now?

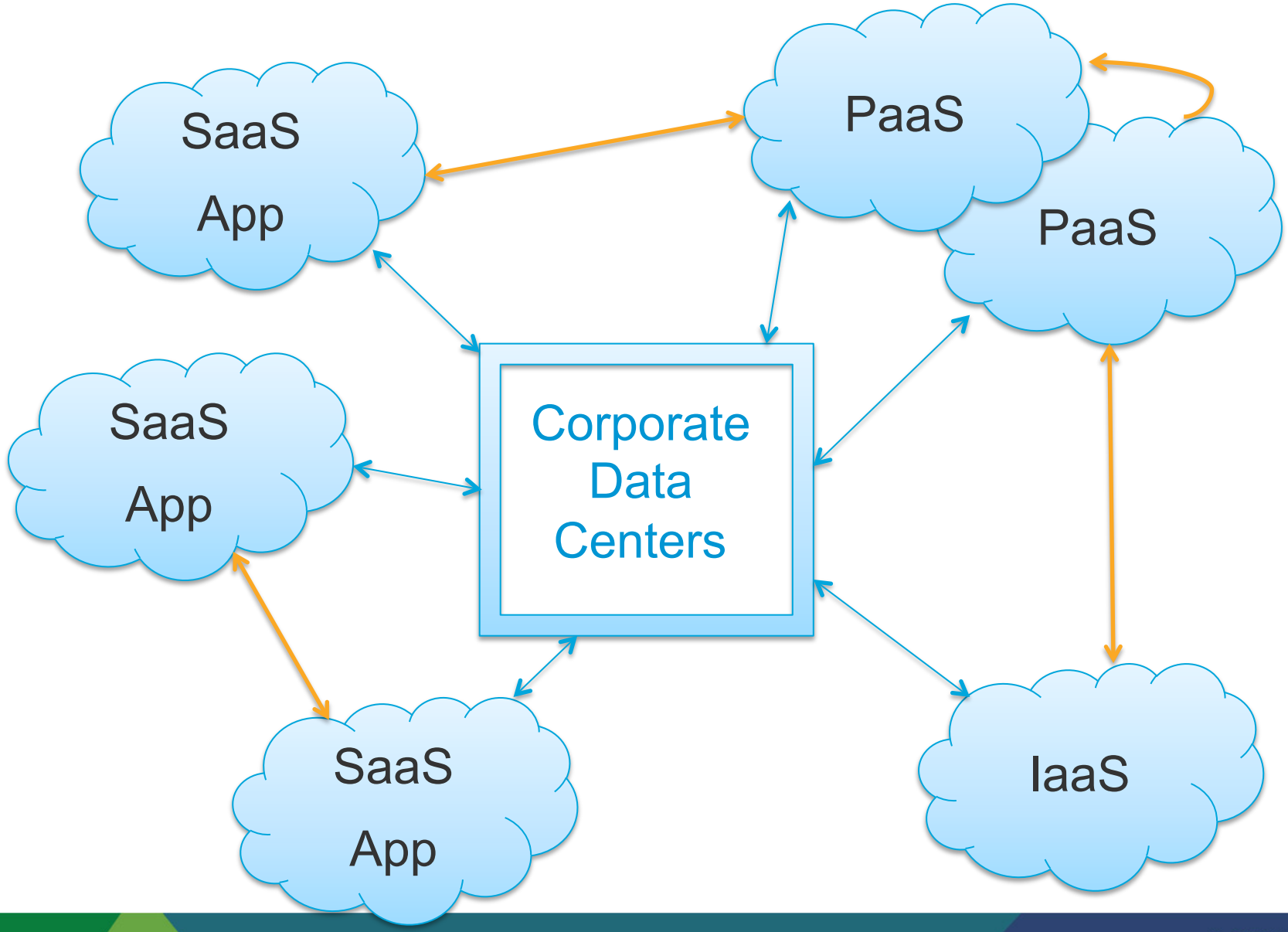
Solving Google style type problems

Latency Sensitive

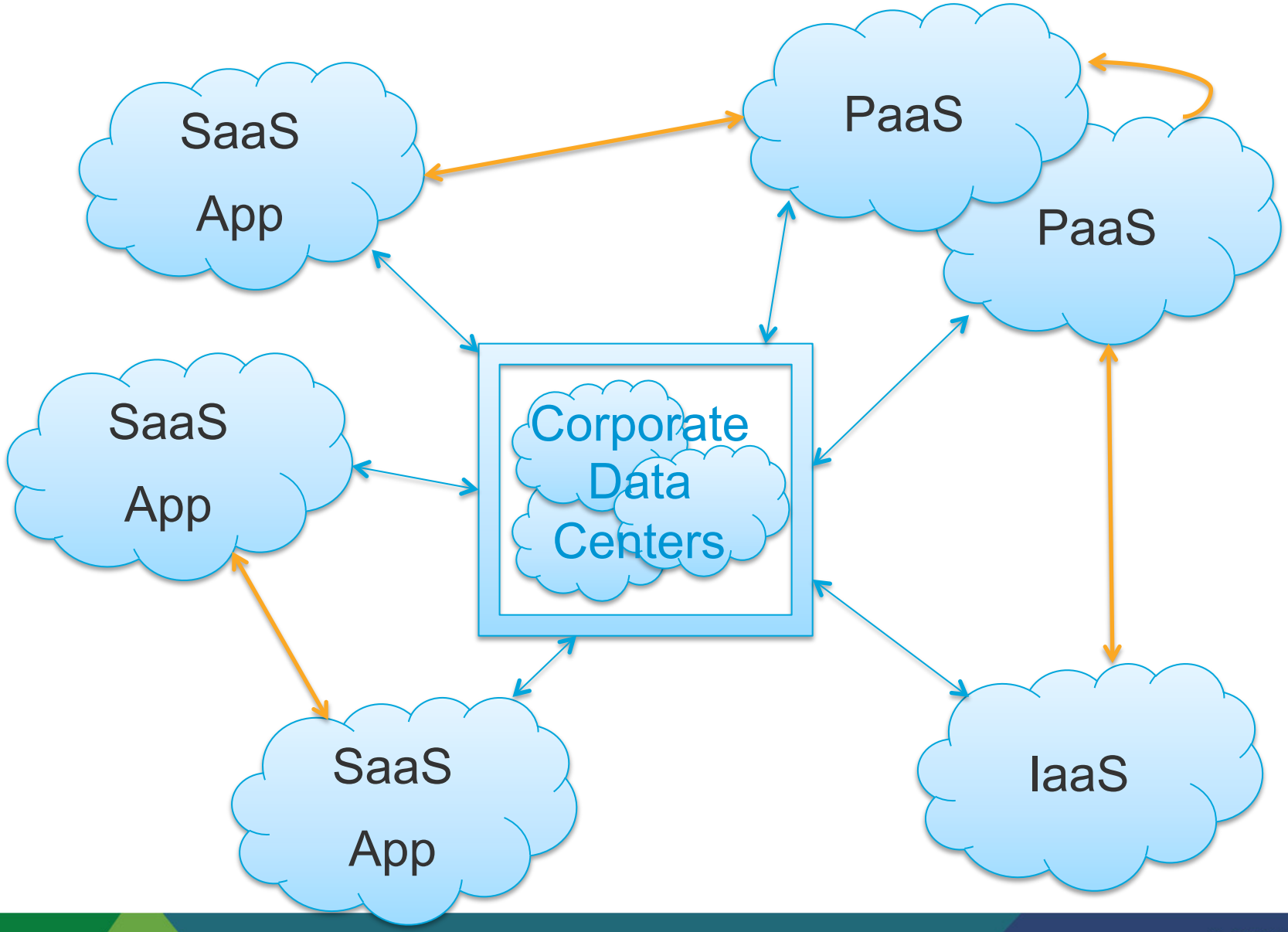


“memory is the new disk”

