JReport®
JINFONET SOFTWARE

JReport Server
Deployment Scenarios

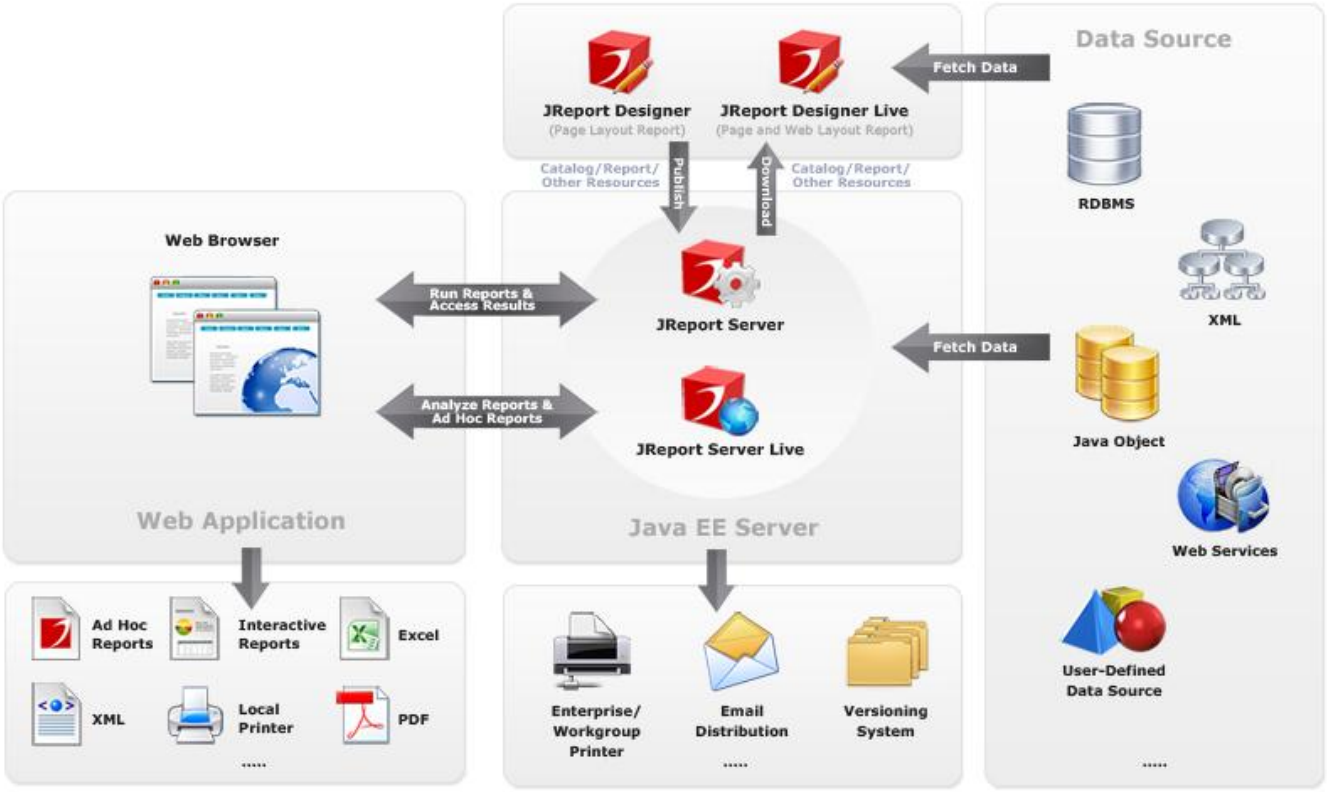
Contents

Introduction

JReport® is the leading embedded reporting solution for Java applications. JReport seamlessly integrates into a Java application to provide a Java EE-compliant, sophisticated and powerful reporting service. The flexibility of the JReport architecture allows it to integrate with varied application architectures, from a simple client/server application to a full-scale enterprise application. This paper describes some of the most common application deployment scenarios and how the JReport reporting solution works with them.

JReport Architecture

Like any web-based Java application tool, JReport has both a development-time and a runtime component: JReport Designer and JReport Server.



Reports are created in the 100% Java, Swing-based JReport Designer, which provides an intuitive Interactive Design Environment (IDE). The report developer can access the desired data source via queries and then build the report by dragging and dropping the data source elements and other objects from the resource palette or the toolbar. Or the Report Wizard can automate the basic report design, which can then be fine-tuned. JReport Designer has a built in preview engine, allowing a report developer to quickly iterate through design and testing sessions by switching between the design view and runtime view of the report with live data.

JReport Server is a 100% Java and Java EE-based server to which reports are first published and then managed via the web-based JReport Administration Console. The JReport Server reliably and robustly performs the duties of an enterprise-level reporting service. It enables the efficient management, sharing, scheduling and delivery of reports and the viewing of them in DHTML or HTML format by end users in web browser sessions. The high-performance engine scales up to meet increased workload. Report results can also be rendered into standard file formats including Microsoft Excel, Adobe Portable Document Format (PDF), XML and text files.

