Spring Boot - I

DEVELOPING APPLICATION WITH SPRING BOOT

INTRODUCTION TO SPRING BOOT

- Create stand-alone Spring applications
  - Embed Tomcat or Jetty directly
- Provide opinionated 'starter' POMs
  - simplify your Maven configuration
- Automatically configure Spring whenever possible
- Provide production-ready features
  - metrics, health checks and externalized configuration

No code generation and **no requirement for XML** configuration

DEMO: START A PROJECT

POM Configuration

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.0.2.RELEASE</version>
</parent>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
```

Standard-alone Application

```
@ComponentScan
@EnableAutoConfiguration
public class Application {
    public static void main(String... args) {
        SpringApplication.run(Application.class, args);
    }
}
```

DEMO: DEVELOP THE BUSINESS

Domain

```
public class Contact {
    private String id;
    private String name;
    private String fullName;
    private String jobTitle;
    private String email;
    private String mobile;
    private String skypeId;
```

Service

```
@Component
public class ContactService {
    public long loadContacts(String filePath) throws IOE
    public List<Contact> searchContacts(String keyword,
    public Contact getContact(String id) {
    public Contact saveContact(Contact contact) {
    public void deleteContacts(String... ids) {
    public void deleteAllContacts() {
}
```

Unit-test for Service

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(classes = Application.class)
public class ContactServiceTest {
    private ContactService contactService;
    public void startUp() throws IOException {
    public void tearDown() {

    @Test
    public void testSearchContacts() {
        contacts = contactService.searchContacts("", 0, 10);
        assertThat(contacts.size(), is(equalTo(5)));
```

DEMO: PERSIST DATA

POM Configuration

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

JPA Entity

```
@Entity
@Table(name = "CONTACTS")
public class Contact {
    @Id
    @Column(name = "ID")
    private String id;

    @Column(name = "NAME")
    private String name;

    @Column(name = "FULL_NAME")
    private String fullName;
```

Spring Data JPA Repository

```
public interface ContactRepository extends JpaRepository<Contact, String> {
    @Query("select c from Contact c where lower(c.id) like :keyword% order by c.name")
    List<Contact> searchContacts(@Param("keyword") String keyword, Pageable pageable);

    @Modifying
    @Query("delete from Contact where id in (:ids)")
    void deleteContacts(@Param("ids") String... ids);
```

DEMO: MAKE A REST SERVICE

POM Configuration

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

REST Controller

```
@RestController
@RequestMapping(value = "/rest/contacts")
public class ContactController {
    @Autowired
    private ContactService contactService;

    @RequestMapping(method = GET)
    @ResponseBody
    public List<Contact> searchContacts(
            @RequestParam String keyword,
            @RequestParam int page,
            @RequestParam int pageSize) {
        return contactService.searchContacts(keyword, page, pageSize);
    }
```

DEMO: VALIDATE THE INPUTS

POM Configuration

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
</dependency>
```

JPA Entity Validation

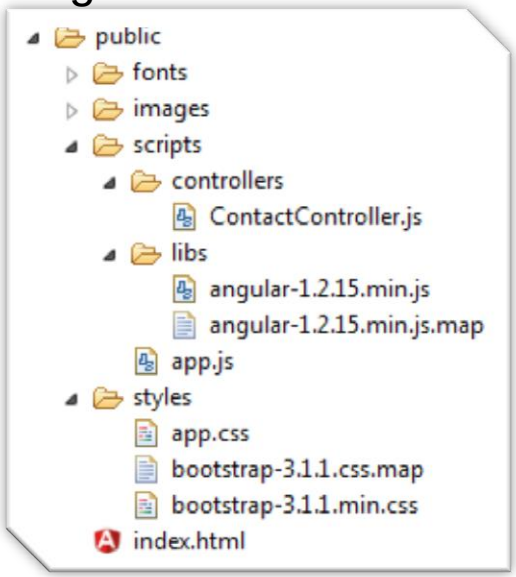
```
@Column(name = "EMAIL")
@Email(message = "{validation.email.message}")
private String email;

@Column(name = "MOBILE")
@Pattern(regexp = "\\d{3,4} \\d{3} \\d{4}", message = "{vali
private String mobile;

@Column(name = "SKYPE_ID")
@Pattern(regexp = "[A-Za-z0-9_,\\-\\.]{6,32}", message = "{v
eId;
```

REST Controller

```
@RequestMapping(value = "/{id}", method = PUT)
@ResponseStatus(HttpStatus.ACCEPTED)
public Contact updateContact(
        @PathVariable("id") String id,
        @RequestBody @Valid Contact contact) {
    contact.setId(id);
    return contactService.saveContact(contact);
}
```

DEMO: VIEW BY ANGULARJS

AngularJS Structure



AngularJS Controller