On modern infrastructure



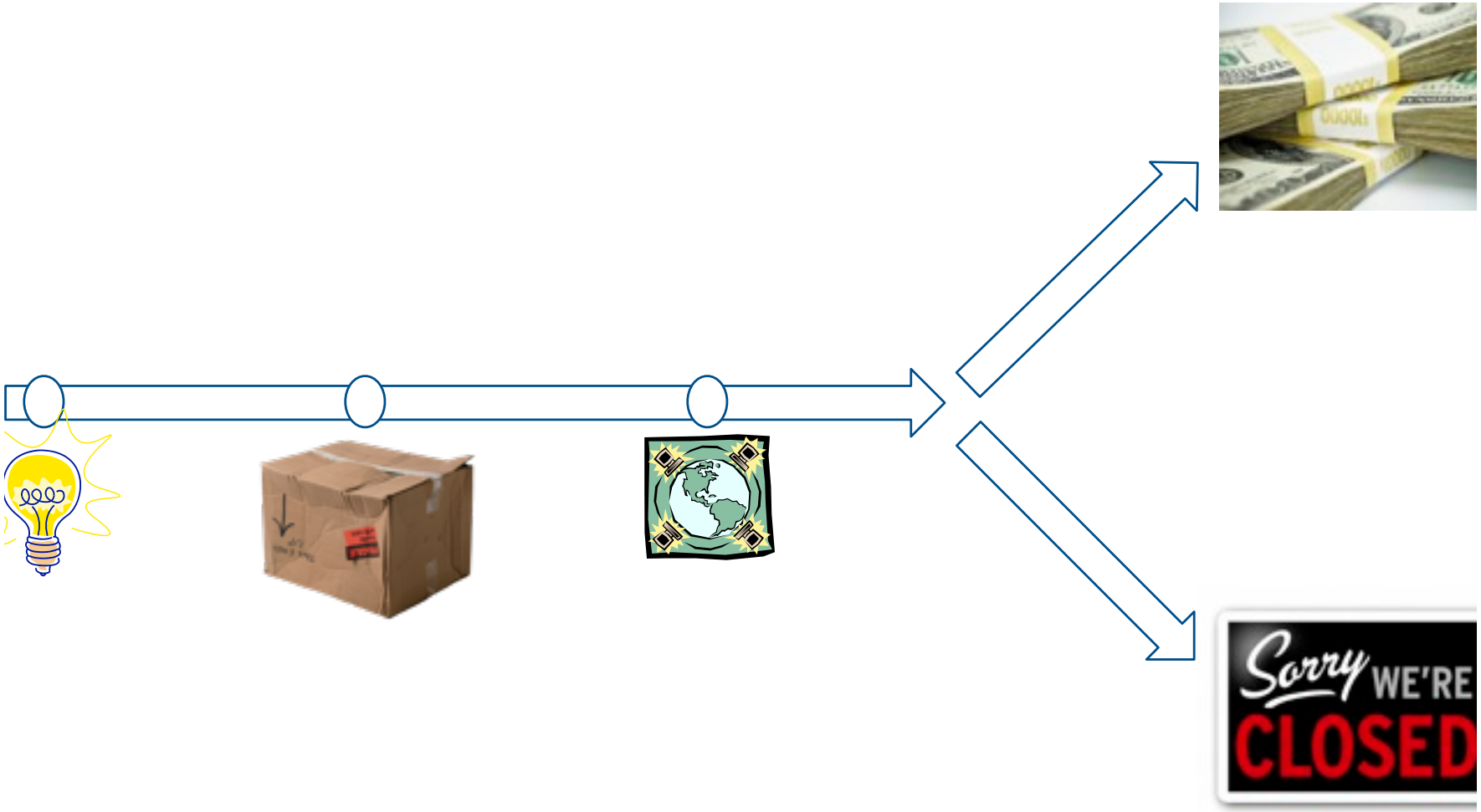
On modern infrastructure



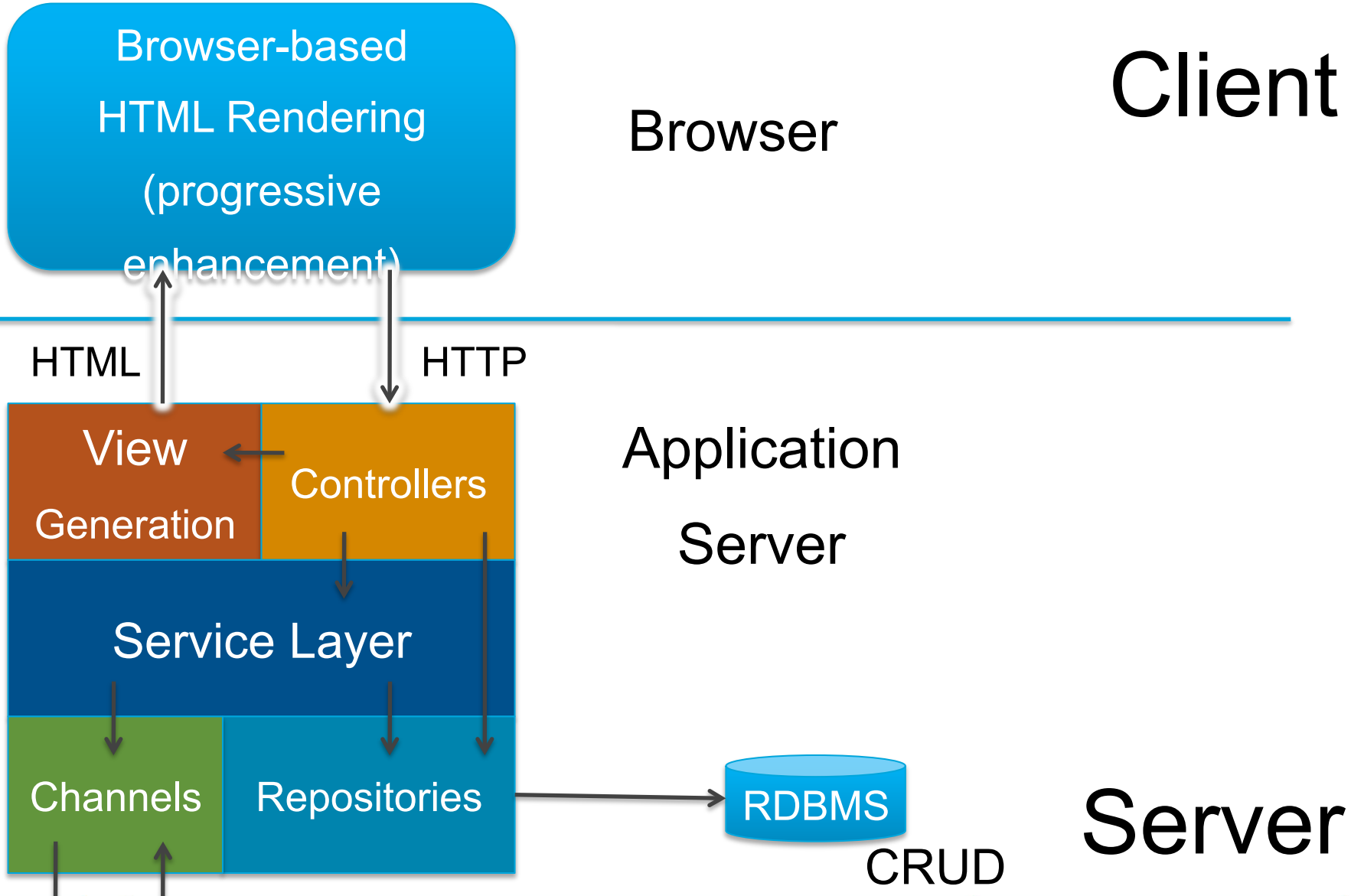
Frequent deployments

So, what does this all mean for your applications?

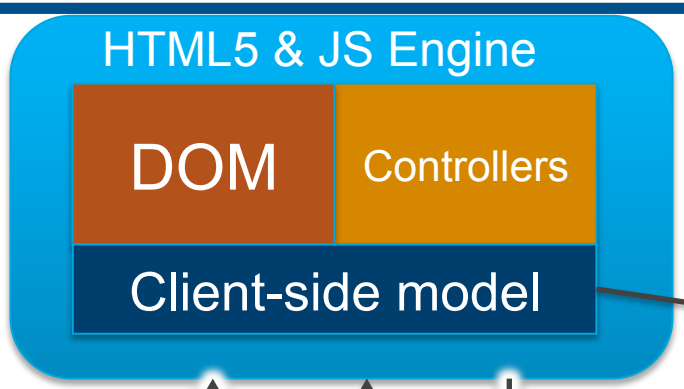
Real life developing a web product



Anatomy of a web app

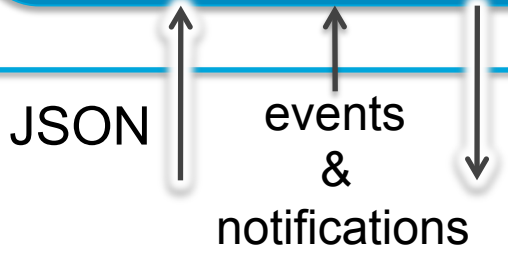


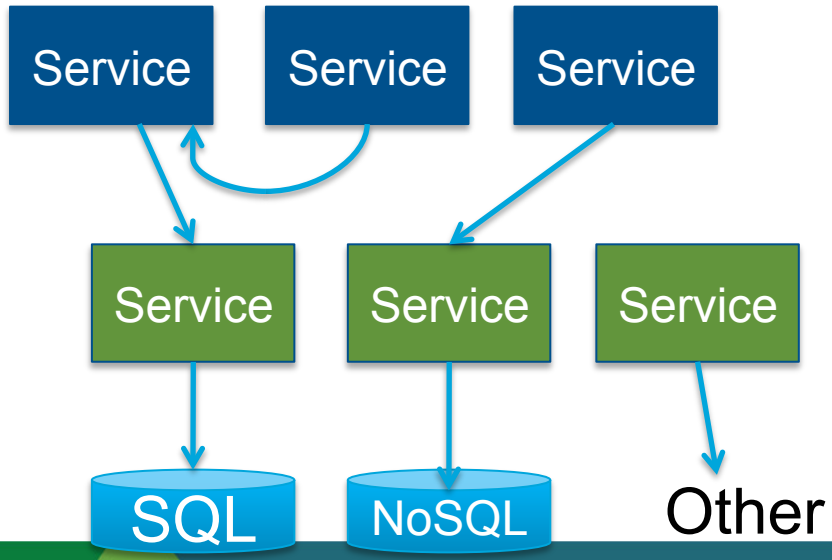
Anatomy of a next-gen app



Browser app or embedded in native

Client





business / domain services

platform services, web APIs

PaaS

Data isn't just relational

- **Relational database stores CRUD data, seeing huge rise in CRAP data**
  - Created, Replicated, Appended, Processed
- **Other store types:**
  - Document [MongoDB]
  - Key-value [Redis]
  - Column-family [Cassandra]
  - Graph database [Neo4j]
  - Blob stores
- **Trend is to supplement RDB with non-relational stores**

