Spring Scala

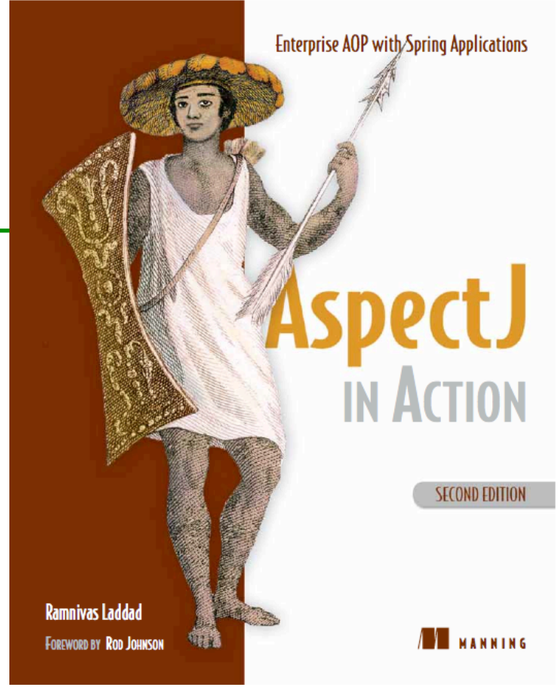
Arjen Poutsma
Ramnivas Laddad
Pivotal

About @poutsma

- Twenty years of experience in Enterprise Software Development
- Joined SpringSource in 2005
- Development lead of Spring Web Services
- Development lead on Spring 3's REST support
- Dabbled in Scala in 2008

About @ramnivas

- Author, AspectJ in Action
- Speaker at many conferences
- Led the framework integration team in CloudFoundry
- Main interests
  - Cloud computing
  - Aspect-oriented programming
  - Scala and functional programming
- Active involvement in AspectJ, Spring, and Cloud Foundry since their early form



Why Spring and Scala?

- Scala has many exciting features
  - –Pattern matching, implicits, functions, JVM, ...
- Spring is the de facto Java Enterprise framework
  - –more than DI, proven, 10+ development years, ...
- Why not combine them?

Introducing Spring Scala

- Separate Spring portfolio project
- Introduced at last year's SpringOne 2GX
- Goal: make it easier to use Spring in Scala
- Built on top of the Spring Java Framework

Spring Scala Features

- Wiring up Scala Beans in XML
  - Support for Scala Collections
- Wiring up Scala Beans in **Scala**
- Scala-friendly versions of Spring Templates

Wiring up in XML

- Constructor injection, or
- @BeanProperty, or
- Spring Scala & Spring 3.2+

Properties in Scala

```
class A(var b: String)
```

```
public class A {
 private String b;
 public A(String b) {
  this.b = b;
 }
 public String b() {
  return b;
 }
 public void b_$eq(String b) {
  this.b = b;
 }
}
```

Demo

Scala Collections

- Rich API
- Seq, IndexedSeq, LinearSeq, Buffer, Set, Map, etc.
- Mutable and Immutable version
- Spring Scala supports all of these
  - PropertyEditors
  - XML namespace

Demo

Wiring up in Scala

- FunctionalConfiguration (aka ScalaConfig)
- Singletons and prototypes
- Bean references
- Configuration Composition
- Importing XML and @Configuration
- Init and destroy methods
- Bean Profiles

```
<beans>
    <bean class="Person">
        <constructor-arg value="John"/>
        <constructor-arg value="Doe"/>
    </bean>
</beans>
```

```
class PersonConfiguration
  extends FunctionalConfiguration {
    bean() {
        new Person("John", "Doe")
    }
}
```

Demo

Spring Templates

- Consistent and convenient approach to data access
  - JdbcTemplate, JmsTemplate, ...
- Spring Scala has Scala versions of these:
  - Functions instead of callbacks
  - Option

JmsTemplate

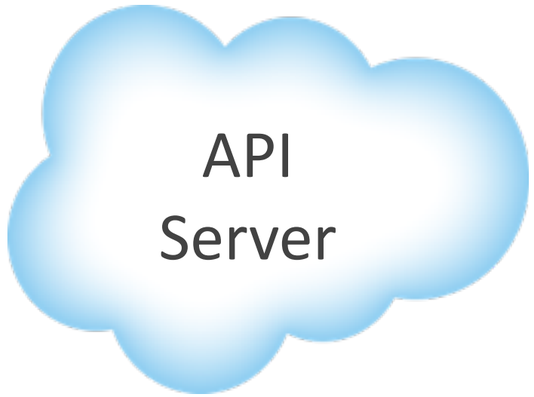
```
val connectionFactory : ConnectionFactory = ...
val template = new JmsTemplate(connectionFactory)

template.send("queue") {
    session: Session => session.
      createTextMessage("Hello World")
}

template.receive("queue") match {
    case Some(m: TextMessage)=> println(m.getText)
    case None => println("No text message received")
}
```