```
var PAGE_SIZE = 24;
$scope.page = page;
$scope.isLoading = true;
$http.get('rest/contacts?keyword=' + $scope.keyword + '&page='
     .success(function(items) {
         $scope.hasMoreContacts = (items.length >= PAGE_SIZE);
         for (var i=0; i<items.length; i++) {
             $scope.contacts.push(items[i]);
         };
     });
```

AngularJS View

```
<div class="col-xs-12 col-sm-6 col-md-4 col-lg-3 card-container"
    ng-repeat="contact in contacts" ng-show="([contact]|filter:keyword).length
    <table class="table">
      <tr>
        <td class="col-xs-1" rowspan="4">
          <img alt="contact-photo" ng-src="rest/photos/{{contact.id}}" class='
        </td>
      </tr>
      <tr ng-class="{'has-error': contact.errors.fullName}">
        <td>
          <input required ng-model="contact.fullName" placeholder="Full Name"
```

DEMO: UPLOAD A PHOTO

POM Configuration

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.1</version>
</dependency>
```

App Configuration

```
contacts:
  photo:
    storage: etc/photos
    resize-width: 150
    resize-height: 150
```

REST Controller

```
public void uploadPhoto(@PathVariable String contactId, HttpServletRequest req)
    MultipartHttpServletRequest request = resolver.resolveMultipart(req);

    MultipartFile file = request.getFile("file");
    File uploadFile = File.createTempFile("contact-", contactId);
    file.transferTo(uploadFile);
```

AngularJS Controller

```
$http.post('rest/photos/' + contactId, formData, {
        headers: { 'Content-Type': undefined },
        transformRequest: angular.identity,
      })
      .success(function() {
          var img = document.getElementById('img' + contactId);
          img.src = 'rest/photos/' + contactId + '?' + new Date().getTime();
    });
```

DEMO: MORE SECURE

POM Configuration

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

App Configuration

```
security:
  user:
    name: admin
    password: admin123
  basic:
    realm: Please provide your a
```

Security Configuration

```
http.httpBasic().and()
    .csrf().disable()
    .authorizeRequests()
    .antMatchers(HttpMethod.GET, "/rest/contacts*/**").permitAll()
    .antMatchers(HttpMethod.GET, "/rest/photos*/**").permitAll()
    .antMatchers("/rest/**").authenticated()
    .anyRequest().permitAll();
```

AngularJS Controller

```
$scope.switchToMode = function(mode) {
    if (mode === '/edit') {
        $http.get('rest/security/user')
            .success(function(user) {
                $scope.editMode = user;
                $location.path($scope.editMode? '/edit'
            });
    } else {
        $scope.editMode = false;
        $location.path('/view');
    }
`;
```

DEMO: PRODUCTION-READY

App Configuration

```yaml
management:
  port: 8081
  address: 127.0.0.1
  context-path: /manage
  security:
    enabled: true
    role: SUPERUSER
```

Trace

```
@RequestMapping(method = GET)
public List<Contact> searchContacts(
        @RequestParam String keyword,
        @RequestParam int page,
        @RequestParam int pageSize) {
    traceAction("search contacts, keyword:
    return contactService.searchContacts(k
}
```

```
@Override
public Map<String, String> health() {
    Map<String, String> health = new LinkedHas

    if (this.dataSource == null) {
        health.put("DB connection", "RED");
        health.put("Error", "No DataSource");
        return health;
    }
```

Health

```yaml
shell:
  ssh:
    enabled: true
    port: 2222
  auth: spring
```

Remoting

```
$ ssh -p 2222 admin@localhost
admin@localhost's password:
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::  (v1.0.2.RELEASE) on H6MMVW1

> contacts
clear  load
```

Gauge & Counter

```
StopWatch watch = new StopWatch();
watch.start();
List<Contact> contacts = contactRepo.searchContact
watch.stop();
gaugeService.submit("query.by.keyword." + keyword,
counterService.increment("search.by.keyword." + ke
```

DEMO: DEPLOY THE APPLICATION

‘Fat’ JAR file Deployment

```
example.jar
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
 |           +-<spring boot
 +-com
 |  +-mycompany
 |     + project
 |        +-YouClasses.class
 +-lib
    +-dependency1.jar
    +-dependency2.jar
```