New Era Requires a Shift: Elasticity from Apps to Data

Develop using modern frameworks:
agile apps decoupled from middleware

Leverage runtime container optimized for virtualization: *provision in seconds*

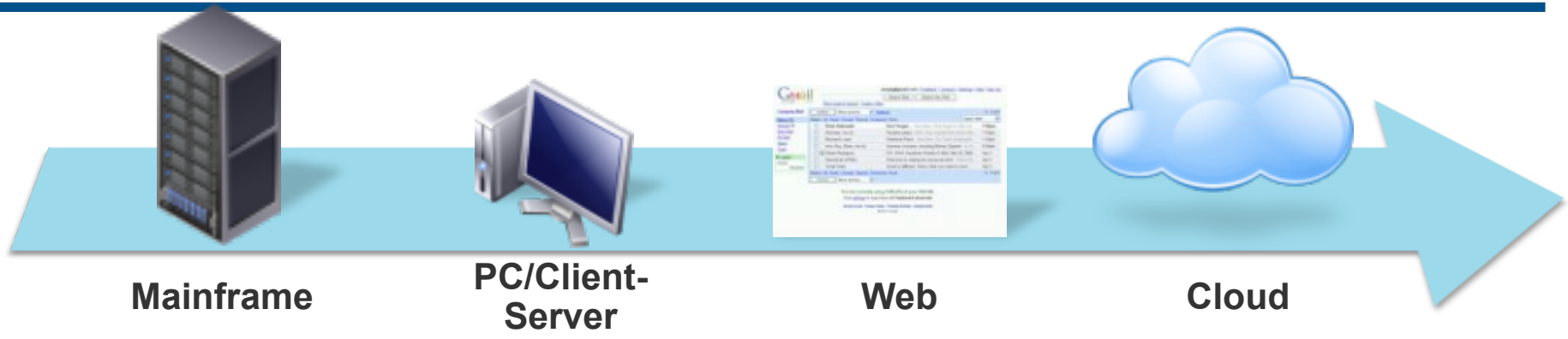
Store app state in elastic data cache:
maximize app scalability

Use cloud-friendly messaging protocols:
enable flexible app integration

Access app data through elastic data fabric and/or in-memory SQL:
maximize data scalability

The evolving runtime environment

New Eras Bring New Application Platforms



App Platform	COBOL	UNIX Services	App Server	PaaS

VMware Cloud Application Platform

- vFabric
- Cloud Foundry

Each new era in computing brings a new application platform: for the Cloud era it is "Platform as a Service"