Demo

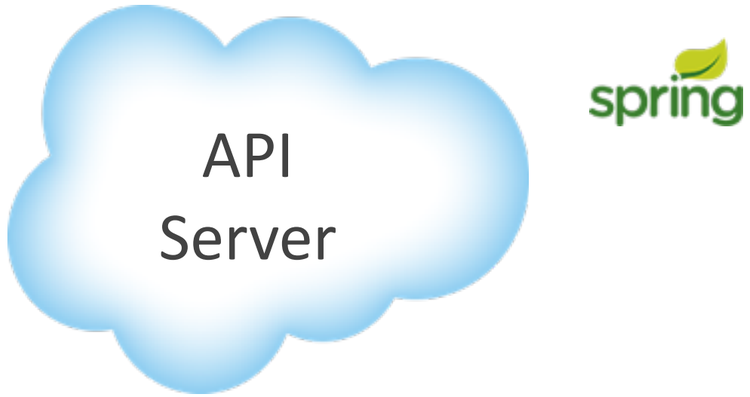
Using
Spring and Scala
in
a real-world App

Demo

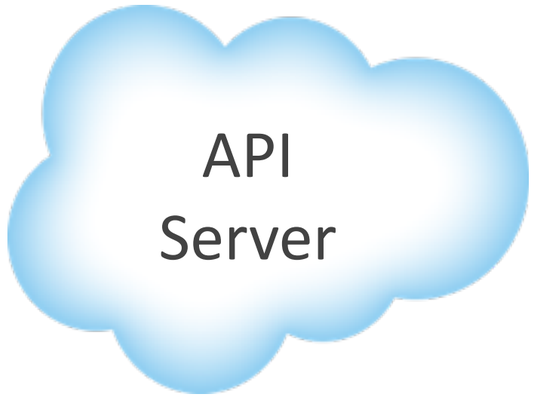
Architecture



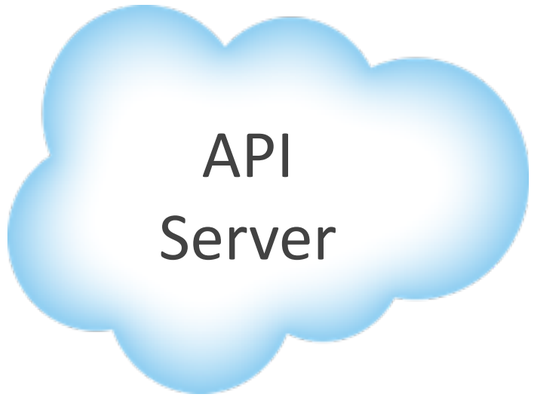
