001B: ♦ Java architecture & design concepts interview questions & answers

Posted on November 11, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Continuation of Java architecture interview Q&A with integration styles & architecture diagrams – part 1

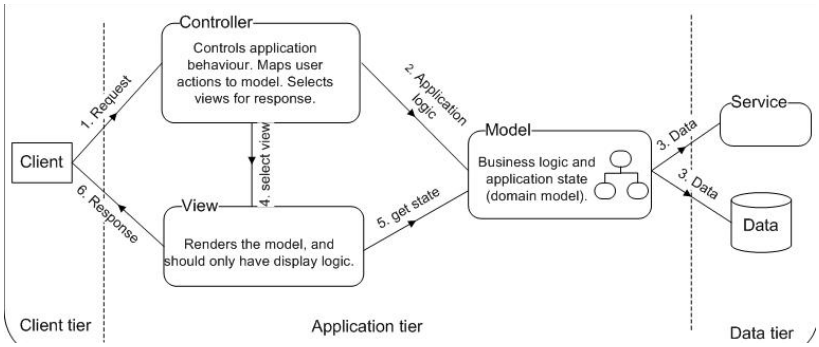
Q3. Can you discuss some of the high level architectures you are experienced with?

A3. Be prepared for a white board session on architectures, especially the bird's eye view of the last application you had worked on. There will be lots of follow on questions like why a

particular approach was used?, what are the benefits and drawbacks of a particular approach?, etc.

1. Model-View-Controller Architecture

Most web and stand-alone GUI applications follow this pattern. For example, Struts, Spring MVC are server side MVC and angularjs, react.js are client side MVC. The model represents the core business logic and state. The view renders the content of the model state by adding display logic. The controller translates the interaction with the view into action to be performed by the model. The actions performed by a model include executing the business logic and changing the state of the model. Based on the user interactions, the controller selects an appropriate view to render. The controller decouples the model from the view.



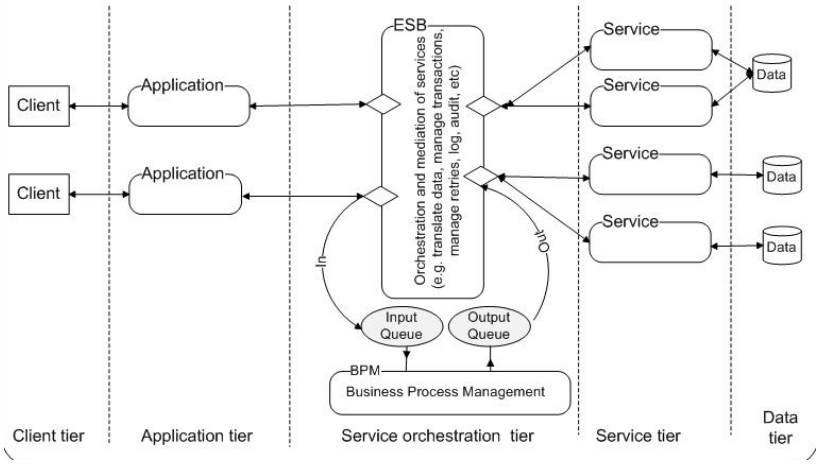
MVC Architecture

2. Service Oriented Architecture (SOA)

The business logic and application state are exposed as reusable services. An Enterprise Service Bus (ESB) is used as an orchestration and mediation layer to decouple the applications from the services.

16 Technical Key Areas

80+ step by step Java Tutorials



SOA (Service Oriented Architecture)

The above architecture has 5 tiers. The application tier could be using a typical MVC architecture. The service orchestration tier could be using ESB products like Oracle Service Bus, TIBCO, etc and BPM products like Lombardi BPM, Pega BPM, etc. In the above diagram, the ESB integrates with the BPM via messaging queues. The service tier consists of individual services that can be accessed through SOAP or RESTful web services. The SOA implementation requires change agents to drive adoption of new approaches. The BPM, application integration, and real-time information all contribute to dynamically changing how business users do their jobs. So, it needs full support from the business, requiring restructuring and also it can take some time to realize the benefits of SOA. Cloud computing is at the leading edge of its hype and as a concept compliments SOA as an architectural style. Cloud computing is expected to provide a computing capability that can scale up (to massive proportions) or scale down dynamically based on demand. This implies a very large pool of computing resources either be within the enterprise intranet or on the Internet (i.e on the cloud).

