vmware | SPRING & CLOUD FOUNDRY 开发者大会

中国北京 - 2012年12月7日-8日

Spring on Cloud Foundry: a Marriage Made in Heaven

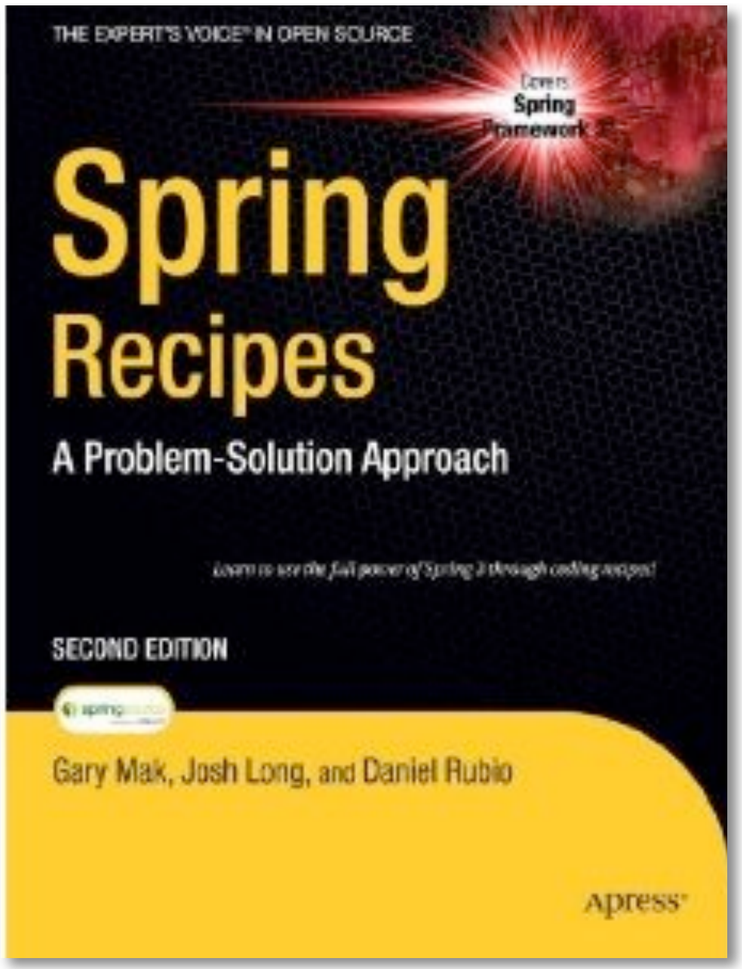
Josh Long

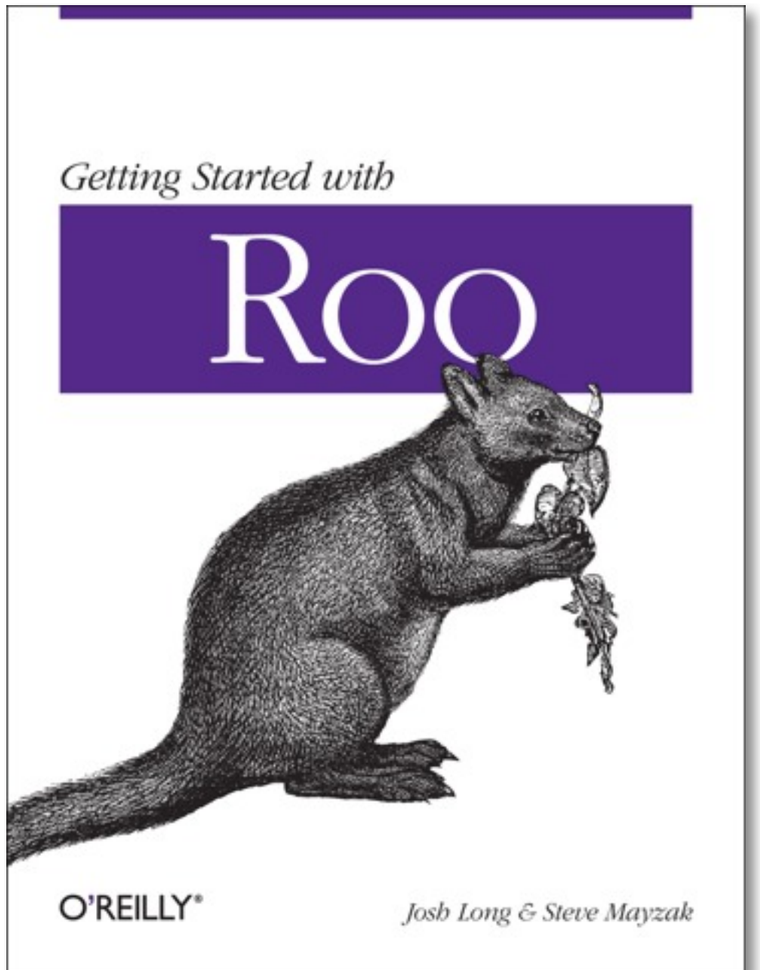
***Spring Developer Advocate**, SpringSource, a Division of VMWare*

http://www.joshlong.com || @starbuxman || josh.long@springsource.com

About Josh Long (龙之春)

Spring Developer Advocate
twitter: @starbuxman
weibo: @springsource
josh.long@springsource.com