vmware

Three layers of Cloud Computing

SaaS

Software as a Service

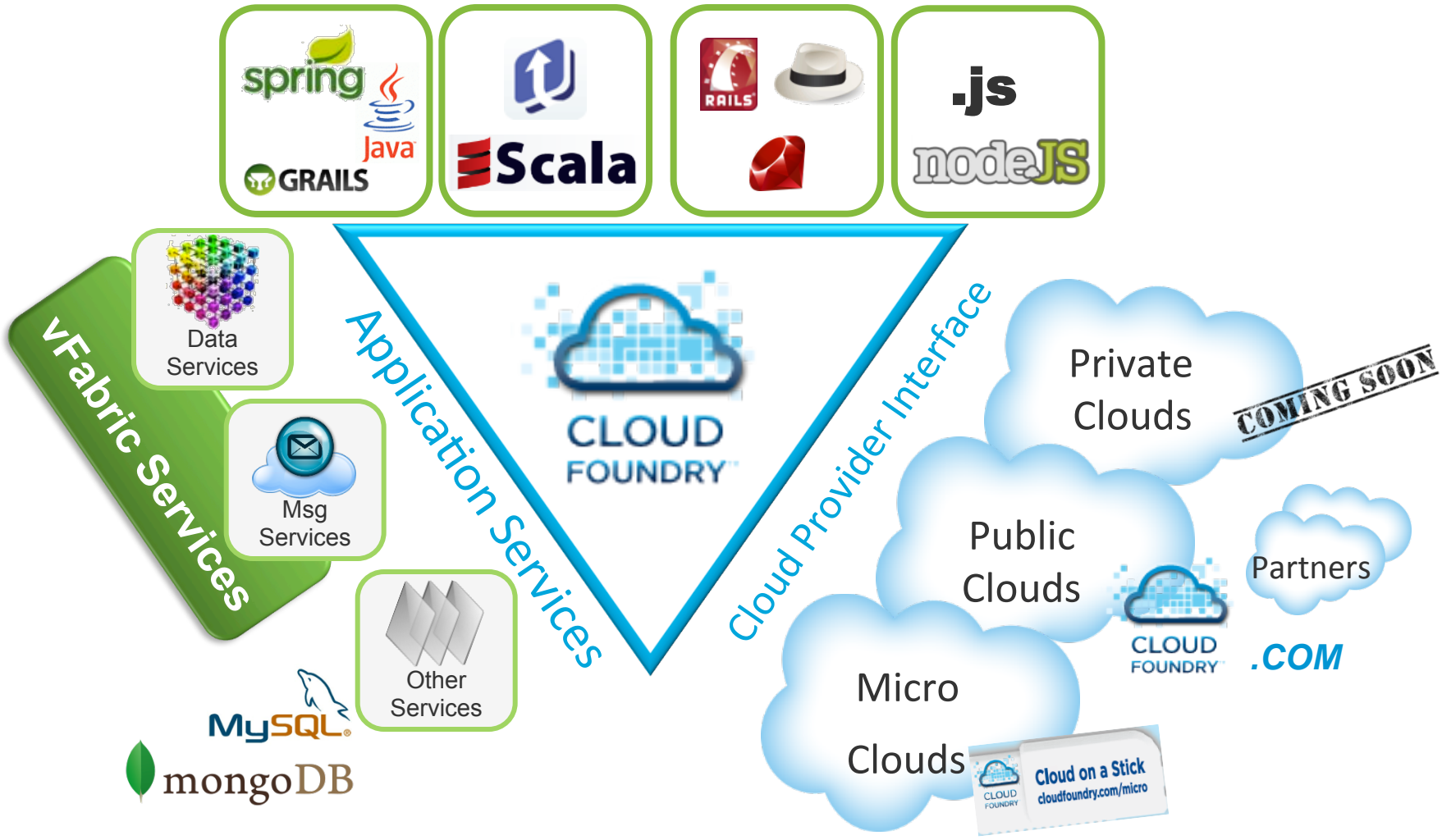
PaaS

Platform as a Service

IaaS

Infrastructure as a Service

Cloud Foundry Big Picture



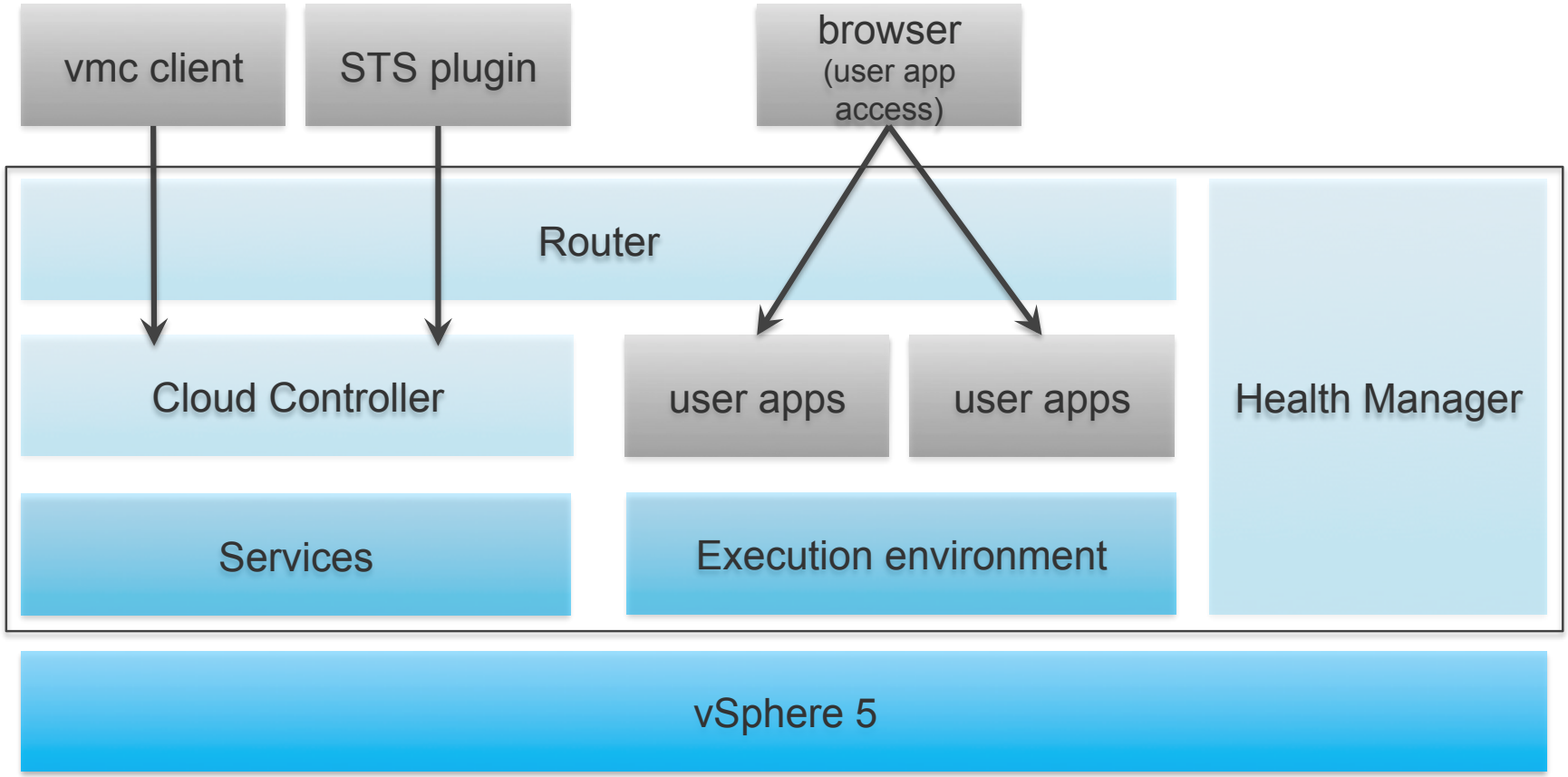
Cloud Foundry Open PaaS

- **Multiple languages and frameworks**
- **Multiple deployment options**
- **A variety of services**

Open Source

Extend it to meet your needs

Inside Cloud Foundry



Broad support for application frameworks

- **JVM**
  - Spring, Grails, Roo, Lift, plain Java
- **Ruby**
  - Rails, Sinatra
- **Node.js**
- **Community contributions**
  - Erlang, Python, PHP, .Net

JVM Frameworks

- **Unit of deployment: Java WARs**
  - Can run any standard WAR file
  - Servlet 2.5
    - don’t assume a particular container
- **Spring, Grails, Lift framework**
  - Auto-reconfiguration goodies

Inside Staged Applications

- **Stager packages applications into executable droplets**
  - provides a runtime container
  - can rewrite configuration files
  - can add libraries
- **For Spring/Grails applications**
  - provides a servlet container
  - deploys the app into the container
  - configures the container to listen on the correct port – adds auto-reconfiguration lib to the class path
  - rewrites web.xml
    - registers auto-reconfiguration BeanFactoryPostProcessor
    - registers CloudApplicationContextInitializer
  - adds JDBC drivers to class path
    - MySQL or PostgreSQL depending on bound services

“Leave my app alone!”

- **No Problem**
- **Plain Java framework**
  - bare minimum staging
  - no manipulation of configuration files – no additions to the class path
  - just your application

Elasticity on demand

- **Scale up in seconds**
  - vmc instances myapp +2
- **Scale down in seconds**
  - vmc instances myapp -2
- **Monitor your application instances**
  - per instance: memory, CPU, disk, uptime
  - vmc stats myapp

Surviving Disaster

- **Applications crash**
  - impossible to avoid
  - it will happen, sooner or later
- **Optimize for mean time to recovery**
  - mean time between failures is not as important

Services: Developer's perspective

- **Use services that meet application's needs**
- **Trivial provisioning of services**
  - vmc create-service mongodb documents-db
  - vmc bind-service inventory-app documents-db
- **Build service-focused polyglot apps**
  - Change languages and framework as needed
- **Not worry about operating services!**