3. Web Oriented Architecture (WOA)

Q. Differentiate between SOA (Service Oriented Architecture) versus WOA (Web Oriented Architecture)?

Tools Tutorials (19)
Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
Can you write code? (
♦ Complete the given
Converting from A to I
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with th
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your .....?

open all | close all
Career Making Know-
Job Hunting & Resum

A. WOA extends SOA to be a light-weight architecture using technologies such as REST and POX (Plain Old XML). POX compliments REST. JSON is a variant for data returned by REST Web Services. It consumes less bandwidth and is easily handled by web developers mastering the Javascript language

SOA and WOA differ in terms of the layers of abstraction. SOA is a system-level architectural style that tries to expose business capabilities so that they can be consumed by many applications. WOA is an interface-level architectural style that focuses on the means by which these service capabilities are exposed to consumers. You can start out with a WOA and then grow into SOA.

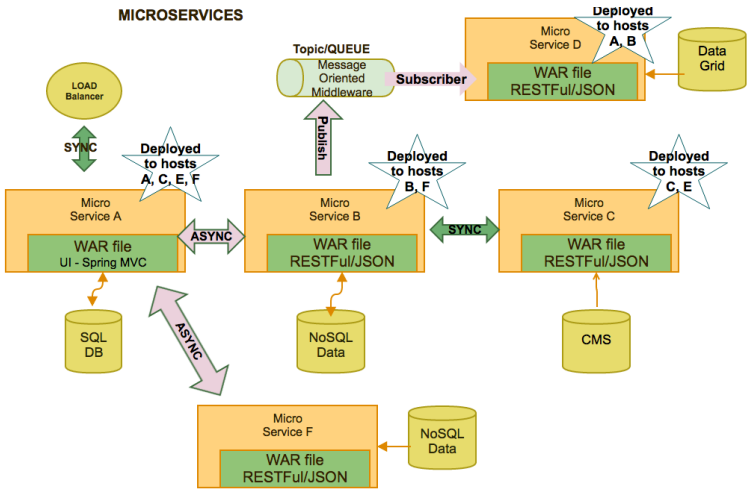
4. Micro Services Architecture (MSA)

Q. Differentiate between SOA (Service Oriented Architecture) versus MSA(Micro Services Architecture)?
A. Microservices allow large systems to be built from a number of collaborating components. The collaborating components can be web services, **messaging**, **event-driven APIs**, Web Sockets, and non-HTTP backed RPC mechanisms. So, A web service can be a microservice, but microservice might not be a web service.

SOA (Service Oriented Architecture)	**MSA (MicroServices Architecture)**
SOA is coarse-grained.	MSA is fine-grained. Often behavior & data are encapsulated. Adheres to the Single Responsibility Principle(SRP).
SOA focuses more on **re-usability**. SOA has higher coupling, where services	MSA focuses more on **low coupling & high cohesion**. It also

depend on other services to function.	emphasizes on **autonomy**, where services can be individually developed, deployed, scaled & monitored to provide business value on its own with both **behavior** & **data**. Lower coupling allows the service to operate independently and high cohesion increases it's ability to add value on its own.

Micro Services focus more on "loose coupling & high cohesion" than reuse. The definition of a **service** is an individual execution unit, and the definition of "**micro**" is the size, so that the service can be autonomously developed, deployed, scaled & monitored. This means the service is full-stack and has control of all the components like UI, middleware, persistence, and transaction. The services can be invoked both synchronously and asynchronously.

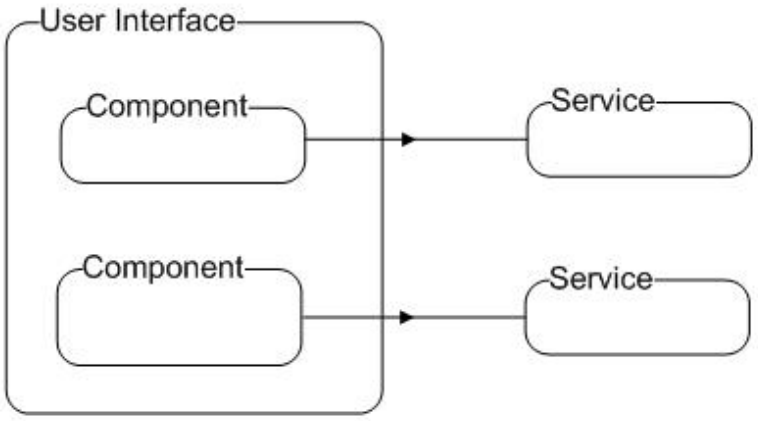


Microservices deployed to multiple hosts (e.g. on the cloud)