There are two ways in which the JReport Server can be deployed to an enterprise application's runtime environment:

- As an integrated component of a web application
- As a standalone server

JReport Server Integrated with a Web Application

In a runtime environment, the integrated JReport Server is represented as standard Java or Java EE components and therefore can be packaged, deployed, and maintained in the same way as the components in a web application.

More specifically, the JReport Server can run as a servlet or as an EJB in containers provided by the Java EE application server. The container supports services such as request dispatching, security, concurrency, and life cycle management. The container also provides access to APIs such as JDBC, naming, transactions, and email. JReport Server fully leverages the Java EE architecture and plugs into these services.

Because it deploys as standard Java EE components, there are many different types of web applications with which the JReport Server can be integrated. This paper describes 8 of the most common scenarios. Each of these scenarios offers a different range of scalability and availability:

		Scalability		
		Lower	**Medium**	**Higher**
Availability	**Lower**	*Scenario 1*	*Scenario 3*	*Scenario 6*
	Medium	*Scenario 2*	*Scenario 4*	*Scenario 7*
	Higher		*Scenario 5*	*Scenario 8*

Table 1: Scalability and Availability Supported by JReport Deployment Scenarios

Note that the actual availability and scalability of the JReport Server is affected by several factors including the types of reports, the number of concurrent viewing sessions, the CPU power and memory of the Java EE Server host, and the size and performance of the database or data source.

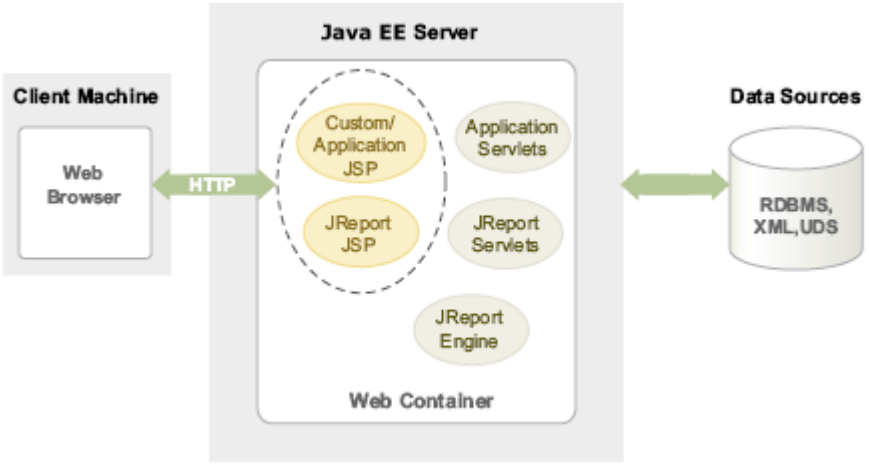
Scalability	*Lower*
Availability	*Lower*

Scenario 1: Single Java EE Server with a Single Instance of JReport Server

At runtime, an instance of the JReport Server consists of the following components: the JReport Engine which is the core component that generates each report result, the JReport JSP which provides an HTTP interface to the Engine, and the JReport servlets. A typical web application also contains application-specific servlets and customized versions of the JReport JSP templates that access the JReport Engine to request a report.

The JReport components can access a variety of data sources, including relational databases via JDBC or ODBC, XML via the hierarchical data source API, and other user-defined data sources (UDS).

The simplest and most direct way to deploy these components is to deploy them as a web application to the same Java EE Server:

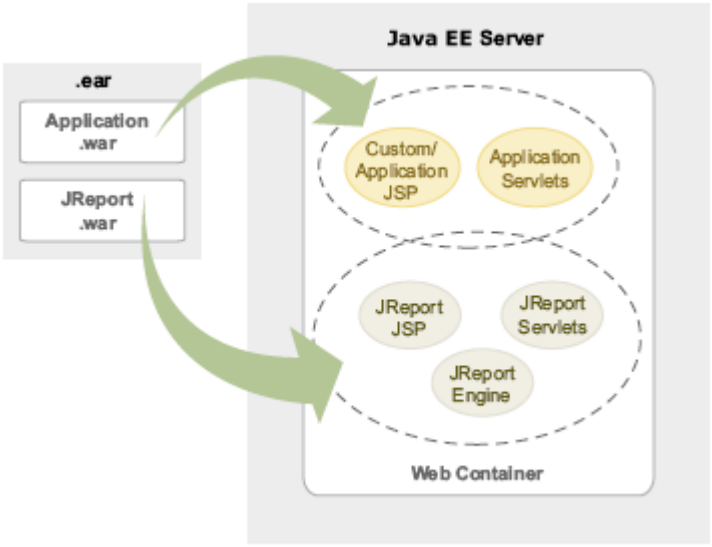


Benefits: This scenario allows direct communication among the components in the web container through local calls within the same JVM. The JReport Server is managed seamlessly as part of the application: it is deployed when the application is deployed; it starts when the application is started; and it stops when the application is stopped. The reporting service is always available to the application.