Architecture



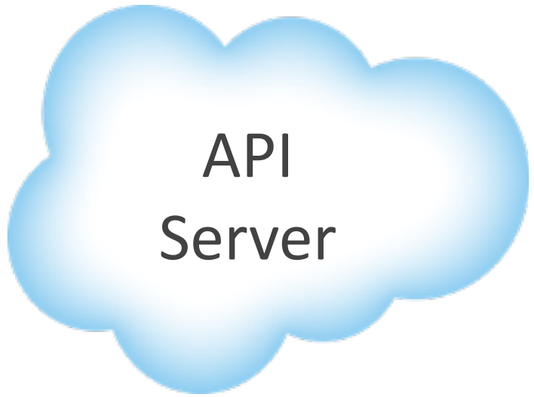
Architecture



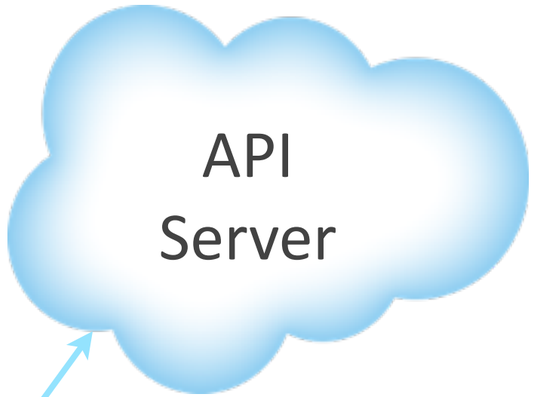
Architecture



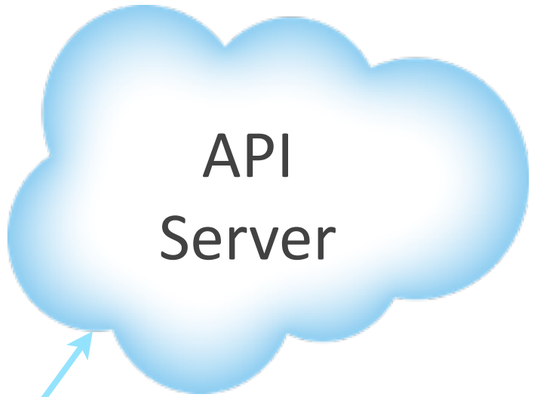