Micro services require a **service registration** as multiple processes working together need to find each other. **Spring Cloud** is built on Spring Boot, and incorporates the registration service called “Eureka” built by Netflix.

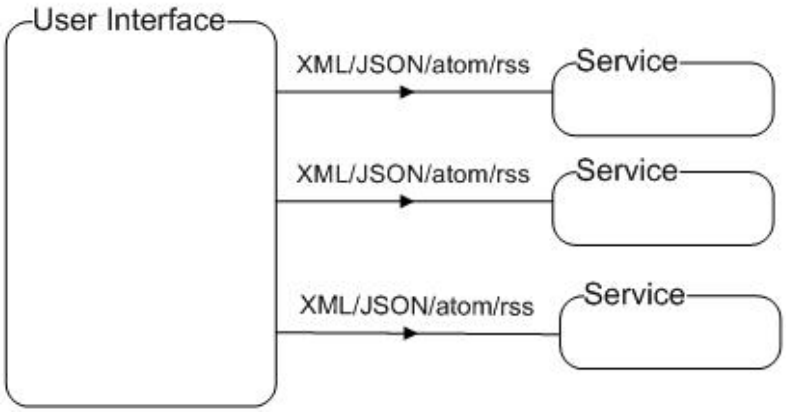
5. User Interface (UI) Component Architecture

This architecture is driven by a user interface that is made up of a number of discrete components. Each component calls a service that encapsulates business logic and hides lower level details. Components can be combined to form new composite components allowing richer functionality. These components can also be shared across a number of applications. For example, JavaScript widgets, Java Server Faces (JSF) components, etc.



UI Component Architecture

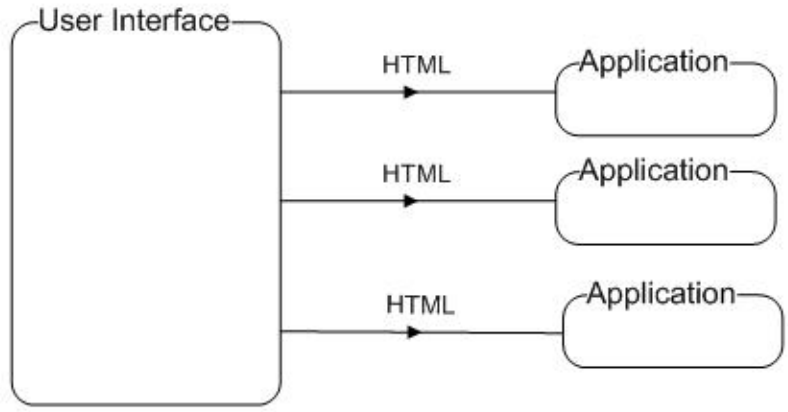
6. RESTful data composition Architecture



RESTful data composition

The user interface can be built by calling a number of underlying services that are each responsible for building part of a page. The user interface translates and combine the data in different formats like XML(translate to HTML using XSLT), JSON (Java Script Object Notation), ATOM (feed for mail messages and calendar applications), RSS (for generating RSS feeds), etc.

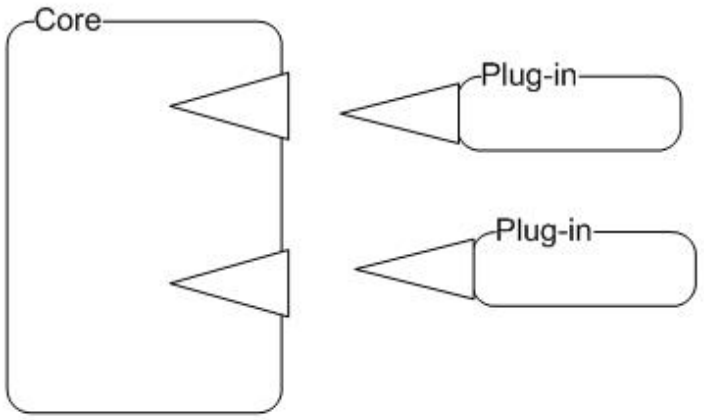
7. HTML composition Architecture



HTML composition architecture

In this architecture, multiple applications output fragments of HTML that are combined to generate the final user interface. For example, Java portlets used inside a portal application server to aggregate individual content, server side or client side mashups.