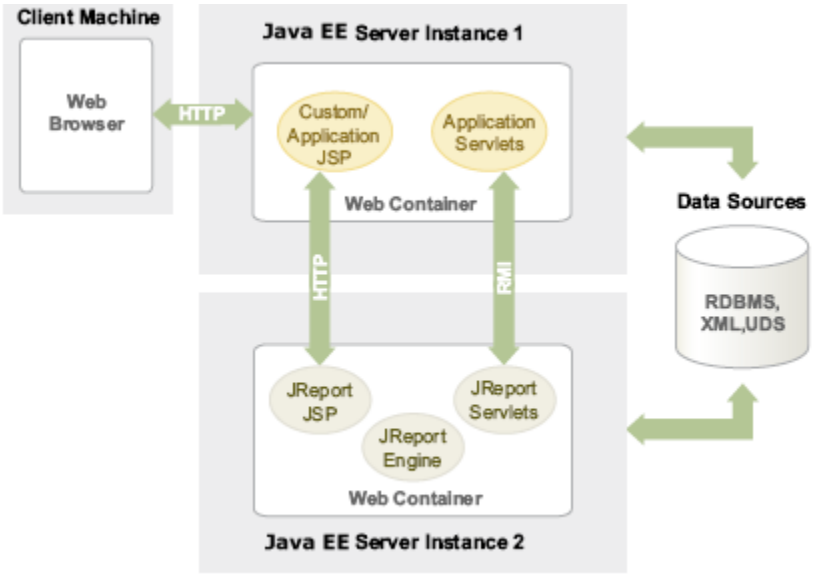
This scenario works very well for enterprise customers of JReport as well as independent software vendors (ISVs) who want to deploy JReport as a seamlessly integrated part of their application from a technical and user interface standpoint. The integrated JReport components can adapt the look and feel of any host application.

Web applications that use this scenario are easy to deploy because JReport includes a utility that automatically creates a web archive (WAR) file with all the JReport Server code that is required in the runtime environment.

After the application components (in one or more WAR files) are combined with the JReport WAR file into an enterprise archive (EAR) file, an application unit is created that can be deployed to a local or remote Java EE server at any time.



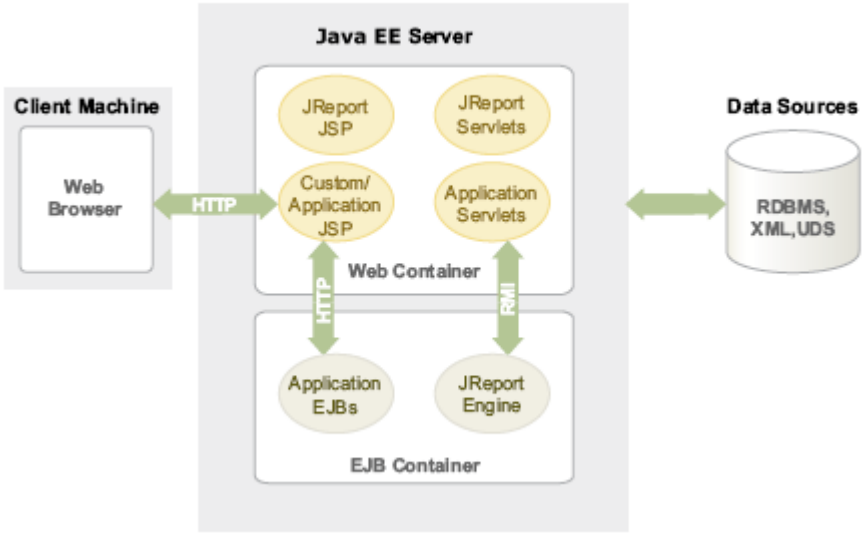
Another way that a single instance of the JReport Server can be deployed is in its own web container in a separate Java EE server instance on the same host. The application pieces-the custom JSP and servlets that access the JReport Engine-are deployed to a separate web container in a separate Java EE server instance on the same host. Calls to the JReport component can be accomplished through RMI or HTTP:



Benefits: In this scenario the components in the separate web containers communicate through HTTP calls or RMI calls. The JReport Server is managed separately from the application: it is deployed, started, and stopped independently from the application. Because the JReport Server operates in its own JVM, its Java EE server instance can be tuned separately from the application's Java EE server instance.

In this scenario, the application WAR files would be deployed to the Java EE Server Instance 1 and the JReport WAR file would be deployed to the Java EE server instance 2.

EJB Variation: The previous scenarios described the JReport Server running in a web container as servlet-based components, which is the default configuration of the JReport Server. The JReport Server can also run as EJB-based components in an EJB container:



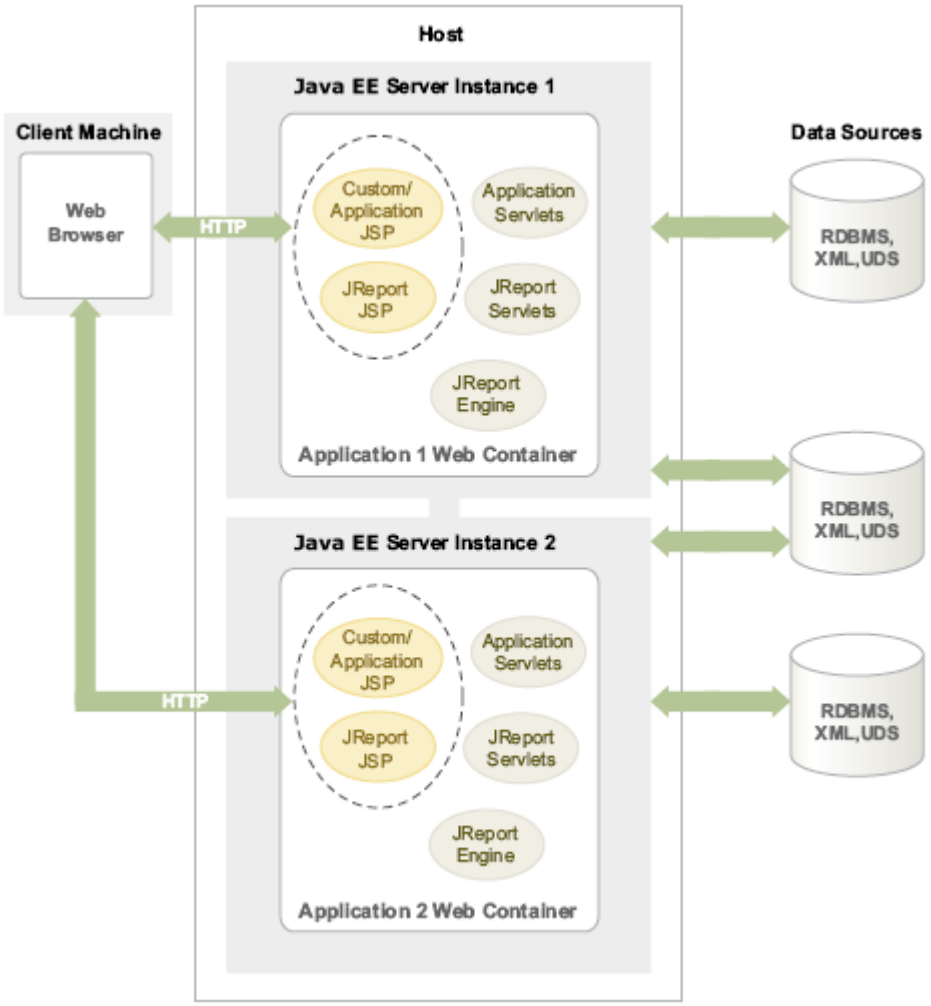
Benefits: If the application uses EJBs for the business logic then JReport Server can be started and managed from the application's EJB. This scenario allows the JReport Server to leverage the architecture of the EJB container.

Throughout the remainder of this document, the JReport Server is shown and described as being run as servlets-based rather than EJB-based. However, in each scenario the JReport Server could also be run as an EJB in an EJB container.

Scalability	*Lower*
Availability	*Medium*

Scenario 2: Single Java EE Server with Multiple Instances of JReport Server

Some runtime environments benefit from running multiple instances of the JReport Server on the same host machine. Each JReport Server instance would run in its own Java EE Server instance:

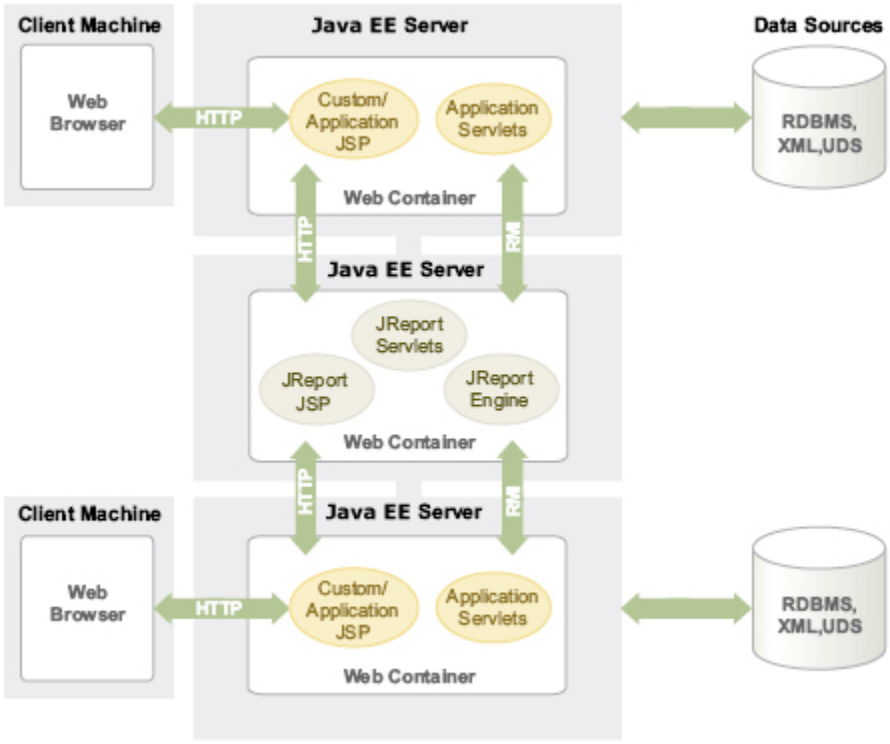


Benefits: This scenario provides the benefits of seamless integration of web application and JReport Server that were described earlier. In this case, two web applications that each use their own instance of the JReport Server A problem in one application does not prevent the other application from accessing its reporting service. There is no sharing of reporting data between them and no conflict arising from having two JReport Servers running on the same host.