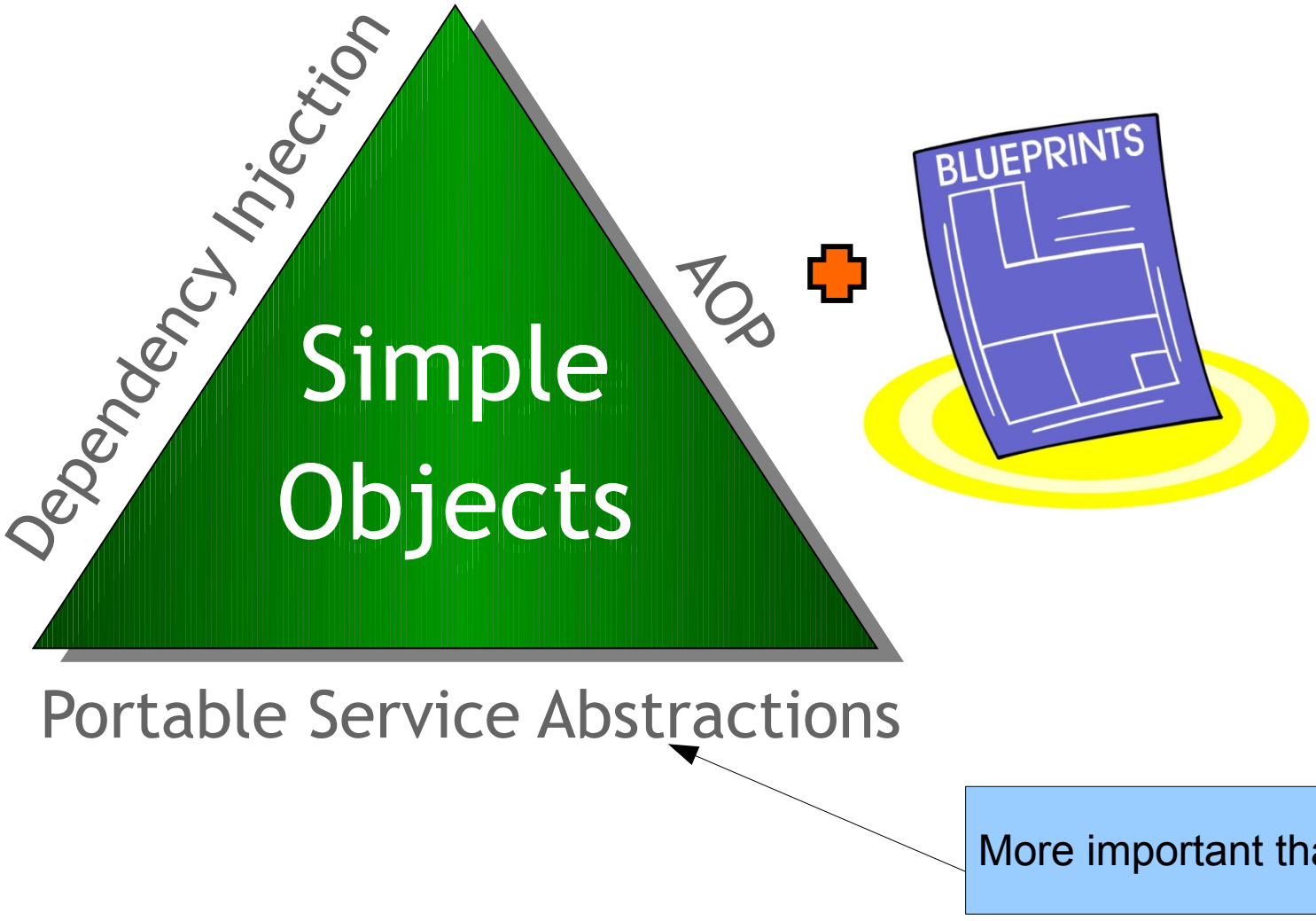
Exposing services

- VCAP_* environment variables provide configuration to applications
- VCAP_SERVICES with service connection info

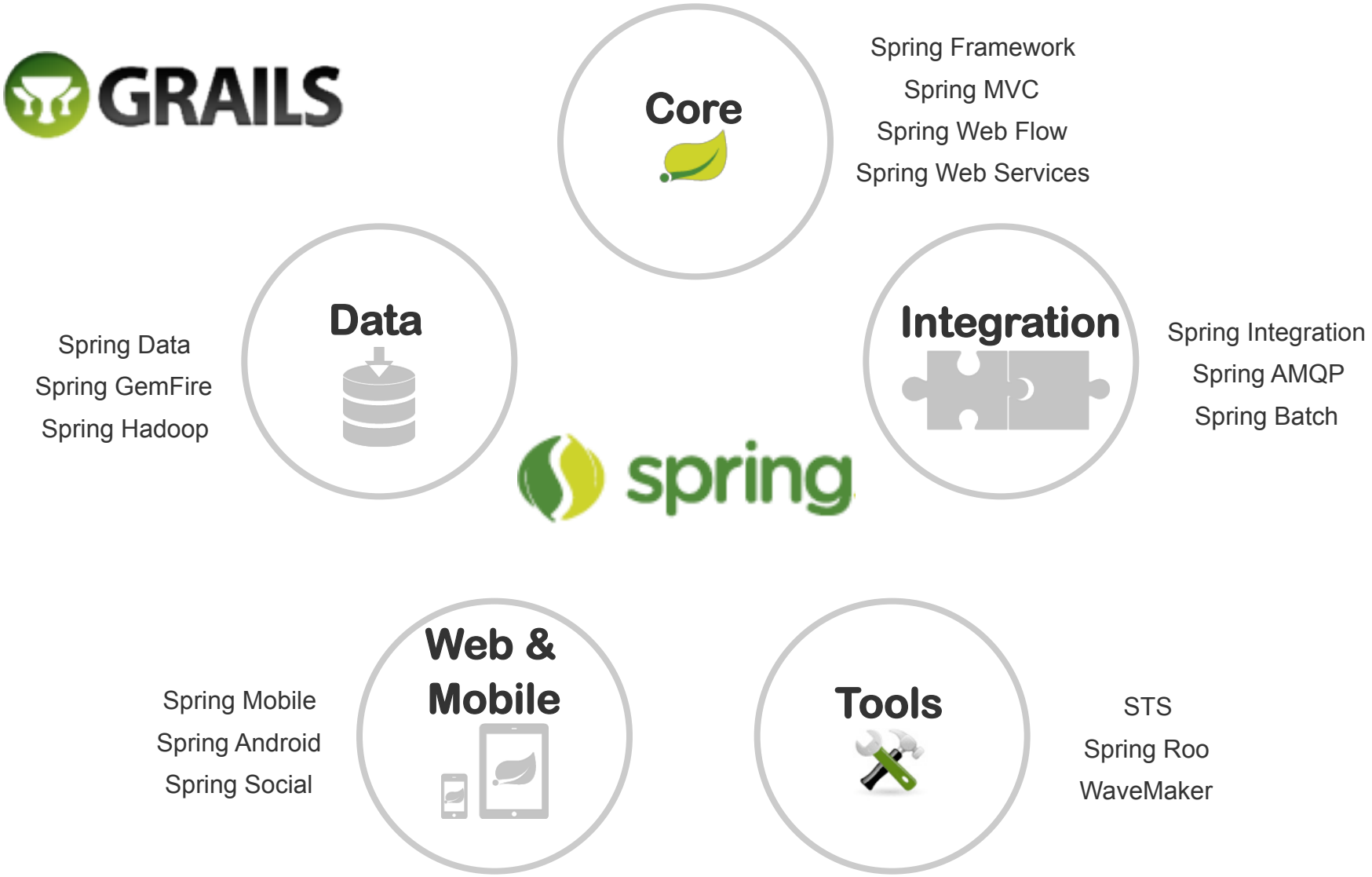
```
{
        "name": "inventory-db",
        "label": "mysql-5.1",
        "plan": "free",
        "credentials": {
                "node_id": "mysql_node_4",
                "hostname": "192.168.2.35",
                "port": 45678,
                "password": "dfdsf89414",
                "name": "kjkrewqr90",
                "user": "hwerkjewk”
        }
}
```

How will Spring help you to move your apps to the cloud?