Architecture



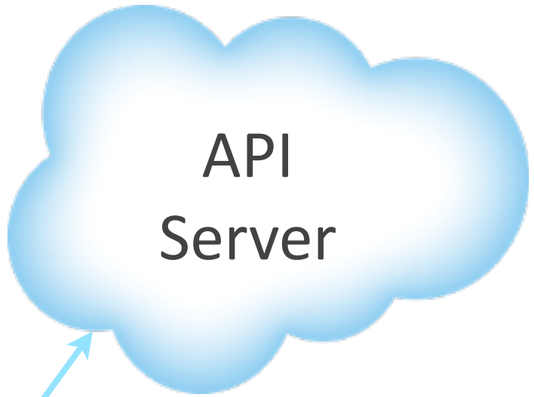
Architecture



Architecture

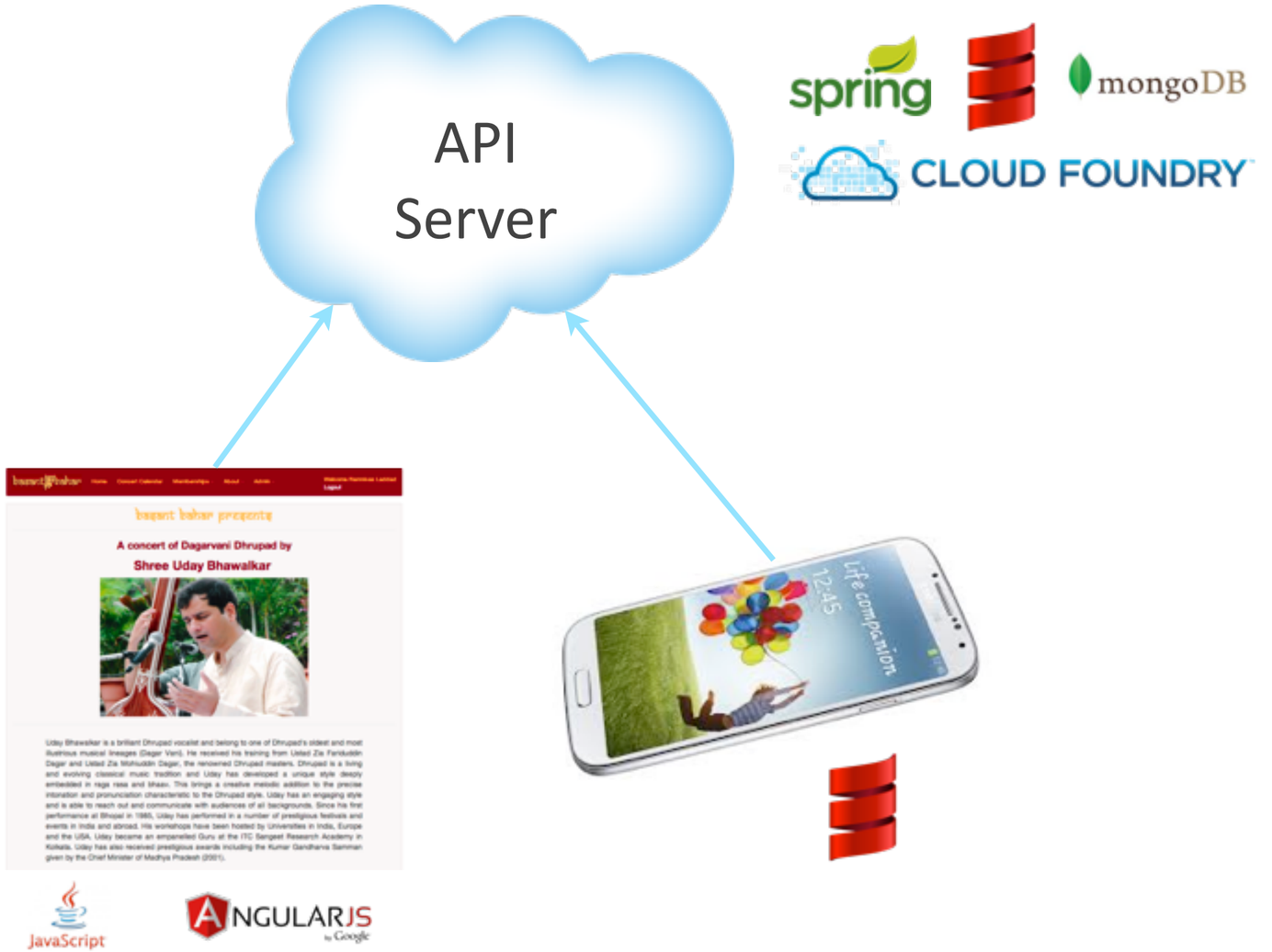


Architecture

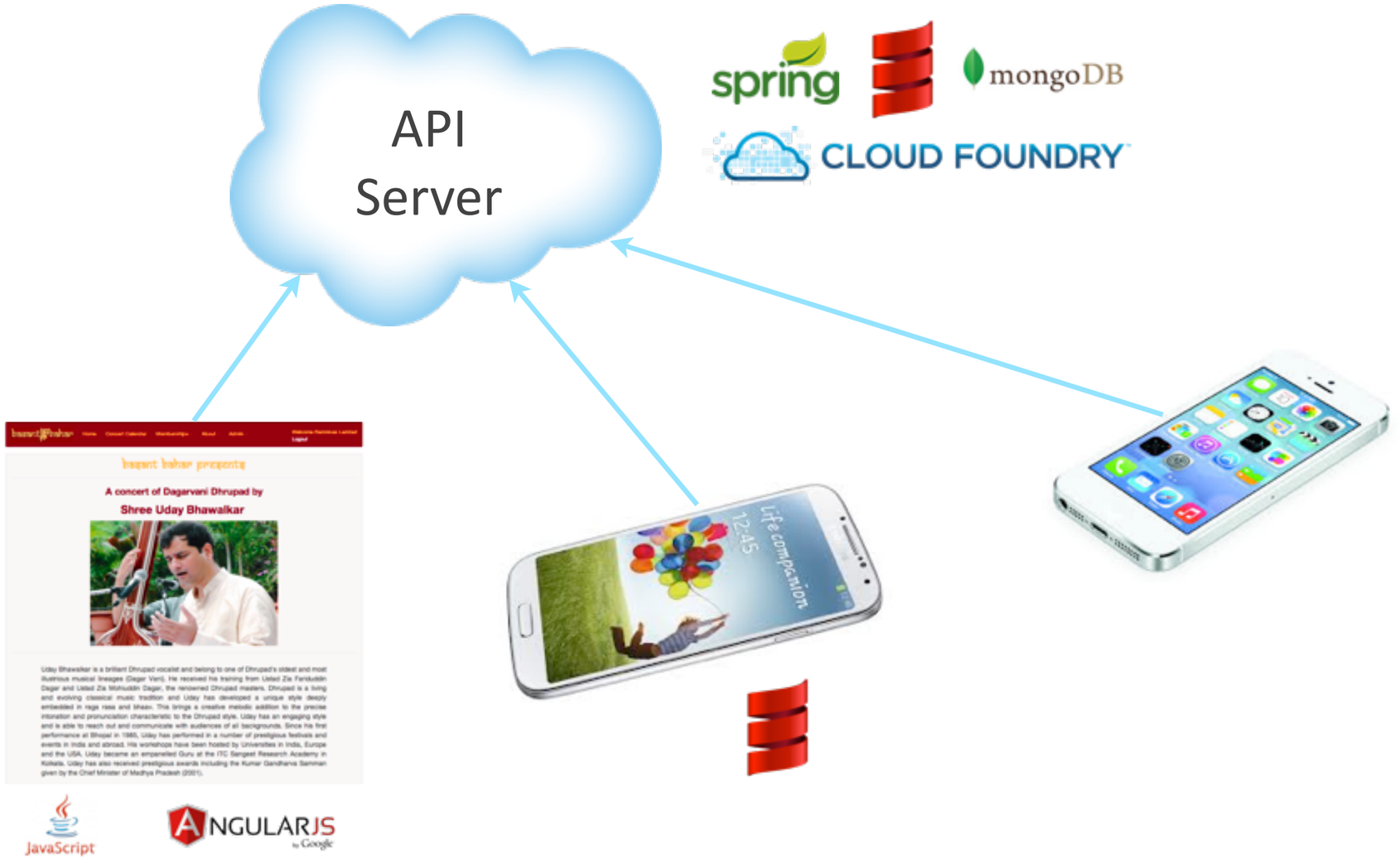