Scalability	*Medium*
Availability	*Lower*

Scenario 3: Multiple Java EE Servers with a Single Instance of JReport Server

With multiple Java EE servers running on separate hosts, one server can be dedicated to host the JReport Server instance. The JReport Server is accessed from applications running in other Java EE servers and can optimally manage the multiple incoming requests:

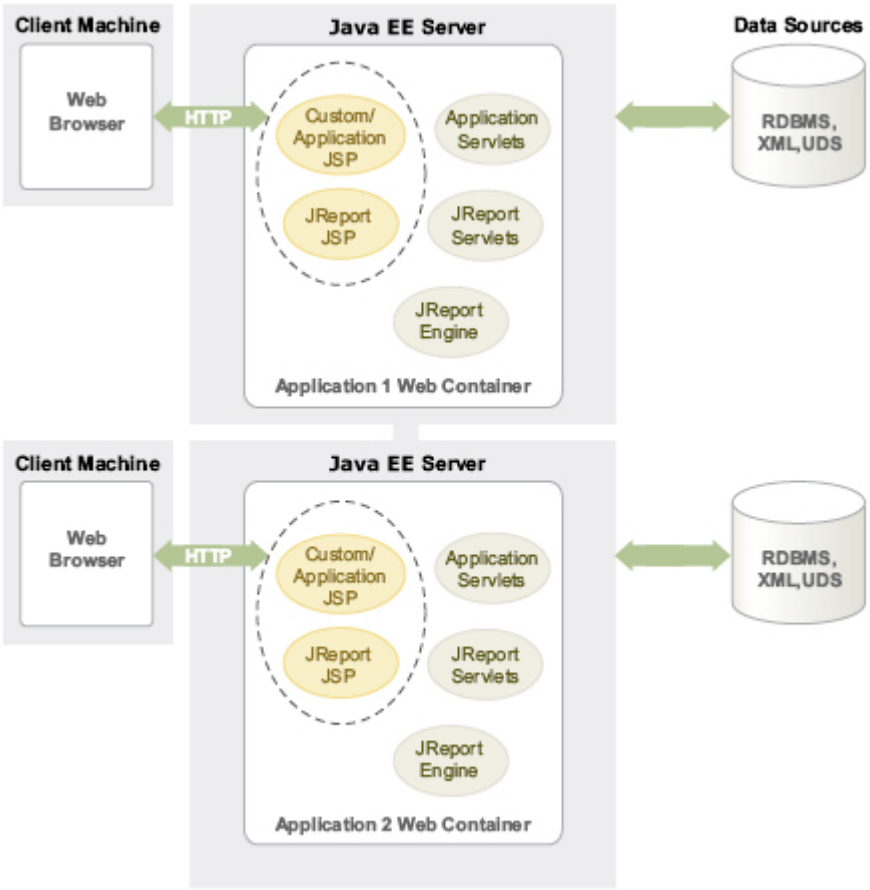


Benefits: This scenario provides a single reporting service that is available to one or more applications. The reporting service can perform robustly here because it has the resources of a dedicated Java EE server instead of one also being used by the application. The Java EE server that hosts the JReport Server can be tuned to optimize the performance of the JReport Server without affecting the application Java EE servers which may require different tuning settings.

Scalability	*Medium*
Availability	*Medium*

Scenario 4: Multiple Java EE Servers with Multiple Instances of JReport Server

In this scenario, multiple hosts are also used to provide additional reporting scalability. Each host has its own dedicated instance of the JReport Server. The JReport Server and the application in the same container are highly accessible to one another; there is no overheard of remote communication:



Benefits: This scenario is good for applications that do not require the higher level of reporting service throughput delivered from clustering, but do require medium reporting service availability to each application.

This scenario works well when the applications that use the reporting services do not need to share other resources on the network such as data sources. For example, a Sales report application can be deployed and maintained separately from a Manufacturing report application. If one application goes down the other is not affected.