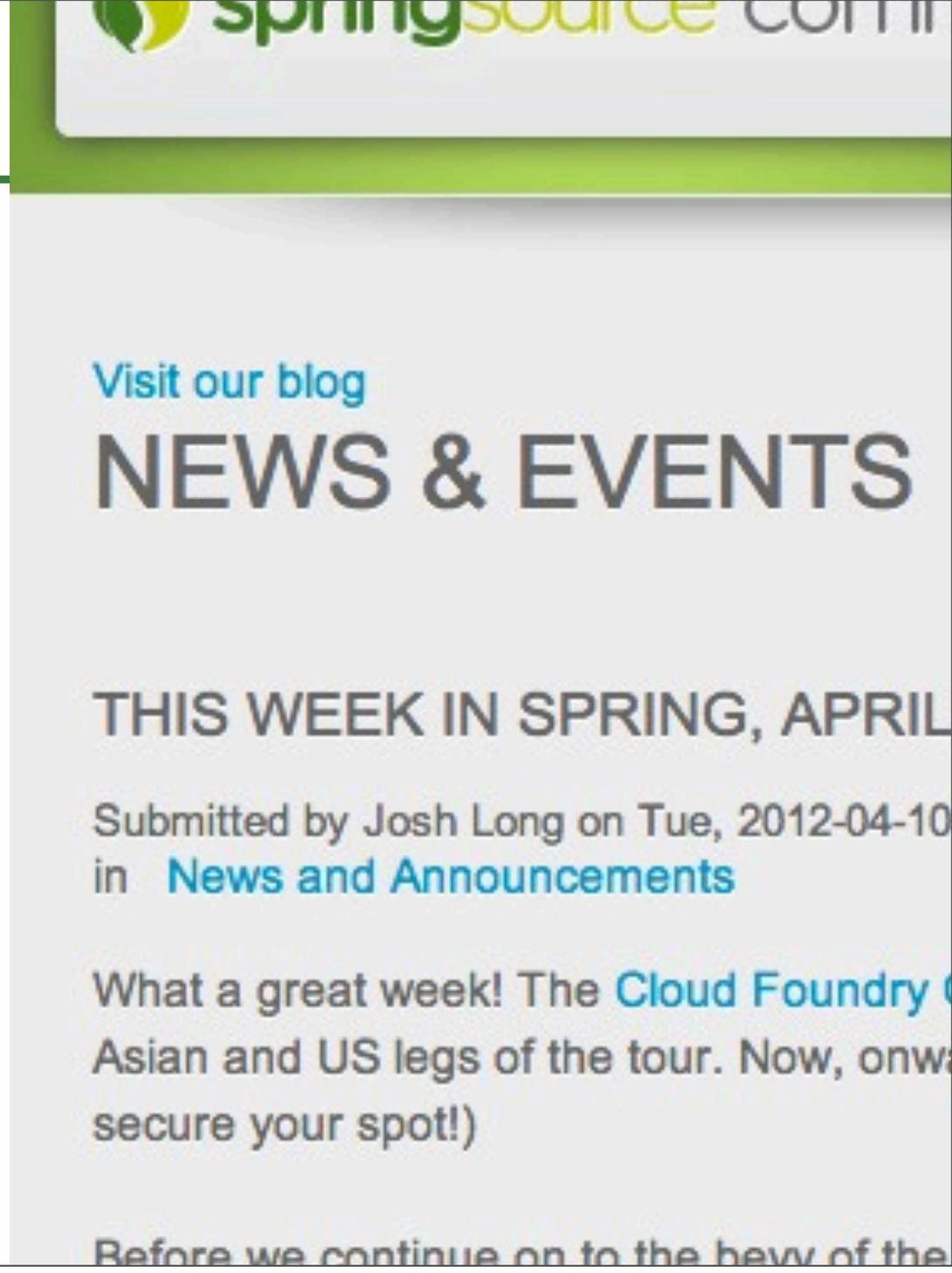


About Josh Long

Spring Developer Advocate
twitter: @starbuxman
josh.long@springsource.com

Contributor To:

- **Spring Integration**
- **Spring Batch**
- **Spring Hadoop**
- Activiti Workflow Engine
- Akka Actor engine



Why Cloud Foundry?

We have amazing languages, tools and frameworks

Idea → **Working Code**

But how long to deploy?

working code

Deployed

But how long to deploy?

working code

Deployed

Hours?

But how long to deploy?

working code

Deployed

Hours?

Days?

But how long to deploy?

working code

Deployed

Hours?

Days?

Weeks?

But how long to deploy?

working code

Deployed

Hours?

Days?

Weeks?

Months?

And if it’s the developers...

Imagine if architects had to be the janitor for every building they designed. This is how the development team felt prior to moving to Windows Azure.

Duncan Mackenzie Nov 07, 2011
http://www.infoq.com/articles/Channel-9-Azure

Challenges of cloud deployment

- Yet another environment for your application to run in
  - Different Java runtime
  - Different databases – sometimes not relational!
- Dynamic environment ⇒ server names not fixed
- Elastic scaling ⇒ servers come and go
- Cloud applications must integrate with off cloud applications

Challenges of cloud deployment

Challenges of cloud deployment

These are problems already solved by the Spring framework

Why Spring?

Why Are We Here?

> Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.
>
> **-Bob Martin**

Why Are We Here?

do NOT reinvent the Wheel!

The Spring framework

- De-facto standard programming model for enterprise Java
- Two million+ developers
- Rapid evolution
  - Spring 1.0 – March 2004
  - Spring 2.0 – October 2006
  - Spring 2.5 – December 2007
  - Spring 3.0 – December 2009
  - Spring 3.1 - December 2011
  - Spring 3.2 - December 2012
- Complete backward compatibility

Spring's aim:

bring simplicity to java development

web tier & RIA

service tier

batch processing

integration & messaging

data access / NoSQL / Big Data

mobile

The Spring framework

the cloud:	lightweight	traditional
CloudFoundry Google App Engine Amazon BeanStalk Others	tc Server Tomcat Jetty	WebSphere JBoss AS WebLogic (on legacy versions, too!)

Spring Makes Building Applications Easy...