Architecture

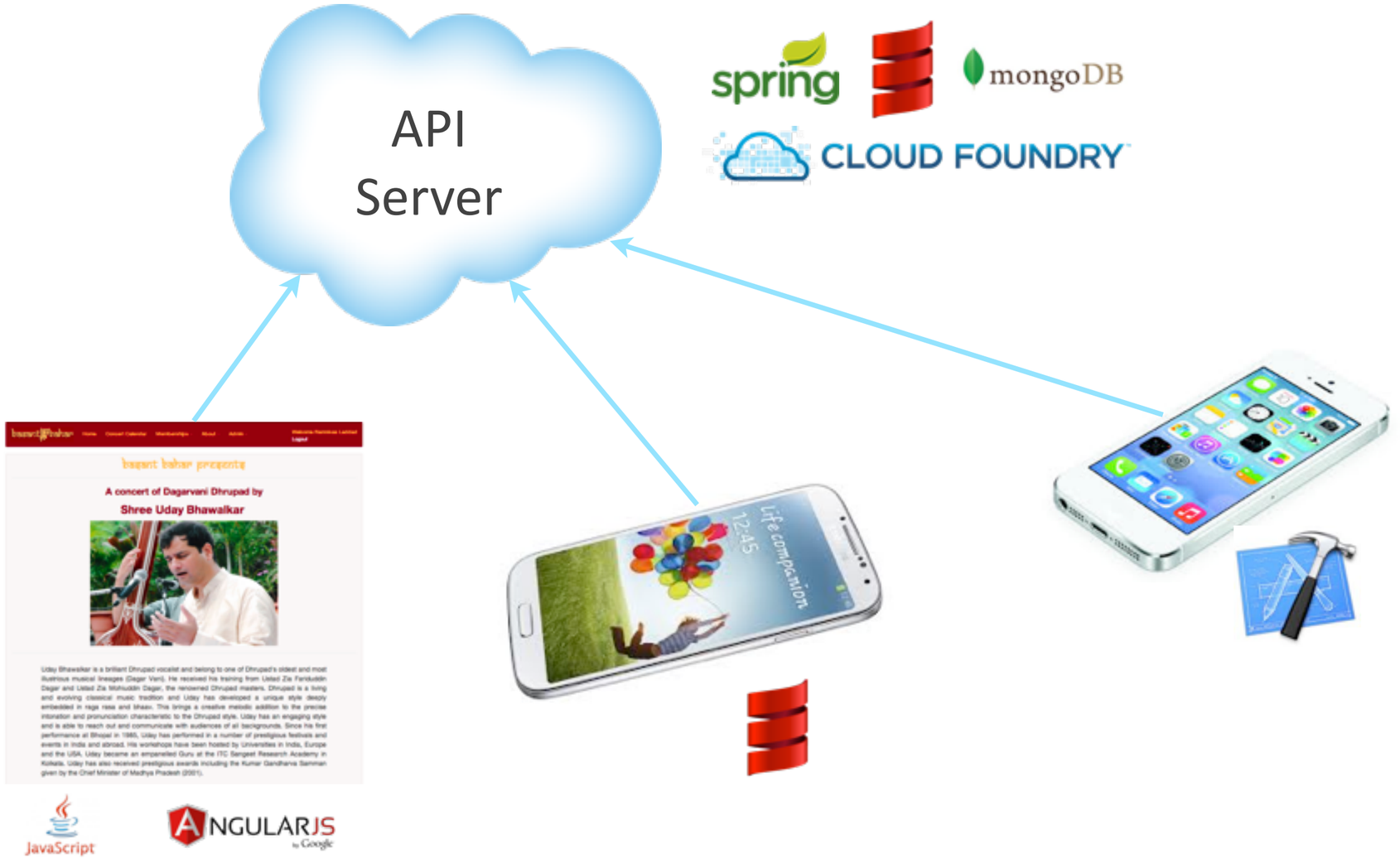
Architecture



Architecture



Architecture



Architecture

Batch Processing
(email, social,
member processing)

