Pivotal Application Architect (Java)
Certification Study Guide

Pivotal

Overview

This guide is designed to assist you in preparing for the *Pivotal Application Architect (Java)* exam.

Logistics

Pivotal partners with Exams Local to remotely proctor the exam. You may take the exam from any location that meets the basic system and test environment requirements. A valid form of photo identification is required for the exam.

For questions regarding your exam purchase, registration process, or anything else related to our certification program, please contact us at education@pivotal.io.

The exam

The *Pivotal Application Architect (Java)* exam is a four-hour performance-based exam with two sections.

The first section contains an exercise in replatforming an application. This section of the exam uses a Linux virtual machine that has all necessary software pre-installed, including the IntelliJ IDE. We provide checkpoints which will verify your deployed software meets the requirements to pass this section. The final examination for this section is pass/fail.

Candidates' internet access is limited during the exam in order to preserve the integrity of the certification.Candidates' access is limited to the following documentation sites.

- Pivotal documentation
- Spring documentation
- Maven repository search
- Tomee documentation
- Struts documentation

Topics

Pivotal Cloud Foundry

- Use the CF CLI.
- Deploy an application.
- View application logs.
- Bind a service to an application.
- Configure routing.

Replatforming

- Use a custom buildpack
- Use a built-in buildpack
- Introduce Spring Boot
- Replatform a Struts application.

Replatforming

We measure candidate's ability to replatform a Struts application to use Spring Boot and to deploy to Pivotal Cloud Foundry. We recommend practicing by replatforming actual Struts applications. The Struts Examples repository contains some sample Struts applications.

In particular, replatform the Struts CRUD app to practice the following actions.

- Use the TomEE buildpack to deploy a war file to Pivotal Cloud Foundry.
- Remove dependencies on the TomEE application server and introduce Spring Boot.
- Add REST endpoints to a Spring Boot application.

Use the outline below:

- Change the pom file to introduce Spring Boot and the MySQL driver.
- A note about versioning
    - If using Spring Boot 1.5.x then the versions in the pom.xml are correct.
    - If using Spring Boot 2.0.x then:
        - Upgrade the org.apache.struts:struts2-core to version 2.5.18.
        - Upgrade the mysql:mysql-connector-java to version 8.0.13.
    - Avoid using Spring Boot 2.1.x at this time since Struts has dependencies that are in conflict with Spring Boot 2.1.x on Pivotal Web Services.
- Create a FilterRegistrationBean for each Filter in the web.xml.
  When creating the FilterRegistrationBean for the StrutsPrepareAndExecuteFilter, configure the URL patterns must only handle the Struts actions.
- On PWS, add a database service since the TomEE buildpack defaults to an in-memory database.
- To replace the previous TomEE custom buildpack you must explicitly specify the Java buildpack.
- Verify the application still functions.

- Add Spring Boot Actuators for operations support.
  You can search the internet for examples with more details and shown below are the steps for adding actuators for Spring Boot 1.x or Spring Boot 2.0.x.

    - Spring Boot 1.X
        - Add org.springframework.boot:spring-boot-starter-actuator and org.springframework.boot:spring-boot-starter-security to your pom.xml
        - Add the @EnableWebSecurity to your Application class.

- Enable security for the Spring Boot Actuators by defining the security parameters.
- Verify the Struts CRUD application still functions as well as the Spring Boot Actuators

- Spring Boot 2.0.X
  - Add org.springframework.boot:spring-boot-starter-actuator, and org.springframework.boot:spring-boot-starter-security to your pom.xml
  - Add the @EnableWebSecurity to your Application class.
  - Add a WebSecurityConfigurationAdapter.

```
@Configuration
@Order(1)
public class ActuatorWebSecurityConfigurationAdapter extends
WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
            .cors()
            .and()
            .csrf()
            .disable()
            .authorizeRequests()
            .requestMatchers(EndpointRequest.to("info", "health")).permitAll()
            .requestMatchers(EndpointRequest.toAnyEndpoint()).authenticated()
            .and()
            .httpBasic()
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .sessionAuthenticationStrategy(new NullAuthenticatedSessionStrategy())
      ;
   }
}
```

  - Enable security for the Spring Boot Actuators by defining the security parameters.

```
(application.yml additions)

spring:
```

```
  security:
    user:
      name: actuatorUser
      password: actuatorUserPassword

management:
  endpoints:
    web:
      exposure:
        include: "*"
```

- Verify the Struts CRUD application still functions as well as the Spring Boot Actuators

Modernization

This section of the exam introduces the candidate to several modernization scenarios, and asks the candidate to answer multiple choice questions about these scenarios. The goal is to measure understanding of different systems, common problems, and recommend solutions.

Pivotal's goal during modernization engagements is to make testable changes to an application (or a vertical slice of an app) so that it runs well on PCF. This may mean decomposing a monolith into a set of microservices aligned with their bounded contexts.

The following scenario is similar to what candidates see during the certification.

Scenario

Showbook is a company that has a social network platform that integrates with other social media outlets. The current version of their platform is a web application deployed as a monolith. Showbook's goal is to modernize this application so it is easy to add new features. The business wants to introduce a mobile app as well. Showbook is seeing major slowdowns within the app. Most of the traffic (95%) is users' likes, and the system is slow to respond to other traffic. Showbook has tried to scale out more instances of their application to handle the increase traffic which has had success but is starting to cost the company extra money for running such large instances.

How should we proceed to best meet our client's business goals?

A) Introduce CQRS.
B) Split off likes into its own microservice.
C) Introduce Kafka for message and move to an event based system.

Answer: B, It allows us to seperate the likes server. We can scale that based off of usage while maintaining the synchronous behaviour the users want and lower the size / cost of VMs for Showbook.

Introducing CQRS would force us to rewrite the whole system and does not give us separate scalability which we need.

Using Kafka could cut the load by allowing likes to happen asynchronously but our users want to see what they have liked instantly and it doesn't make sense in this case.

Resources

- Spring Boot documentation
- Pivotal Cloud Foundry application deployment documentation
- Spring documentation
- Pivotal Cloud Foundry documentation
- Twelve-factor applications
- Deploying in tomee
- Spring Struts integration
- Service Discovery Guide
- Circuit Breaker Guide
- Spring Cloud Config
- Configuration Server Guide
- Spring Session
- RabbitMQ Guide
- Redis
- MySQL Guide
- Spring Boot Actuator Guide
- Spring Boot Security
- CI / CD
- Bounded Context
- The Application Continuum

Supporting course

Pivotal Platform Acceleration Lab for Application Architects

Contact

Thank you again for choosing Pivotal as your education partner and good luck with your projects. If you have encountered any errors, or have any suggestions or inquiries please contact education@pivotal.io.

Pivotal