Tell Spring About Your Objects

```
package org.springsource.examples.spring31.services;
...
@Configuration
@ComponentScan(“the.package.with.beans.in.it”)
public class ServicesConfiguration {

...

}
```

Tell Spring About Your Objects

```
public class Main {
  static public void main (String [] args) throws Throwable {
        ApplicationContext ac = new AnnotationConfigApplicationContext(
            org.springsource.examples.spring31.services.ServicesConfiguration.class);
        ...
  }
}
```

Tell Spring About Your Objects

```
package the.package.with.beans.in.it;

@Service
public class CustomerService {

   public Customer createCustomer(String firstName,
                                   String lastName,
                                   Date signupDate)    {


   }

      ...

}
```

I want Database Access ...

```
package org.springsource.examples.spring31.services;
...
@Configuration
public class ServicesConfiguration {


    @Bean
    public DataSource dataSource() throws Exception {
        SimpleDriverDataSource simpleDriverDataSource =
          new SimpleDriverDataSource();
         ....
        return simpleDriverDataSource;
    }

}
```

I want Database Access ... with Hibernate 4 Support

```
package org.springsource.examples.spring31.services;
...
@Configuration
public class ServicesConfiguration {
    ...

    @Bean
    public SessionFactory sessionFactory() throws Exception {
        Properties props = new Properties();
        // ... show_sql, dialect, etc.
        return new LocalSessionFactoryBuilder( dataSource() )
                .addAnnotatedClasses(Customer.class)
                .addProperties(props)
                .buildSessionFactory();
    }

}
```

I want Database Access ... with Hibernate 4 Support

```
package the.package.with.beans.in.it;
...
@Service
public class CustomerService {

    @Inject
    private SessionFactory sessionFactory;

    public Customer createCustomer(String firstName,
                                   String lastName,
                                   Date signupDate)    {
        Customer customer = new Customer();
        customer.setFirstName(firstName);
        customer.setLastName(lastName);
        customer.setSignupDate(signupDate);

        sessionFactory.getCurrentSession().save(customer);
        return customer;
    }

    ...

}
```

I want Declarative Transaction Management...

```
package org.springsource.examples.spring31.services;
...
@Configuration
@EnableTransactionManagement
public class ServicesConfiguration {

   ...

   @Bean
    public PlatformTransactionManager transactionManager() throws Exception {
       return new HibernateTransactionManager(this.sessionFactory());
    }
}
```

I want Declarative Transaction Management...

```
package the.package.with.beans.in.it;
...
@Service
public class CustomerService {

    @Inject
    private SessionFactory sessionFactory;

    @Transactional
    public Customer createCustomer(String firstName,
                                   String lastName,
                                   Date signupDate)    {
        Customer customer = new Customer();
        customer.setFirstName(firstName);
        customer.setLastName(lastName);
        customer.setSignupDate(signupDate);

        sessionFactory.getCurrentSession().save(customer);
        return customer;
    }
    ...
}
```

I want Declarative Cache Management...

```
package org.springsource.examples.spring31.services;
...
@Configuration
@EnableTransactionManagement
@EnableCaching
public class ServicesConfiguration {
   ...
   ...

   @Bean
   public CacheManager cacheManager() {
      SimpleCacheManager scm = new SimpleCacheManager();
      Cache cache = new ConcurrentMapCache("customers");
      scm.setCaches(Arrays.asList(cache));
      return scm;
   }
}
```

I want Declarative Cache Management...

```
package the.package.with.beans.in.it;
...
@Service
public class CustomerService {

    @Inject
    private SessionFactory sessionFactory;

    @Transactional
    @Cacheable("customers")
    public Customer createCustomer(String firstName,
                                   String lastName,
                                   Date signupDate)    {
        Customer customer = new Customer();
        customer.setFirstName(firstName);
        customer.setLastName(lastName);
        customer.setSignupDate(signupDate);

        sessionFactory.getCurrentSession().save(customer);
        return customer;
    }
}
```

I want a RESTful Endpoint...

```
package org.springsource.examples.spring31.web;
..

@Controller
public class CustomerController {

   @Inject
   private CustomerService customerService;

   @RequestMapping(value = "/customer/{id}",
                           produces = MediaType.APPLICATION_JSON_VALUE)
   public @ResponseBody Customer customerById(@PathVariable("id") Integer id) {
      return customerService.getCustomerById(id);
   }
    ...
}
```

vmware | SPRING & CLOUD FOUNDRY
开发者大会

Building Spring Applications Targeting Cloud Foundry

UNDERSTANDING CLOUD FOUNDRY SERVICES

Cloud Foundry: Services

- Services are one of the extensibility planes in Cloud Foundry
  - there are more services being contributed by the community daily!
- MySQL, Redis, MongoDB, RabbitMQ, PostgreSQL
- Services may be shared across applications
- Cloud Foundry abstracts the provisioning aspect of services through a uniform API hosted in the cloud controller
- It's very easy to take an app and add a service to the app in a uniform way
  - Cassandra? COBOL / CICS, Oracle

Cloud Foundry: Services

- Take Advantage of Services
  - they cost *nothing* to setup
  - they deliver value

- They Encourage Better Architectures
  - Need a fast read-write cache? **Redis** is ready to go!
  - Need to store long-tail documents? Give **MongoDB** a try
  - Need to decouple what applications do from when they do it? Use messaging and **RabbitMQ**

Cloud Foundry Exposes Services Through Environment Variables

$VCAP_SERVICES:
{"redis-2.2":[{"name":"redis_sample","label":"redis-2.2","plan":"free",
"tags":["redis","redis-2.2","key-value","nosql"],
"credentials":
{"hostname":"172.30.48.40",
"host":"172.30.48.40",
"port":5023,
"password":"8e9a901f-987d-4544-9a9e-ab0c143b5142",
"name":"de82c4bb-bd08-46c0-a850-af6534f71ca3"}
}],
"mongodb-1.8":[{"name":"mongodb-e7d29","label":"mongodb-1.8","plan":"free","tags”:
.....

Spring's the Best Toolkit for Your Services

- Spring Data
  - supports advanced JPA, MongoDB, Redis connectivity
- Spring AMQP, Spring Integration
  - Supports messaging, and event-driven architectures
- Spring core
  - Has best-of-breed support for RDBMS access, be it through JPA, JDBC, JDO, Hibernate, etc.