API
Server

spring

mongoDB

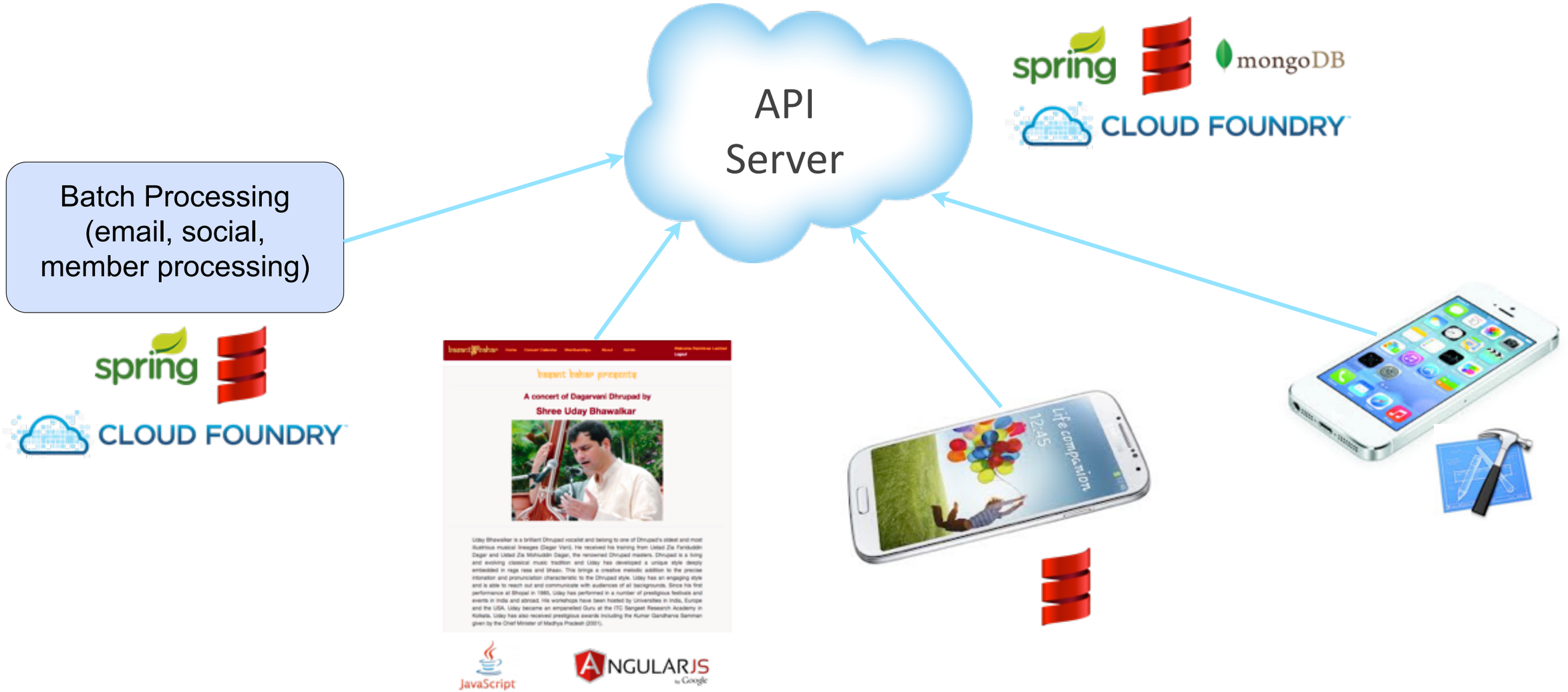
CLOUD FOUNDRY

JavaScript

ANGULARJS by Google

springone 2GX

Architecture



REST API: Generic CRUD

```
GET     /api/<entity>
GET     /api/<entity>/{id}

POST    /api/<entity>
PUT     /api/<entity>/{id}

DELETE  /api/<entity>/{id}
```

REST API: Generic CRUD Example

```
GET     /api/concerts
GET     /api/concerts/{id}

POST    /api/concerts
PUT     /api/concerts/{id}

DELETE  /api/concerts/{id}
```

REST API: Specialized end points

```
GET /api/nextConcert
GET /api/concertsThisYear
GET /api/concerts/year/{year}
```

REST API: Generic CRUD Example


```
{
  "id":"522d03c44a4e15050ac0e4b5",
  "publish":false,
  "concertArtists": [
    {
      "artistId":"522d01b84a4e15050ac0e4b3",
      "instrument":"Sitar",
      "role":"Main"
    },
    {
       "artistId":"522d02814a4e15050ac0e4b4",
       "instrument":"Tabla",
       "role":"Main"
    }
  ],
  "title":"Sitar concert",
  "description":"...",
  "photoUrl":"http://basant-bahar.cfapps.io/api/files/86fd83dc7f5e336b8b9d318335a36777.jpg",
  "startDateTime":"2013-12-22T01:00:00.000Z",
  "endDateTime":"2013-12-22T05:00:00.000Z",
  "venueId":"52226cc52a4e1e70527c55fb",
  "price":25
}
```