Key Elements of Spring: Ready for 2012 & Beyond



Spring Focus Areas



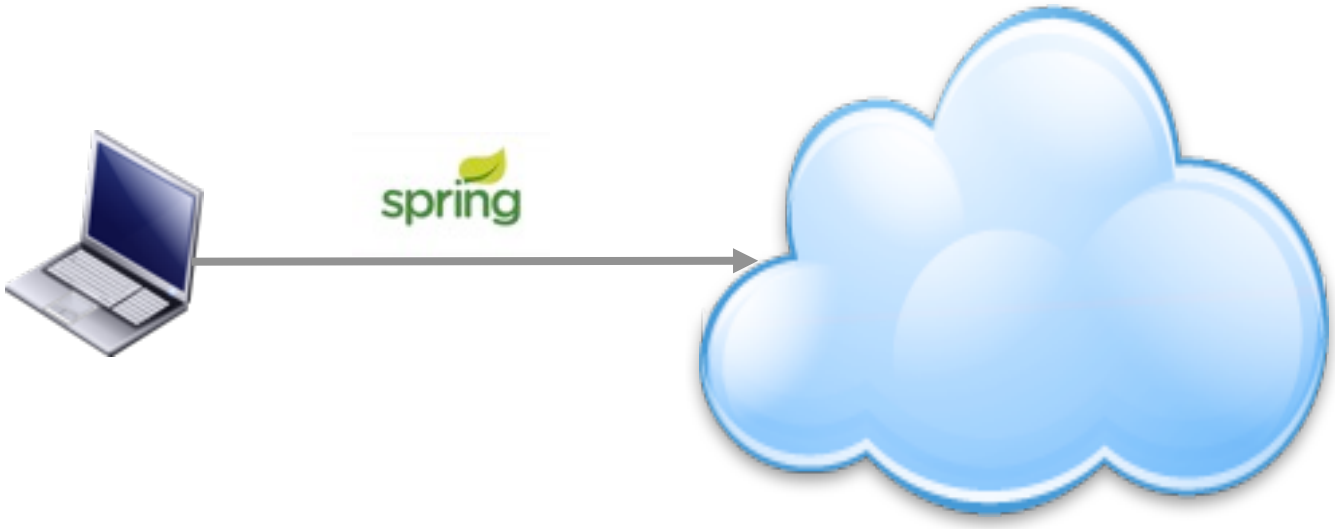
3

Spring 3.2 Strategy

- **Early support for latest Java specifications**
  - Java EE 7 as the central theme
  - As usual, support for selected specifications in individual form
  - With Java 8's language and API enhancements in mind already
- **Preserving compatibility with Java 5+**
  - Java SE 5+ as well as Java EE 5+
  - For the entire Spring 3.x branch
  - However, stronger focus on a Java SE 7 and Servlet 3.0+ world
- **Best possible experience on modern deployment environments**
  - From Tomcat 7 and WebSphere 8 to Google App Engine and Cloud Foundry

Auto-Reconfiguration: Getting Started

- Deploy Spring apps to the cloud **without changing a single line of code**
- Cloud Foundry automatically re-configures bean definitions to bind to cloud services
- Works with spring and grails frameworks



Auto-Reconfiguration: Relational DB

- **Detects beans of type javax.sql.DataSource**
- **Connects to MySQL or PostgreSQL services**
  - Specifies driver, url, username, password, validation query
- **Creates Commons DBCP or Tomcat DataSource**

```
<bean class="org.apache.commons.dbcp.BasicDataSource"
            destroy-method="close" id="dataSource">
      <property name="driverClassName" value="org.h2.Driver" />
      <property name="url" value="jdbc:h2:mem:" />
      <property name="username" value="sa" />
      <property name="password" value="" />
</bean>
```

Auto-Reconfiguration: ORM

- **Adjusts Hibernate Dialect**
- **Changes hibernate.dialect property to MySQLDialect (MyISAM) or PostgreSQLDialect**
  - org.springframework.orm.jpa.AbstractEntityManagerFactoryBean
  - org.springframework.orm.hibernate3.AbstractSessionFactoryBean(Spring 2.5 and 3.0)
  - org.springframework.orm.hibernate3.SessionFactoryBuilderSupport (Spring 3.1)

```
<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
            id="entityManagerFactory">
      <property name="dataSource" ref="dataSource"/>
</bean>
```

Auto-Reconfiguration: NoSQL

- **Works with Spring Data**
  - Connects to MongoDB service (Document Store)
  - Connects to Redis service (Key-Value Store)

What is Spring Data?

- **Umbrella of projects embracing the various new data access technologies**
  - Non-relational DBs
  - Map-Reduce frameworks – Cloud-based data services
- **Enhances developer productivity**
  - Removes API noise, boiler-plate code and resource management
  - Offers a consistent programming model
- **Builds on top of existing Spring features and projects**
  - e.g. Inversion of control, life-cycle management, type conversion, portable data access exceptions, caching
  - Easy to add to your application

Auto-Reconfiguration: RabbitMQ

- **Works with Spring AMQP 1.0**
  - Provides publishing, multithreaded consumer generation, and message converters
  - Facilitates management of AMQP resources while promoting DI and declarative configuration
- **Detects beans of type**

  org.springframework.amqp.rabbit.connection.ConnectionFactory
- **Connects to Rabbit Service**
  - Specifies host, virtual host, port, username, password
- **Creates CachingConnectionFactory**

```
<rabbit:connection-factory id="rabbitConnectionFactory" host="localhost"
password="testpwd" port="1238" username="testuser" virtual-host="virthost" />
```

Auto-Reconfiguration: How it works

- **Cloud Foundry installs a BeanFactoryPostProcessor in your application context during staging**
  - Adds jar to your application
  - Modifies web.xml to load BFPP
- **Adds context file to contextConfigLocation – web-app context-param**
  - Spring MVC DispatcherServlet init-param
- **Adds PostgreSQL and MySQL driver jars as needed for DataSource reconfiguration**