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

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
- Replaces existing DataSource

```
<bean class = "...BasicDataSource" id="dataSource">
  <property name = "url" value = "jdbc:h2:mem"/>
  <property name ="password" value =""/>
  <property name = "username" value = "sa"/>
  <property name = "driverClass" value = "com.h2.Driver"/>
</bean>
```

```
import org.apache.commons.dbcp.BasicDataSource;
...
@Bean(destroyMethod = "close")
public BasicDataSource dataSource(){

  BasicDataSource  bds = new BasicDataSource();
  bds.setUrl( "jdbc:h2:mem");
  bds.setPassword("");
  bds.setUsername("sa");
  bds.setDriverClass( Driver.class);
  return bds;
}
```

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
public LocalContainerEntityManagerFactoryBean entityManager(){
 LocalContainerEntityManagerFactoryBean lcem =
      new LocalContainerEntityManagerFactoryBean();
 lcem.setDataSource( dataSource() );
 return lcem;
}
```

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

vmware | SPRING & CLOUD FOUNDRY 开发者大会

Building Spring Applications Targeting Cloud Foundry

EXPLICITLY BINDING TO SERVICES FROM JAVA

The Environment

- Asking Questions
  - You can introspect the environment variables (`System.getenv("..")`), or...
  - import the CloudFoundry runtime API from Java!
    - *(much simpler)*

```
<dependency>
  <groupId>org.cloudfoundry</groupId>
   <artifactId>cloudfoundry-runtime</artifactId>
   <version>0.8.2</version>
 </dependency>
```

Accessing Services from Java

```
CloudEnvironment cloudEnvironment = new CloudEnvironment();

Collection<RedisServiceInfo> serviceInfos =
    cloudEnvironment.getServiceInfos(RedisServiceInfo.class);

assert serviceInfos.size() > 0 : "there must be at least one bound Redis instance!";

RedisServiceInfo serviceInfo = serviceInfos.iterator().next(); // get the first one

RedisServiceCreator serviceCreator = new RedisServiceCreator();
RedisConnectionFactory cf = serviceCreator.createService(serviceInfo);
```

Accessing Services from Java

```
CloudEnvironment e = new CloudEnvironment();

Iterable<RdbmsServiceInfo> dsServiceInformation =
    e.getServiceInfos( RdbmsServiceInfo.class) ;

Iterable<RedisServiceInfo> redisServiceInformation =
    e.getServiceInfos( RedisServiceInfo.class) ;

Iterable<RabbitServiceInfo> rabbitServiceInformation =
    e.getServiceInfos( RabbitServiceInfo.class) ;

Iterable<MongoServiceInfo> mongoServiceInformation =
    e.getServiceInfos( MongoServiceInfo.class) ;
```

Accessing Services from Spring Java Configuration

```
@Bean public RedisConnectionFactory redis() {

    CloudEnvironment cloudEnvironment = new CloudEnvironment();
    Collection<RedisServiceInfo> serviceInfos =
        cloudEnvironment.getServiceInfos(RedisServiceInfo.class);
    assert serviceInfos.size() > 0 : "there must be at least one bound Redis instance!";
    RedisServiceInfo serviceInfo = serviceInfos.iterator().next(); // get the first one
    RedisServiceCreator serviceCreator = new RedisServiceCreator();
    RedisConnectionFactory cf = serviceCreator.createService(serviceInfo);
    return cf;
}
```

vmware | SPRING & CLOUD FOUNDRY
开发者大会

Building Spring Applications Targeting Cloud Foundry

EXPLICITLY BINDING TO SERVICES FROM XML

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

<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: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" service-name="mySQLSvc">
    <cloud:pool pool-size="1-5"/>
    <cloud:connection properties="charset=utf-8"/>
</cloud:data-source>

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

<cloud:properties>

- Exposes basic information about services that can be consumed with Spring's property placeholder support
- Basic attributes:
  - id: the name of the properties bean
- Properties automatically available when deploying Spring 3.1 applications

```
<cloud:properties id="cloudProperties" />
<context:property-placeholder properties-ref="cloudProperties"/>

@Autowired private Environment environment;


@Bean
public ComboPooledDataSource dataSource() throws Exception {
  String user = this.environment.getProperty
        ("cloud.services.mysql.connection.username");
  ComboPooledDataSource cpds  = new  ComboPooledDataSource();
  cpds.setUser(user);
  return cpds;
}
```

vmware | SPRING & CLOUD FOUNDRY
开发者大会

Building Spring Applications Targeting Cloud Foundry

THE SPRING ENVIRONMENT ABSTRACTION

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
- “default” profile automatically activates by Spring unless something else is activated
- “cloud” profile automatically activates on Cloud Foundry
  - usage of the cloud namespace should occur within the cloud profile block

Isolating Cloud Foundry Configuration

```
@Configuration
@Profile("default")
public class LocalConfiguration {
   @Bean
   public DataSource dataSource() {
      BasicDataSource bds = new BasicDataSource();
      // ...
      return bds;
   }
}
```

```
@Configuration
 @Profile("cloud")
 public class CloudConfiguration {
    private CloudEnvironment cloudEnvironment = new CloudEnvironment();

    @Bean
    public DataSource dataSource() {
       Collection<RdbmsServiceInfo> rdbmsServiceInfo = cloudEnvironment.getServiceInfos(RdbmsServiceInfo.class);
       RdbmsServiceCreator rdbmsServiceCreator = new RdbmsServiceCreator();
       return rdbmsServiceCreator.createService(rdbmsServiceInfo.iterator().next());
    }
 }
```

Isolating Cloud Foundry Configuration

```
<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <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/my_db" />
      </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

Cloud Properties Example

- Use service properties to create your own connection factories
  - e.g. c3p0 connection pool

```
import com.mchange.v2.c3p0.* ;

@Autowired
private Environment  e;

@Bean
public ComboPooledDataSource cpds (){
 ComboPooledDataSource cpds = new ComboPooledDataSource ();

 String host=e.getProperty("cloud.services.mysql.connection.host"),
       port=e.getProperty("cloud.services.mysql.connection.port"),
       name=e.getProperty("cloud.services.mysql.connection.name"),
       pw=e.getProperty("cloud.services.mysql.connection.password");

 cpds.set...( host ... );
 return cpds;
}
```

vmware | SPRING & CLOUD FOUNDRY
开发者大会

Building Spring Applications Targeting Cloud Foundry

NOSQL WITH SPRING ON CLOUD FOUNDRY

Relational databases are great...