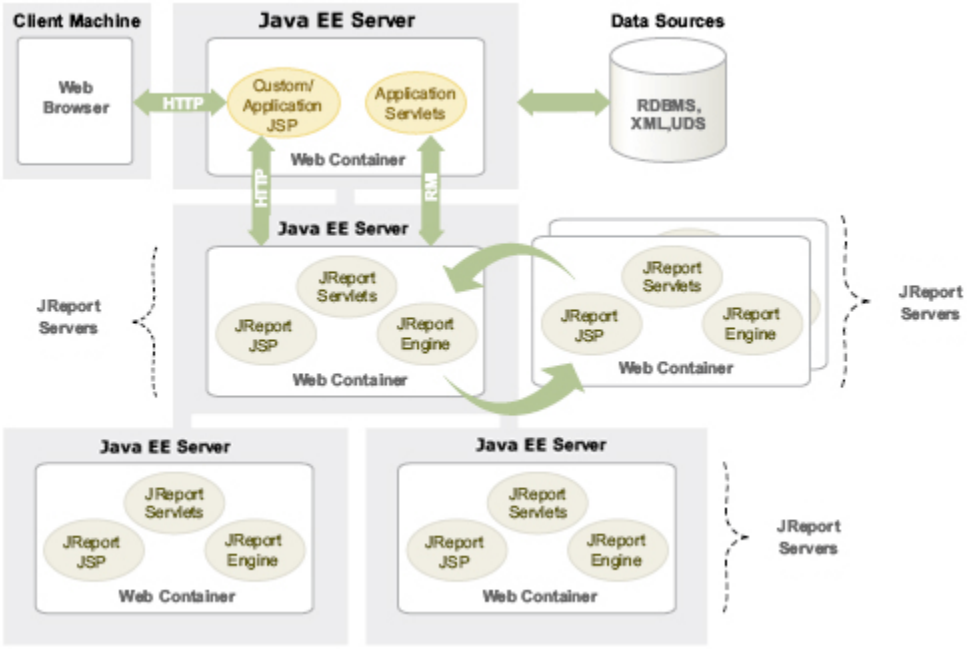
Scalability	*Medium*
Availability	*Higher*

Scenario 5: Multiple Java EE Servers with Clustered instances of JReport Server

With multiple Java EE servers, the JReport Server clustering feature can be used to maximize the performance of each JReport Server instance

In this scenario, the JReport Server is deployed to each Java EE server, and there it is configured to be the admin server, the admin server's backup, or a slave server. All report requests are routed first to the admin server and then to the suitable slave server; all configuration and maintenance is controlled from the admin server.

From the JSP or servlet requesting the report, it appears as though there is just one JReport Server. One or more backup servers can be configured to be used automatically if an admin failure is detected. The backup server in the diagram can be located on any Java EE server in the network.

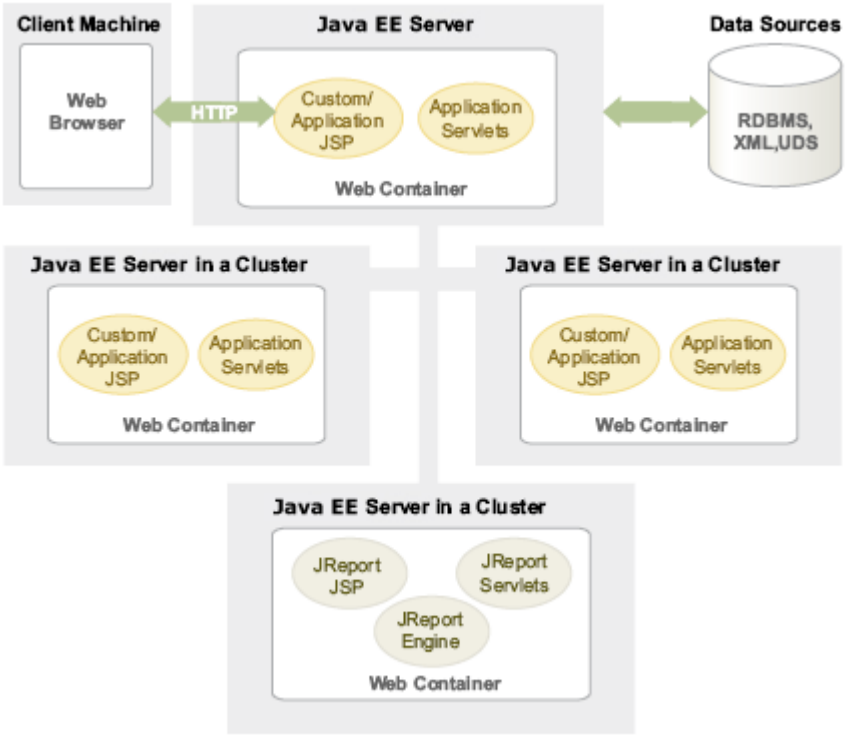


Benefits: Clustering delivers improved "n-fold" throughput because there are "n" more JReport engines to generate reports concurrently. The admin server routes the report generation request to the slave server with the highest availability to generate the report based on one of three load-balancing algorithms. Therefore, each JReport Server instance is used to its fullest potential. Clustering also delivers a very high level of reporting service availability because if a Java EE server or JReport Server fails, the associated requests are automatically reallocated to other servers.