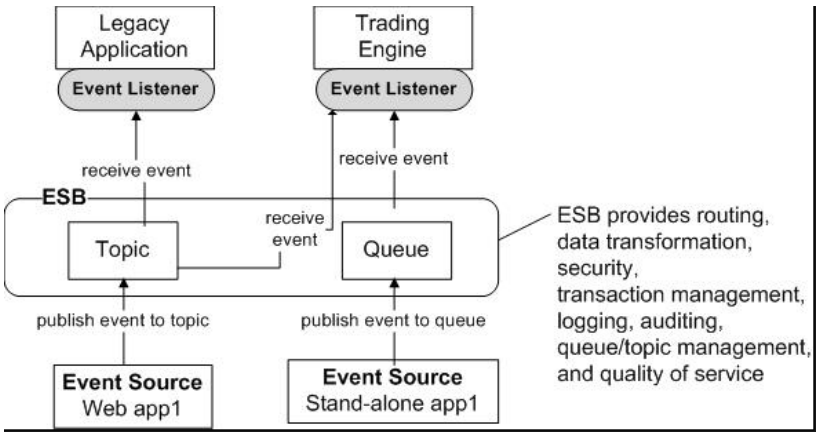
8. Plug-in Architecture



plugin architecture

In this architecture, a core application defines an interface, and the functionality will be implemented as a set of plug-ins that conform to that interface. For example, the the Eclipse RCP framework, Maven build tool, etc use this architecture.

9. Event Driven Architecture (EDA)



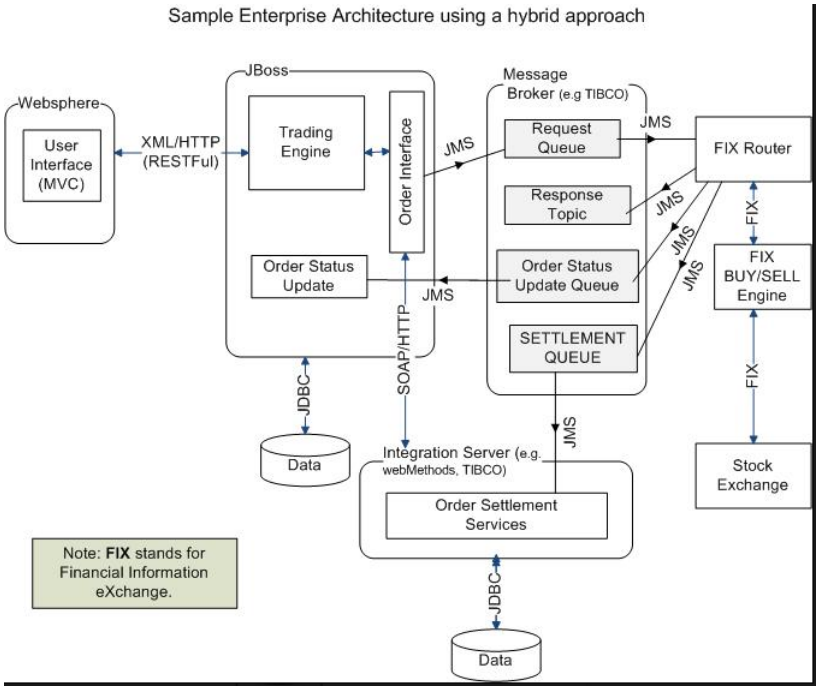
EDA

The EDA pattern decouples the interactions between the event publishers and the event consumers. Many to many communications are achieved via a topic, where one specific event can be consumed by many subscribers. The EDA also supports asynchronous operations and acknowledgments through event messaging. This architecture requires effective

monitoring in place to track queue depth, exceptions, and other possible problems. The traceability, isolation, and debugging of an event can be difficult in some cases. This architecture is useful in scenarios where the business process is inherently asynchronous, multiple consumers are interested in an event(e.g. order status has changed to partially-filled), no immediate acknowledgment is required (e.g. an email is sent with the booking details and itinerary), and real-time request/response is not required (e.g. a long running report can be generated asynchronously and made available later via online or via email).