Entity abstraction

```
class Entity(val id: String)
```

Defining Artist

```
  import java.{util => ju}

@Document(collection="artists")
case class Artist(id: String, publish: Boolean,
                  title: String, name: String,
                  instruments: ju.ArrayList[String],
                  bio: String, photoUrl: String,
                  youtubeVideoIds: ju.ArrayList[String])
  extends Dto {

  override def toString = s"$title $name"
}
```

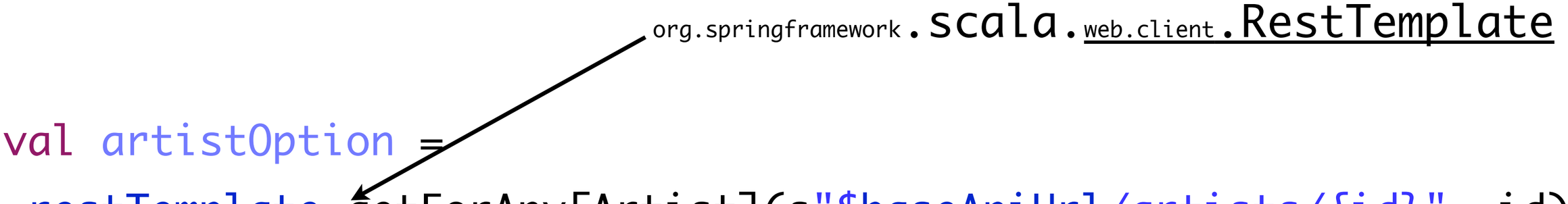
CRUD controller

```
abstract class CrudController[E <% Entity, D <% Dto]
  (service: EntityService[E],
   protected val mapper: DTOMapper[E, D]) {

@RequestMapping(value=Array("/{id}"),
        method=Array(RequestMethod.GET))
@ResponseBody
def find(@PathVariable() id: String) : D = {
  mapper.convertToDto(service.findOne(id))
}
...
```

Artist as an Entity

```
object Artist {
  implicit class ArtistEntity(artist: Artist)
      extends Entity(artist.id)
}
```

Spring Scala: RestTemplate




```
val artistOption =
 restTemplate.getForAny[Artist](s"$baseApiUrl/artists/{id}", id)

artistOption.map(...)
artistOption.foreach(...)
...
```

Scala-friendly API with RestTemplate

- Use ClassTags instead of Class<>
  - `getForAny[Artist]("...", ...)`
  - `getForObject("...", Artist.class, ...)`

- Return Option instead of Object
  - Pattern matching instead of null check

- Deals with serializing case classes, Scala collections etc.

Spring Data Repository

```
trait ConcertRepository extends MongoRepository[Concert, String] {
  def findByStartDateTimeBetween(start: Date, end: Date,
                  sort: Sort): ju.List[Concert]
}

trait ArtistRepository extends MongoRepository[Artist, String]

trait MemberRepository extends MongoRepository[Member, String]

trait VenueRepository extends MongoRepository[Venue, String]
```

Configuring Mongo

```
@Configuration
@EnableMongoRepositories(basePackages=
                Array("org.basantbahar.repository"))
class MongoConfig extends AbstractMongoConfiguration {

  override val getDatabaseName = "basant-bahar"

  override val mongo = new Mongo()

}
```

Configuring Mongo

```
@Configuration
@EnableMongoRepositories(basePackages=
                Array("org.basantbahar.repository"))
class MongoConfig extends AbstractMongoConfiguration {

  override val getDatabaseName = "basant-bahar"

  override val mongo = new Mongo()

}
```

Auto-reconfigured on Cloud Foundry

Dependency Injection

- Constructor-based... always
  - Matches Scala's preference for immutable objects
  - No need for @BeanProperty annotations
    - Spring-scala removes this need

```
class ConcertController @Autowired()
    (service: ConcertService,
     artistRepository: ArtistRepository,
     venueRepository: VenueRepository)
  extends CrudController[Concert, ConcertDto]
  (service, new ConcertDTOMapper(artistRepository, venueRepository)) {
```

Functional Configuration

```
class ApiConfig extends WebMvcConfigurerAdapter
        with FunctionalConfiguration {

  importClass(classOf[ApiContext])


  ...

  bean() {
    new CommonsMultipartResolver()
  }
}
```