Auto-Reconfiguration: Limitations

- **Exactly one service of a given type bound to application**
  - e.g. Only one relational DB service (MySQL or PostgreSQL)
- **Exactly one bean of matching type in application**
  - e.g. Only one bean of type javax.sql.DataSource
- **Auto-Reconfiguration is skipped if limitations not met**
- **Custom configuration is not preserved**
  - e.g. Pool sizes, caching or connection properties
- **Use cloud namespace instead**

Auto-Reconfiguration: Opting Out

- **Two ways to explicitly disable auto-reconfiguration:**
  - Choose framework “JavaWeb” when deploying application
    - Application remains unchanged during staging
    - Unable to take advantage of profile feature
  - Use any <cloud> element that creates a bean representing a service
    - Explicit control of service bindings implies that auto- reconfiguration is unnecessary

Introducing… the Cloud Namespace

- **<cloud:> namespace for use in Spring app contexts**
- **Provides application-level control of bean service bindings**
- **Recommended for development of new cloud apps**
- **Use when:**
  - You have multiple services of the same type
  - You have multiple connecting beans of the same type
    - e.g. DataSource, MongoDBFactory
  - You have custom bean configuration
    - e.g. DataSource pool size, connection properties

Including Cloud Namespace in Your App

- Declare Maven Dependency and Repository
- Add namespace declaration to app context files

```
<dependencies>
    <dependency>
        <groupId>org.cloudfoundry</groupId>
        <artifactId>cloudfoundry-runtime</artifactId>
        <version>0.8.1</version>
    </dependency>
.....
<repositories>
    <repository>
        <id>org.springframework.milestone</id>
        <name>Spring Framework Milestone Repository</name>
        <url>http://maven.springframework.org/milestone</url>
    </repository>
.....
```

<cloud:service-scan>

- **Scans all services bound to the application and creates a bean of an appropriate type for each**
  - Same bean types as auto-reconfiguration
- **Useful during early development phases**

```
<beans   ...
      xmlns:cloud="http://schema.cloudfoundry.org/spring"
      xsi:schemaLocation="http://schema.cloudfoundry.org/spring
      http://schema.cloudfoundry.org/spring/cloudfoundry-spring-0.8.xsd
      ...">

      <cloud:service-scan/>

</beans>
```

<cloud:service-scan> Autowire Dependencies

- Created beans can be autowired as dependencies
- Use @Qualifier with service name if multiple services of same type bound to app

```
@Autowired(required=false)
private ConnectionFactory rabbitConnectionFactory;

@Autowired
private RedisConnectionFactory redisConnectionFactory;

@Autowired
@Qualifier("test_mysql_database")
private DataSource mysqlDataSource;

@Autowired(required=false)
@Qualifier("test_postgres_database")
private DataSource postgresDataSource;
```

<cloud:service-scan> Declare Dependencies

- **Created beans ids will match service names**
- **Use service name in dependency declarations**

```
<!--  Connects to cloud service named "contacts-db" -->
<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
            id="entityManagerFactory">
       <property name="dataSource" ref="contacts-db"/>
</bean>

<!--  Connects to cloud service named "tweet-cache" -->
<bean id= "redisTemplate" class= "org.sf.data.redis.core.RedisTemplate" >
      <property name="connectionFactory" ref="tweet-cache"/>
      ....
</bean>
```

<cloud:data-source>

- **Configures a DataSource bean**
  - Commons DBCP or Tomcat DataSource
- **Basic attributes:**
  - id: defaults to service name
  - service-name: only needed if you have multiple relational database services bound to the app

```
<cloud:data-source id="dataSource"/>

<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
            id="entityManagerFactory">
      <property name="dataSource" ref="dataSource"/>
</bean>
```

Spring 3.1 Environment Abstraction

- **Bean definitions for a specific environment (Profiles)**
  - e.g. development, testing, production
  - Possibly different deployment environments
  - Activate profiles by name
    - spring.profiles.active system property
    - Other means outside deployment unit
    - “default” profile activates if no other profiles specified
- **Custom resolution of placeholders**
  - Dependent on the actual environment
  - Ordered property sources
- **Requires Spring 3.1 (or later)**

Isolating Cloud Foundry Configuration

- **Switch between local, testing and Cloud Foundry deployments with Profiles**
- **“cloud” profile automatically activates on Cloud Foundry**
  - usage of the cloud namespace should occur within the cloud profile block

Isolating Cloud Foundry Configuration

```
<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
        id="entityManagerFactory">
    <property name="dataSource" ref="dataSource"/>
</bean>

<beans profile="cloud">
    <cloud:data-source id="dataSource" />
</beans>

<beans profile="default">
    <bean class="org.a.commons.dbcp.BasicDataSource" id="dataSource">
        <property name="url" value="jdbc:mysql://localhost/stalker" />
    </bean>
</beans>
```

Using Profiles to Enable Features

- **Use profiles to add features when deploying to Cloud Foundry**
  - e.g. Using Send Grid to send email

```
<beans profile="cloud">
      <bean name="mailSender" class="example.SendGridMailSender">
            <property name="apiUser" value="youremail@domain.com" />
            <property name="apiKey" value="secureSecret" />
      </bean>
</beans>
```

Cloud Properties

- **Cloud Foundry uses Environment abstraction to automatically expose properties to Spring 3.1 apps**
  - Basic information about the application, such as its name and the cloud provider
  - Detailed connection information for bound services
    - cloud.services.{service-name}.connection.{property}
    - aliases for service name created based on the service type
      - e.g. “cloud.services.mysql.connection.{property}”
      - only if there is a single service for that type bound

Profile Support: How it works

- **Cloud Foundry installs a custom ApplicationContextInitializer in your app during staging**
  - Modifies web.xml
    - Adds to contextInitializerClasses context-param
- **Adds “cloud” as an active profile**
- **Adds a PropertySource to the Environment**

Summary

- **Comprehensive set of services**
- **Spring developers served well**
  - Dependency injection proves the right approach, again!
- **Many simplifications to use services**
  - Auto-reconfig
  - Cloud namespace
  - Cloud profile
- **Focus on your app; let us worry about services!**

Architectural Principles for the cloud

- **Decoupled**
- **Elastic**
  - Early instrumentation
  - Continuous optimization
  - Fast provisioning
- **Lightweight**
- **Framework based**
  - Spring Data
- **Container independent**
- **Enhanced with new NoSQL**

For any questions, contact

tkarlsson@vmware.com

+46 702 840075