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

WAR file Deployment

```
<!-- ... -->
<packaging>war</packaging>

<!-- ... -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

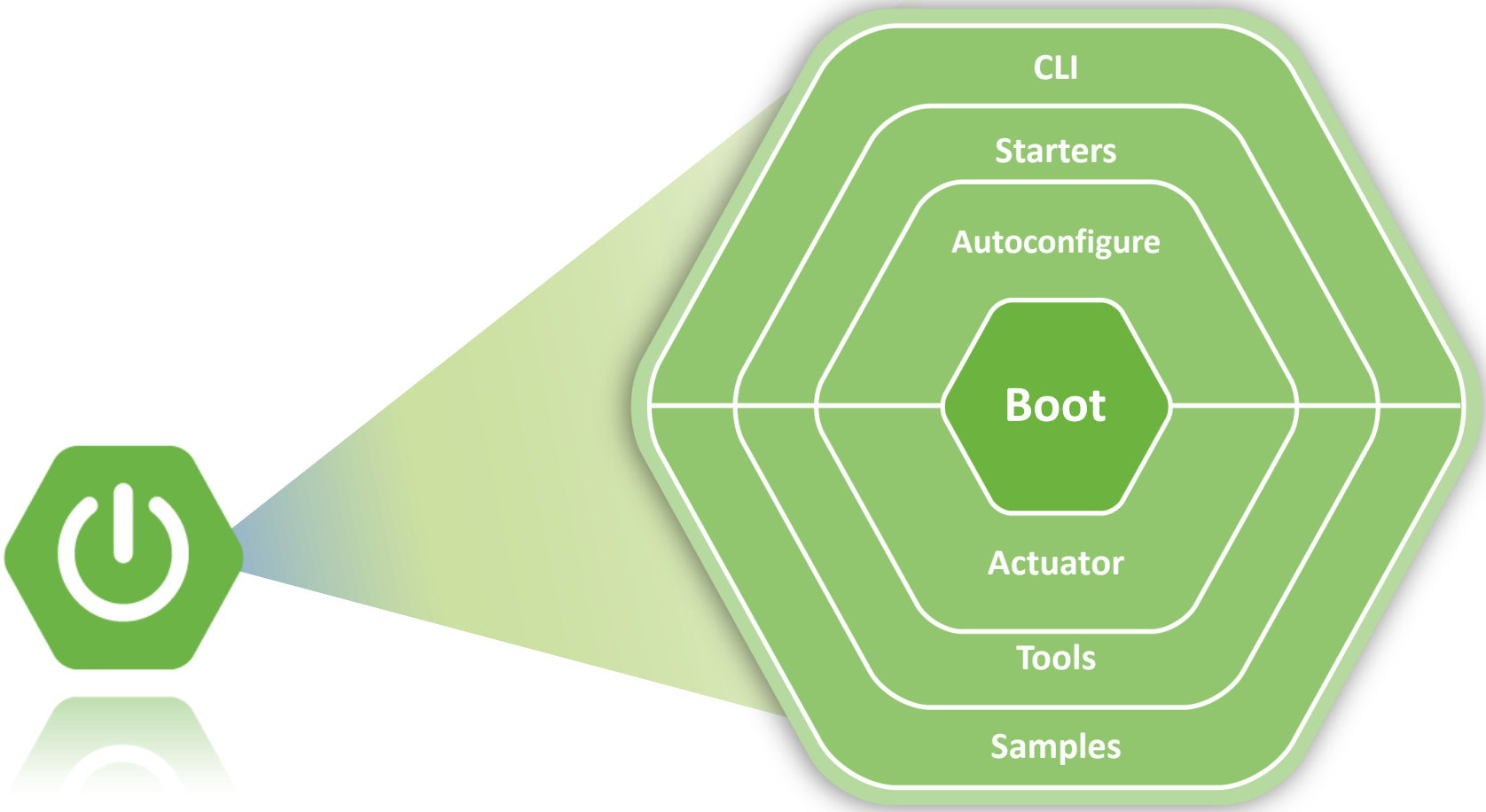
```
@ComponentScan
@EnableAutoConfiguration
@EnableTransactionManagement
@Configuration
public class Application extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(Application.class);
    }
```

@SpringBootApplication

`@SpringBootApplication` is a convenience annotation that adds all of the following:

- `@Configuration` tags the class as a source of bean definitions for the application context.
- `@EnableAutoConfiguration` tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.
- Normally you would add `@EnableWebMvc` for a Spring MVC app, but Spring Boot adds it automatically when it sees **spring-webmvc** on the classpath. This flags the application as a web application and activates key behaviors such as setting up a `DispatcherServlet`.
- `@ComponentScan` tells Spring to look for other components, configurations, and services in the `hello` package, allowing it to find the controllers.

Spring Boot Modules



Binding to Command Line Arguments

- SpringApplication binds its own bean properties to command line arguments, and then adds them to the Spring Environment, e.g.

  $ java -jar target/*.jar --server.port=9000

- Externalize Config
  - Just put application.properties in your classpath, e.g.

  application.properties

  server.port: 9000

- Use YAML (if you must)
  - Just put application.yml in your classpath

- Both properties and YAML add entries with period-separated paths to the Spring Environment.

The Actuator

- Adds common non-functional features to your application and exposes MVC endpoints to interact with them.
- Security
  Secure endpoints: /metrics, /health, /trace, /dump,
- /shutdown, /beans Audit
  /info
- If embedded in a web app or web service can use the same port or a different one (and a different network interface).

THANK YOU