- SQL
  - High-level
  - Sorting
  - Aggregation
- ACID semantics
- Well supported
  - JDBC
  - Hibernate/JPA
  - Spring
- Well understood
  - Developers
  - Operators

... but they have limitations

- Object/relational impedance mismatch
- Complicated to map rich domain model to relational schema
- Difficult to handle semi-structured data, e.g. varying attributes
- Schema changes
- Extremely difficult/impossible to scale
- Poor performance for some use cases

Solution: Spend Money

http://upload.wikimedia.org/wikipedia/commons/e/e5/Rising_Sun_Yacht.JPG

- Hire more DevOps
- Use application-level sharding
- Build your own middleware
- …

OR

http://www.trekbikes.com/us/en/bikes/road/race_performance/madone_5_series/madone_5_2/#

Solution: Use NoSQL

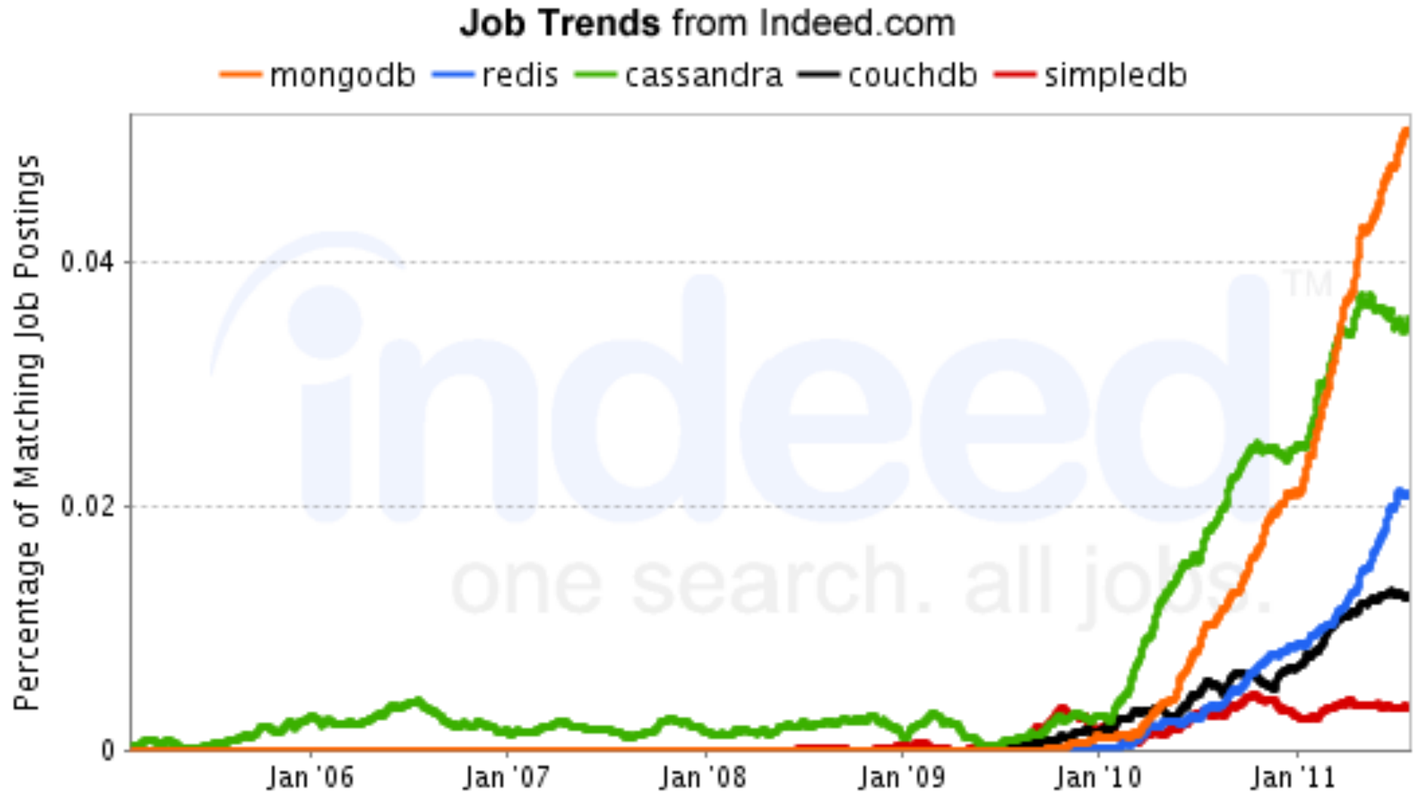
Benefits

- Higher performance
- Higher scalability
- Richer data-model
- Schema-less

Drawbacks

- Limited transactions
- Relaxed consistency
- Unconstrained data

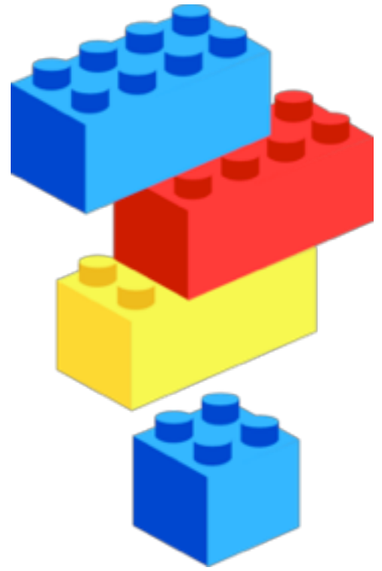
Growing in popularity…



http://www.springsource.org/spring-data

- **Spring Data Key-value**
- **Spring Data Document**
- **Spring Data Graph**
- Spring Data Column
- Spring Data Blob
- Spring Data JPA Repository / JDBC Extensions
- Spring Gemfire / Spring Hadoop ...
- Grails i**N**c**O**n**S**e**Q**uentia**L**