Functional Configuration

```
<servlet>
  <servlet-name>api</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.scala.web.context.support.FunctionalConfigWebApplicationContext</param-value>
  </init-param>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>org.basantbahar.config.api.ApiConfig</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
```

Java’s verbosity

Java’s verbosity

```
public UserDetails loadUserDetails(OpenIDAuthenticationToken token) {
    String id = token.getIdentityUrl();

    CustomUserDetails user = registeredUsers.get(id);

    if (user != null) {
        return user;
    }

    String email = null;
    String firstName = null;
    String lastName = null;
    String fullName = null;

    List<OpenIDAttribute> attributes = token.getAttributes();

    for (OpenIDAttribute attribute : attributes) {
        if (attribute.getName().equals("email")) {
            email = attribute.getValues().get(0);
        }

        if (attribute.getName().equals("firstname")) {
            firstName = attribute.getValues().get(0);
        }

        if (attribute.getName().equals("lastname")) {
            lastName = attribute.getValues().get(0);
        }

        if (attribute.getName().equals("fullname")) {
            fullName = attribute.getValues().get(0);
        }
    }

    if (fullName == null) {
        StringBuilder fullNameBldr = new StringBuilder();

        if (firstName != null) {
            fullNameBldr.append(firstName);
        }

        if (lastName != null) {
            fullNameBldr.append(" ").append(lastName);
        }
        fullName = fullNameBldr.toString();
    }

    user = new CustomUserDetails(id, DEFAULT_AUTHORITIES);
    user.setEmail(email);
    user.setName(fullName);

    registeredUsers.put(id, user);

    user = new CustomUserDetails(id, DEFAULT_AUTHORITIES);
    user.setEmail(email);
    user.setName(fullName);
    user.setNewUser(true);

    return user;
}
```

Java's verbosity

```
def loadUserDetails(token: OpenIDAuthenticationToken) : UserDetails = {
  val id = token.getIdentityUrl();

  val user = userRepository.findOne(id)

  if (user == null) {
   val attributes = token.getAttributes().map {
     attribute => attribute.getName() -> attribute.getValues().get(0)
   }.toMap

    val email = attributes("email")
    val fullName = attributes.getOrElse("fullname", {
      val firstName = attributes("firstname")
      val lastName = attributes("lastname")
      s"$firstName $lastName"
    })

    val newUser = OpenIdUserDetails(id, email, fullName, DEFAULT_AUTHORITIES)
    userRepository.save(newUser)
    newUser
  } else {
    user
  }
}
```

```
public UserDetails loadUserDetails(OpenIDAuthenticationToken token) {
    String id = token.getIdentityUrl();

    CustomUserDetails user = registeredUsers.get(id);

    if (user != null) {
        return user;
    }

    String email = null;
    String firstName = null;
    String lastName = null;
    String fullName = null;

    List<OpenIDAttribute> attributes = token.getAttributes();

    for (OpenIDAttribute attribute : attributes) {
        if (attribute.getName().equals("email")) {
            email = attribute.getValues().get(0);
        }

        if (attribute.getName().equals("firstname")) {
            firstName = attribute.getValues().get(0);
        }

        if (attribute.getName().equals("lastname")) {
            lastName = attribute.getValues().get(0);
        }

        if (attribute.getName().equals("fullname")) {
            fullName = attribute.getValues().get(0);
        }
    }

    if (fullName == null) {
        StringBuilder fullNameBldr = new StringBuilder();

        if (firstName != null) {
            fullNameBldr.append(firstName);
        }

        if (lastName != null) {
            fullNameBldr.append(" ").append(lastName);
        }
        fullName = fullNameBldr.toString();
    }

    user = new CustomUserDetails(id, DEFAULT_AUTHORITIES);
    user.setEmail(email);
    user.setName(fullName);

    registeredUsers.put(id, user);

    user = new CustomUserDetails(id, DEFAULT_AUTHORITIES);
    user.setEmail(email);
    user.setName(fullName);
    user.setNewUser(true);

    return user;
}
```

What's next

- Web app
  - –Authentication through OAuth
  - –Batch processing for reminders, membership management
    - RestTemplate-based client
  - –PoC using Scala.js with Angular
- Android
  - –Scala-based native app
- iOS
  - –Native app

Further information

- Github repo opened October 2012
- First milestone released December 2012
- Second milestone released April 2013
- First release candidate release September 2013
- 1.0 GA “soon”
- Community-driven

https://github.com/SpringSource/spring-scala