10. Hybrid Architecture

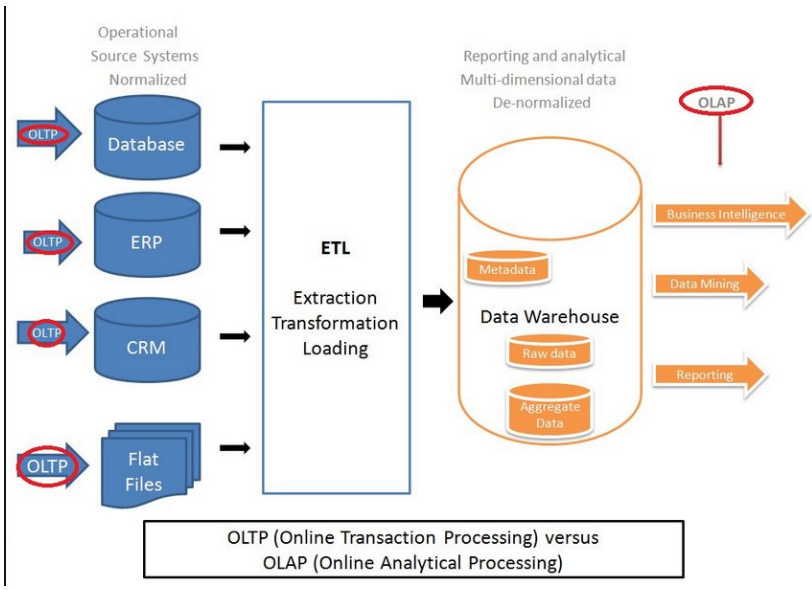
Most conceptual architectures use a hybrid approach using a combination of different architectures discussed above on the benefits of each approach and its pertinence to your situation. Here is a sample hybrid approach depicting an online trading system.



Hybrid Architecture

11. OLAP (OnLine Analytical Processing) Architecture

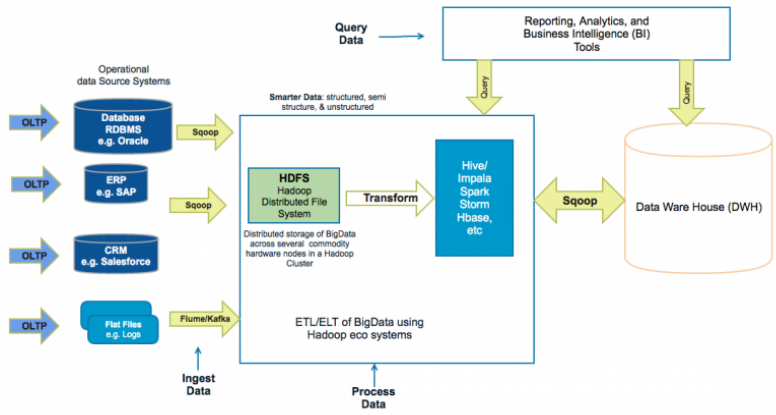
for business intelligence (BI), data mining, and complex reporting. IBM Cognos, JasperSoft, Oracle Enterprise BI server, etc are OLAP systems. ETL (Extract Transform & Load) or ELT (Extract Load & Transform) operations are performed to move OLTP data from various source systems into data warehouse systems. ETLs can be performed with Spring batch or commercial tools like Data Stage from IBM, Pentaho and Informatica.



OLTP vs OLAP

12. OLAP BigData Architecture

for business intelligence (BI), data mining, and complex reporting of BigData in peta bytes as opposed to tera bytes. Handles **structured** (e.g. RDBMS), **unstructured** (e.g. images, PDFs, etc), and **semi structured** (e.g. log files, XML files) data. As the volume and complexity of data grow, the time to process grows. The traditional ETL processes struggle to complete within the SLAs. Traditional ELT approach puts excessive loads into data warehouse (DWH). Hadoop can not only send some data to DWH, and also provides the infrastructure to be queried directly from the HDFS via HBase, Hive, Spark, Storm, etc. for the Big Data to be queried and analyzed.