Spring Data Building Blocks



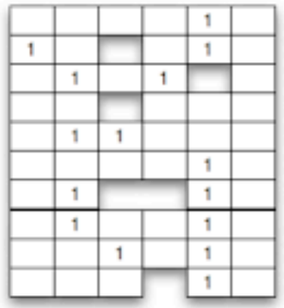
- Low level data access APIs
  - ✓MongoTemplate, RedisTemplate ...
- Object Mapping (Java and GORM)
- Cross Store Persistence Programming model
- Generic Repository support
- Productivity support in Roo and Grails

NoSQL with Redis

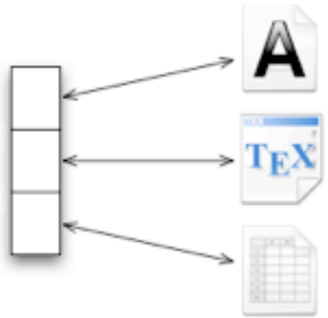
NoSQL offers several data store categories

Key-Value

Column



Document



Graph

Redis,
Riak

Spring Data Redis

- **Works with Redis**
  - super fast
  - “data structure server” (maps, lists, sets, queue, etc.)
- **`RedisTemplate` reduces tedious boilerplate code to one liners**
- **`CacheManager` implementation**
  - works with Spring 3.1’s `CacheManager` implementation
- **`RedisMessageListenerContainer`**
  - provides same queue / publish-subscribe features as with JMS and AMQP

Configuring Redis support

```
@Bean
 public RedisConnectionFactory redisConnectionFactory() {
    return new JedisConnectionFactory();
 }

 @Bean
 public RedisTemplate<String, Object> redisTemplate()
                throws Exception {
    RedisTemplate<String, Object> ro = new RedisTemplate<String, Object>();
    ro.setConnectionFactory(redisConnectionFactory());
    return ro;
}
```

Handling Persistence Duties with Redis...

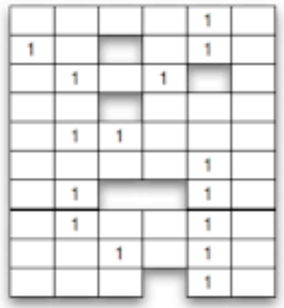
```
@Inject RedisTemplate redisTemplate;

@Override
public Customer getCustomerById(long id) {
   String ln = (String)redisTemplate.opsForValue().get(lastNameKey(id)) ;
   String fn = (String)redisTemplate.opsForValue().get(firstNameKey(id));
   return new Customer(id, fn, ln);
}

private void setCustomerValues(long lid, String fn, String ln) {
   this.redisTemplate.opsForValue().set(lastNameKey(lid), ln);
   this.redisTemplate.opsForValue().set(firstNameKey(lid), fn);
}

@Override
public Customer updateCustomer(long id, String fn, String ln) {
   setCustomerValues(id, fn, ln);
   return getCustomerById(id);
}
```

NoSQL with MongoDB

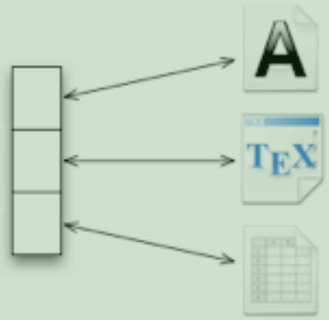
NoSQL offers several data store categories

Key-Value

Column



Document



Graph

MongoDB

\

Spring Data MongoDB

- **works with MongoDB**
- **provides obvious API integrations:**
  - MongoTemplate
  - Mongo-backed `MessageStore` (Spring Integration)
- **Advanced API integrations, too:**
  - cross-store persistence
  - MongoRepository (built on Spring Data JPA)

Simple Domain Class

```
public class Person {

  private String id;
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
  public int getAge() {
    return age;
  }
```

Mongo Template

Direct Usage of the Mongo Template:

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");

    mongoOps.insert(new Person("Joe", 34));

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

Mongo Template

Direct Usage of the Mongo Template:

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");

    mongoOps.insert(new Person("Joe", 34));

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

Insert into “Person” Collection

Mongo Template

Direct Usage of the Mongo Template:

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOpera                                          go(), "database");

    mongoOps.i

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

findOne using query: { "name" : "Joe"}
in db.collection: database.Person

Mongo Template

Direct Usage of the Mongo Template:

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");

    mongoOp

    log.inf                                                    s("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

Dropped collection [database.person]

JPA and MongoDB can be used in cross-store persistence

JPA “Customer” with a “SurveyInfo” Document

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String firstName;

  private String lastName;

  @RelatedDocument
  private SurveyInfo surveyInfo;

  // getters and setters omitted

}
```

Using a Cross-Store

Saving a Customer with a SurveryInfo

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestionAndAnswer("age", "22")
  .addQuestionAndAnswer("married", "Yes")
  .addQuestionAndAnswer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

Using a Cross-Store

Saving a Customer with a SurveryInf

Create Customer

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestionAndAnswer("age", "22")
  .addQuestionAndAnswer("married", "Yes")
  .addQuestionAndAnswer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

Using a Cross-Store

Saving a Customer with a SurveryInfo