Hadoop Architecture Overview

Q4. In your experience, what are some of the common architectural and development mistakes?

A4.

#1: Not externalizing configuration values in .properties or XML files. For example, not making the number of threads used in a batch job configurable in a .properties file. You may have a batch job that worked well in DEV to UAT (user acceptance) environments, but when deployed to PROD, due difference in the JDBC driver version or issue discussed in the #2 was throwing an IOException when run as multi-threaded over larger data sets. If the number of threads are configured in a .properties file, it can be easily made a single threaded application by changing the .properties file without having to redeploy and retest the application until a proper fix is made. This applies to all URLs, server and port numbers, etc.

#2: Not testing the application with the right volume of data. For example, testing your application with 1 to 3 accounts instead of 1000 to 2000 accounts, which is the typical scenario in the production environment. The performance tests need to be conducted with the real life data, and not cut down data. Not adhering to real life performance test scenarios can cause unexpected performance, scalability, and multi-threading issues. It is imperative that you test your application for larger volume of data to ensure that it works as expected and meets the SLAs

(i.e. Service Level Agreements) in the non-functional specification.

#3: Naively assuming that external or other internal services that are invoked from your application is going to be reliable and always available. Not allowing for proper service invocations timeouts and retries can adversely impact the stability and performance of your application. Proper outage testings need to be carried out. This is very crucial because the modern applications are distributed and service oriented with lots of web services. Indefinitely trying for a service that is not available can adversely impact your application. The load balancers need to be properly tested to ensure that they are functioning as expected by bringing each balanced node down.

#4: Not adhering to the bare minimum security requirements. As mentioned above, web services are everywhere, and web services can be easily exploited by the hackers for the denial of service attack. So, use of SSL layer, basic authentication, and penetration testing with tools like Google skipfish are mandatory. Unsecured applications can not only adversely impact stability of an application, but also can tarnish an organization's reputation due to data integrity issues like customer "A" being able to view customer "B's" data.

#5: Not performing cross browser compatibility testing. Modern web applications are rich single page applications making use of JavaScript code and frameworks like angularjs. The sites you build need to use responsive designs to work well across devices and browsers. It is imperative that proper cross-browser and device compatibility testing is performed to ensure it works in all of them.

#6: Not externalizing business rules that are likely to change often. For example, tax laws, government or industry compliance requirements, classification laws, etc. Use business rules engines like Drools that allow you to externalize rules into database tables and excel spreadsheets. The business can take ownership of these

rules, and can react quickly to changes to tax laws or compliance requirements with minimal changes and testing.

#7: Not having proper documentation in the form of

— Unit tests with proper code coverage.
— Integration tests.
— A confluence or wiki page listing all the software artifacts like classes, scripts, configuration files that have been modified or newly created.
— High level conceptual diagrams depicting all the components, interactions, and structures.
— Basic documentation for developers on “how to set up the DEV environment with data source details.

Points 1 and 2 are the primary form of documentation in an agile project in addition to the COS (Condition Of Satisfaction) created via tools like MindMap.

#8: Not having proper disaster recovery plans, system monitoring and archival strategies in place. It is easy to get missed on these activities in a rush to get the application deployed to meet the tight deadlines. Not having proper system monitoring through Nagios and Splunk can not only impact the stability of the application, but also can hinder current diagnostics and future improvements.

#9: Not designing Database tables with proper house keeping columns like created_datetm, update_datetm, created_by, updated_by and timestamp, and provision to logically delete records with columns like ‘deleted’ with ‘Y’ or ‘N’ values or record_status like ‘Active’ or ‘Inactive’.

#10: Not having proper system back out plan to restore the system to its stable state before deployment if anything goes wrong. This plan needs to be properly reviewed and signed-off by the relevant teams. This includes backing out to previous versions of software artifacts, any data inserted into the database, properties file entries, etc.