```
Customer customer = new
customer.setFirstName("S
customer.setLastName("Ol
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestionAndAnswer("age", "22")
  .addQuestionAndAnswer("married", "Yes")
  .addQuestionAndAnswer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

Create SurveyInfo

Using a Cross-Store

Saving a Customer with a SurveryInfo

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyIn                                    ()
   .addQue
   .addQue                                  es")
   .addQuestionA      swer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

Assign Survey to Customer

Using a Cross-Store

Saving a Customer with a SurveryInfo

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestion
  .addQuestion                                    s")
  .addQuestion                                     "Norwegian");
customer.setSu                                    );
customerRepository.save(customer);
```

Save

Using a Cross-Store

Saving a Customer with a SurveryInfo

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestion
  .addQuestion                                    s")
  .addQuestion                                    "Norwegian");
customer.setSu
customerRepository.save(customer);
```

Save

Mongo Document:

```
{ "_id" : ObjectId( "4d9e8b6e3c55287f87d4b79e" ),
  "_entity_id" : 1,
  "_entity_class" : "org.springframework.data.mongodb.examples.custsvc.domain.Customer",
  "_entity_field_name" : "surveyInfo",
  "questionsAndAnswers" : { "married" : "Yes",
    "age" : "22",
    "citizenship" : "Norwegian" },
  "_entity_field_class" : "org.springframework.data.mongodb.examples.custsvc.domain.SurveyInfo" }
```

vmware | SPRING & CLOUD FOUNDRY 开发者大会

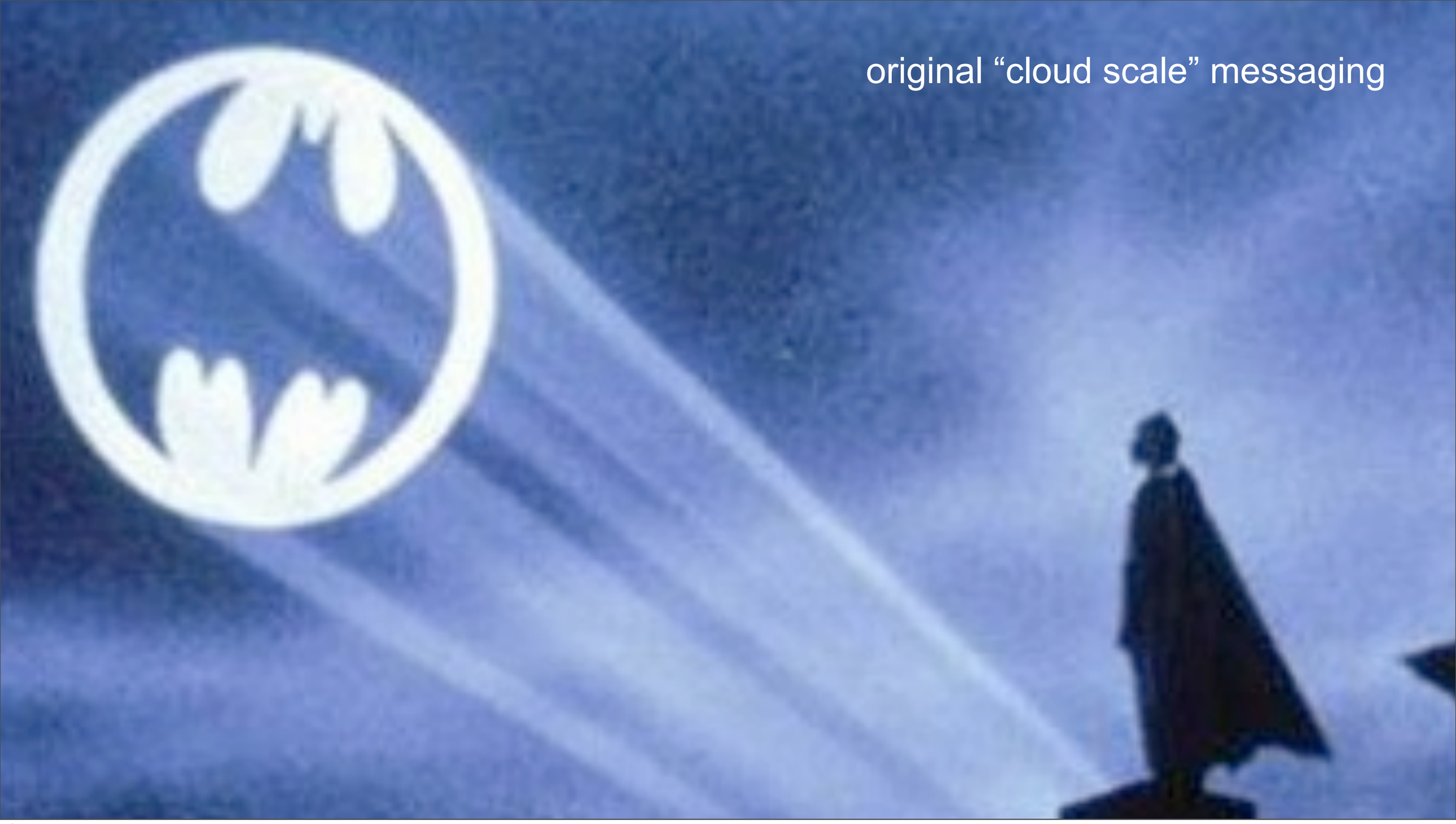
Building Spring Applications Targeting Cloud Foundry