#11: Not performing proper capacity planning at the beginning of the project. Its no longer sufficient to simply say that you “need a Unix box, an Oracle database server and a JBoss application server” when specifying your platform. You need to be really precise about the

— specific versions of operating systems, JVMs, etc
— how much memory (including physical memory, JVM heap size, JVM stack size, and JVM perm gen space)
— CPU (number of cores)
— load balancer, number of nodes required, node types like active/active or active/passive and clustering requirements.
— file system requirements, for example, your application may archive generated reports and keep it for a year before archiving them. So, you need to have enough hard disk space. Some applications require to generate data extract files to be generated and temporarily stored to be picked up by the other system processes or data warehouse systems for multi dimensional reporting. Some data files are SFTP’ed from other internal or external systems, and need to be kept for a period like 12 to 36 months before archived.

Q5. What causes performance issues in Java?
A5.

#1. Not favoring lock free algorithms & non blocking I/O. Even the most well designed concurrent application that uses locks is at risk of blocking. For example, the java.util.concurrent package that allows concurrent reads and the Java NIO (New I/O using non-blocking multiplexers) respectively.

#2. Not reducing the memory size. Reduce the number of objects you create. Apply the flyweight design pattern where applicable. Favor stateless objects. Where applicable write immutable objects that can be shared between threads. Fewer objects mean lesser GC.

#3. Not tuning your JVM. Tune your JVM with appropriate heap sizes and GC configuration. Before tuning profile your

application with real life data. Basically you want to avoid GC pauses and increase GC throughput.

#4. Not using caching at all or caching the wrong objects.

#5. Writing bad SQLs or regular expressions that back tracks frequently. Bad database design without proper indices or table partitioning.

#6. Not having proper service timeouts.

#7. Making too many network round trips.

#8. Writing your own solution as opposed to favoring a proven third-party library and choosing the wrong framework, tool, or library for the task at hand. For example, favoring an SQL database when NoSQL is more suited for the task at hand.

#9. Not testing the load balancers and server clustering with proper outage and performance testing.

#10. Not having proper performance focus from the beginning. No proper capacity planning and JMeter scripts to test performance.

Q6. In your experience, what are some of the key security considerations in an enterprise Java application?
A6.

#1: Using HTTP over SSL. That is using HTTPS.

#2: Using two way SSL for RESTful Web Service calls where application1 invokes application2 via RESTful Web services.

#3: Sensitive data in .properties and other config files need to be properly encrypted.

#4: All applications need to be subjected to security audit and penetration testing using tools like Google Skipfish.

#5: Proper process should be in place and the team should be educated on the importance of security procedures and techniques. Look for potential security holes during peer code reviews. Continuously review and upgrade the processes and the procedures. Have proper password vaults with limited access to them.

#6: Having proper authorization, authentication, and password reset processes implemented in the application. Use SSO (Single Sign On) and 2 factor authentication where required.

#7: Having proper disaster recovery plans and performing regular backups.

#8: Having proper regular monitoring in place and keeping archived logs for a certain period to analyze any security holes or threats.

Q7. Can you list some key software design principles?
A7.

#1: DRY (Don't Repeat Yourself) principle. Divide systems into components, and, further, components into sub components to manage complexity and avoid duplication. Clearly define responsibilities of each system, component, and sub component.

#2: KISS (Keep It Simple Stupid) principle. Don't over engineer your system. Don't anticipate every possible future requirement. Build the system for the current requirements. If built with best practices for the immediate requirements, the system can adapt to future changes.

#3: YAGNI (You "Ain't Gonna Need It) principle. Don't build what is not required. Carefully determine the core requirements to build the minimum viable product. Apply the 80/20 principle. The 80% of the value is going to be derived from the 20% of the functionality. YAGNI is similar to the KISS principle, as it strives for a simple solution. However, KISS strives for a simple solution by trying to implement something

as easily as possible. YAGNI strives for simplicity by not implementing it at all.

#4: Design by Contract. Clearly define the pre and post conditions of your system, components, and modules. Code to interface as opposed to implementation.

#5: Favor **low coupling**, **high cohesion**, and **high encapsulation** in the modules and systems you build.

#6 Give equal importance to **non functional requirements** like security, performance, scalability, cross browser compatibility, monitoring, auditing, archiving, disaster recovery and maintainability when designing systems.

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

767 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

389 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

240 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ SSL and truststore vs keystore for Java developers

Novice Java Sample Resume or CV ›

Posted in Java Application Architecture Interview Essentials, Java Architecture Interview Q&A, member-paid

Tags: Architect FAQs, Java/JEE FAQs, JEE FAQs

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.

↑