MESSAGING WITH RABBITMQ ON CLOUD FOUNDRY

jlong — bash — 100×21

```
Password: *******
Successfully logged into [http://api.cloudfoundry.com]

jlongmbp17:~ jlong$ vmc services

============== System Services ==============

+------------+---------+-----------------------------------------+
| Service    | Version | Description                             |
+------------+---------+-----------------------------------------+
| atmos      | 1.4.1   | Atmos object store                      |
| mongodb    | 1.8     | MongoDB NoSQL store                     |
| mysql      | 5.1     | MySQL database service                  |
| postgresq                                                      |
| rabbitmq   4       | RabbitMQ messaging service              |
| redis      2       | Redis key-value store service           |
+------------+---------+-----------------------------------------+

============ Provisioned Services ============

+---------------------+------------+
```

original “cloud scale” messaging



Messaging Use Cases

Decoupling

shopping cart sending request CC merchant

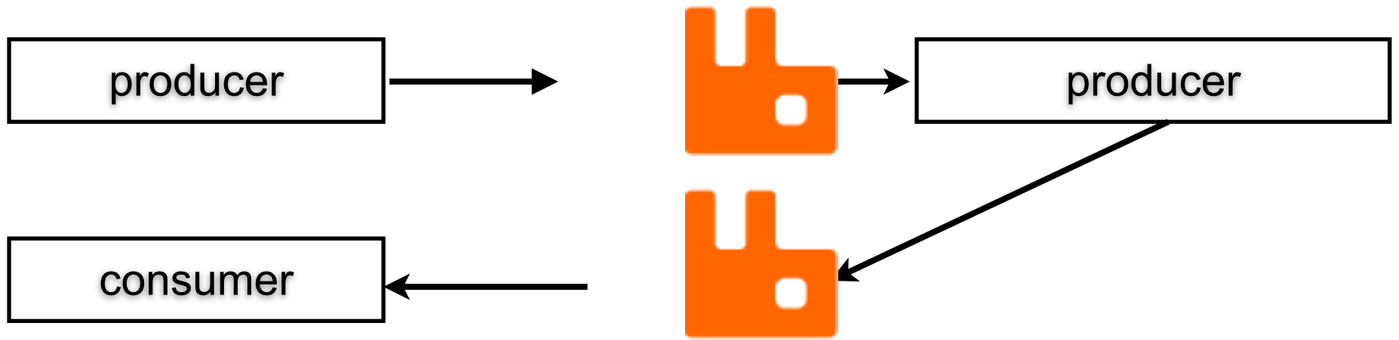
Messaging Use Cases

Bidirectional Decoupling

eg: remote procedure call

Messaging Use Cases

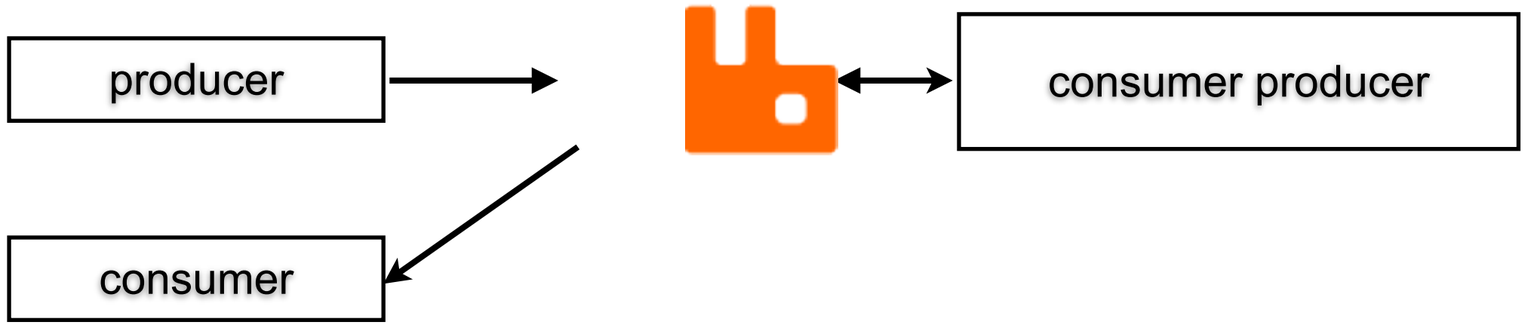
Bidirectional Decoupling



eg: place order and wait for confirmation

Messaging Use Cases

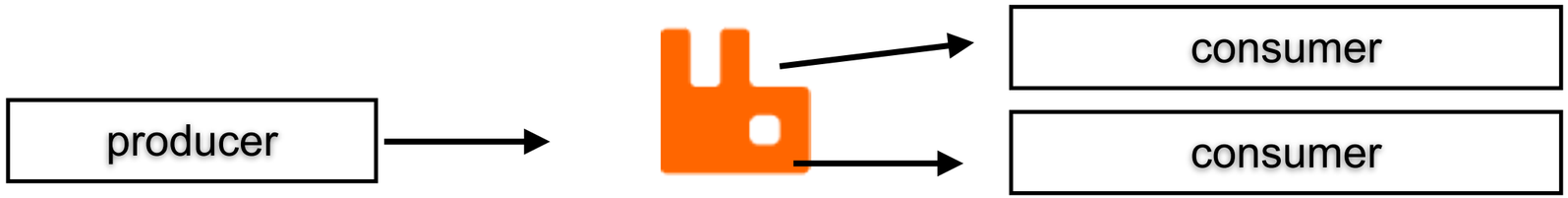
Bidirectional Decoupling



eg: place order and wait for confirmation

Messaging Use Cases

work distribution and decoupling



distribution can be duplicate or round-robin:
- duplication for information (logging, auditing, etc)
- round robin for scaling and load balancing

AMQP

- **Sending and Receiving AMQP messages**

```
@Component
public class MessageSender {

  @Autowired
  private AmqpTemplate amqpTemplate;

  public void send(String message) {
    this.amqpTemplate.convertAndSend(
          "myExchange", "some.routing.key", message);
  }

}
```

```
@Component
public class MessageReceiver {

  @Autowired
  private RabbitTemplate rabbitTemplate;

  public void read() throws Exception {
     String value = rabbitTemplate.receiveAndConvert("myQueueName");
  }

}
```

@starbuxman | josh.long@springsource.com

http://slideshare.net/joshlong

Questions?