Scalability	*Higher*
Availability	*Lower*

Scenario 6: Clustered Java EE Servers with a Single Instance of JReport Server

The runtime environment of a web application may include Java EE servers operating under the Java EE server's clustering feature. A JReport Server instance operating within the cluster receives the same availability and scalability benefits as other application components in the cluster. As long as the workload is appropriate, a single instance of the JReport Server can provide the reporting service to all of the Java EE servers in the cluster:

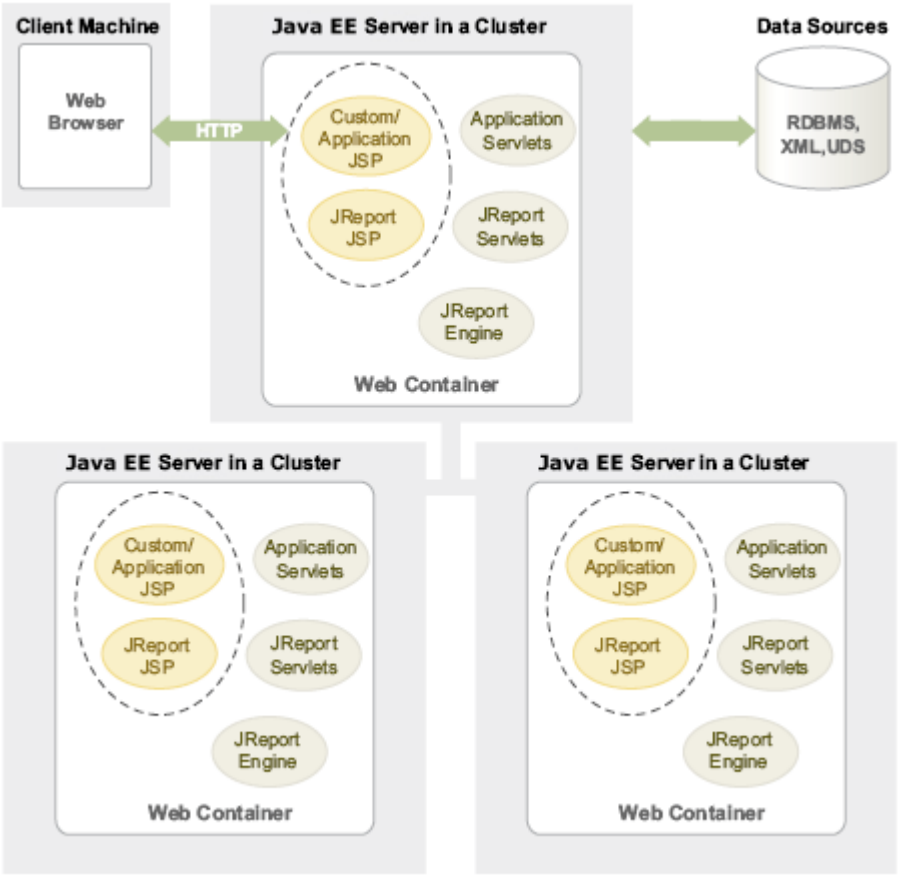


Benefits: This scenario relies on the clustering feature of a Java EE server product to achieve a high performance level for the entire application. The scenario provides a reporting service that is shared among multiple applications. The JReport Server is deployed and maintained separately from them. This scenario is useful when the combined throughput requirements of the applications can be met by a single reporting service.

Scalability	*Higher*
Availability	*Medium*

Scenario 7: Clustered Java EE Servers with Multiple instances of JReport Server

Java EE servers operating under the Java EE server's clustering feature can each have their own instance of the JReport Server:

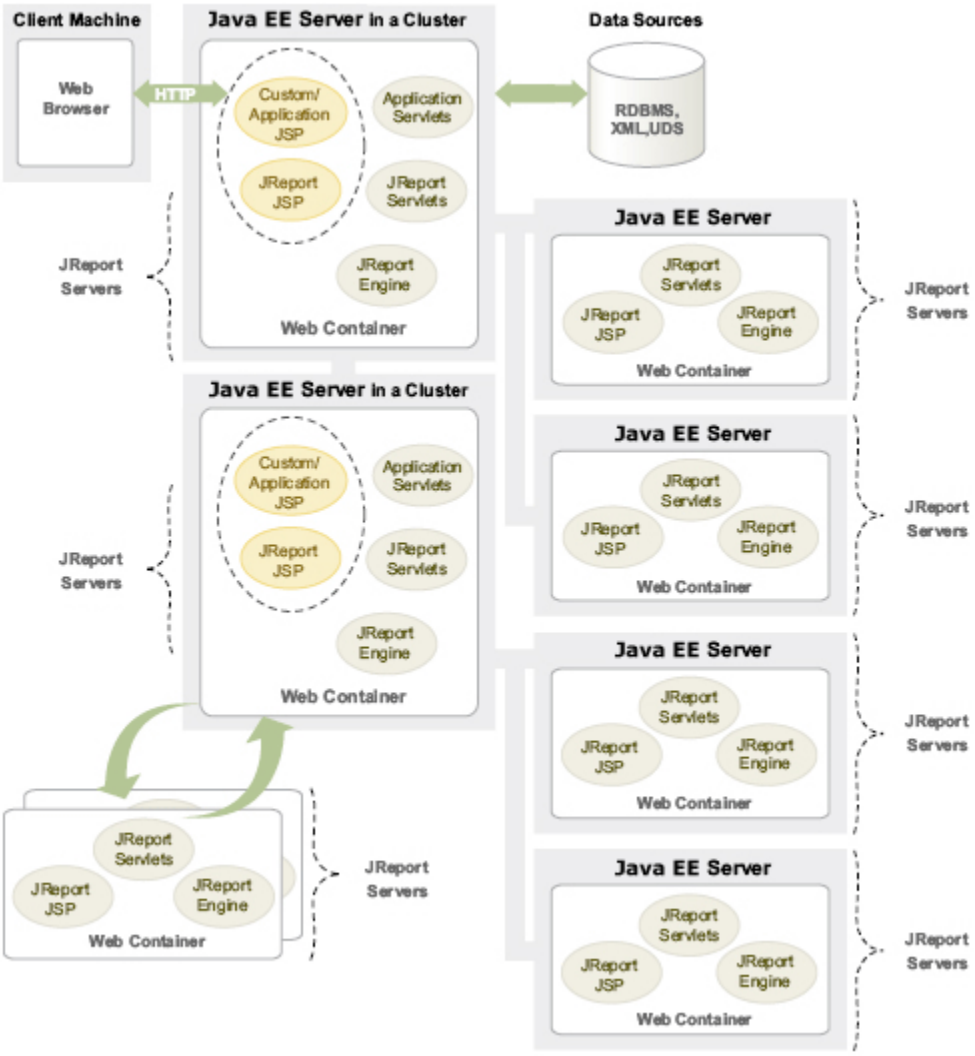


Benefits: This scenario relies on the clustering feature of a Java EE server product to achieve a high performance level for the entire application. Each application has access to a local JReport Server.

Scalability	*Higher*
Availability	*Higher*

Scenario 8: Clustered Java EE Servers with Clustered Instances of JReport Server

Java EE servers operating under the Java EE server's clustering feature can each have their own instance of the JReport Server:

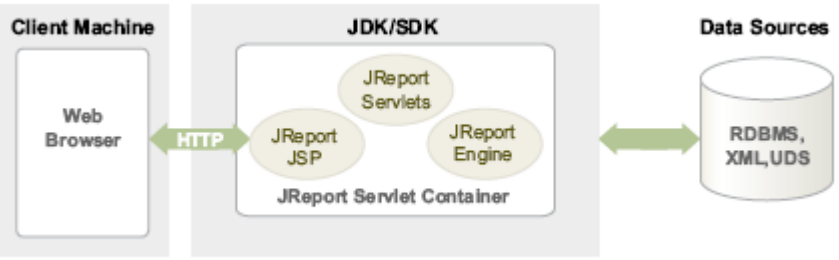


Benefits: This scenario provides the benefits of Java EE server clustering combined with the benefits of JReport Server clustering. That is, this scenario offers a very high level of report throughput, reporting service reliability, and reporting service failover for any application in the Java EE server cluster.

JReport Server as a Standalone Server

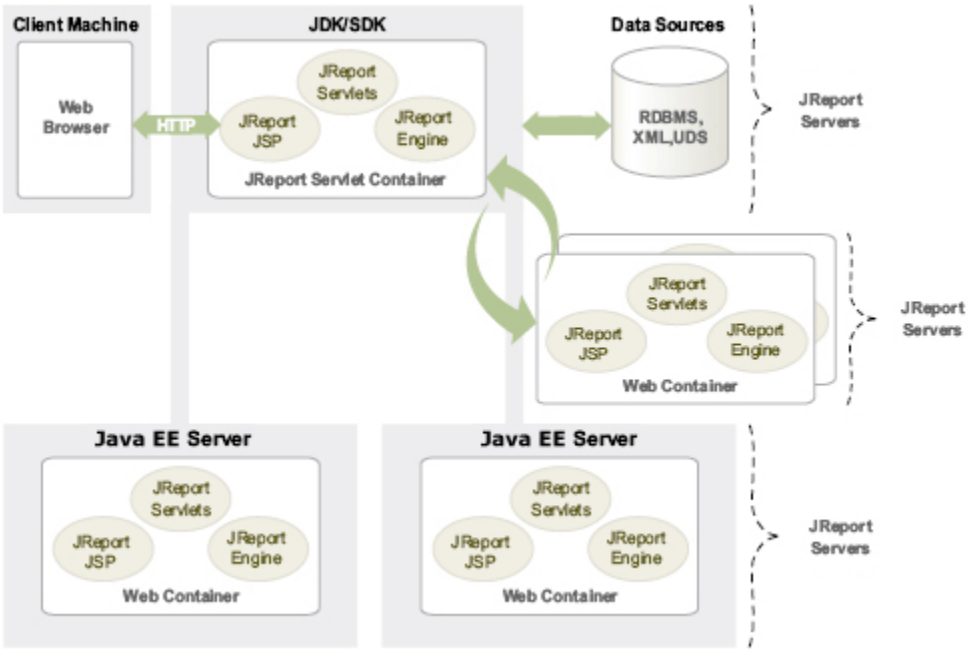
In the previous scenarios, the JReport Server ran inside of a servlet container provided by the Java EE server. The JReport Server instance can instead run in a servlet container provided by JReport. Because a Java EE server is not required, running the JReport Server in this way is called a standalone JReport Server.

The standalone JReport Server can be run on any machine that has the Java JDK/SDK:



Benefits: The standalone JReport Server provides the same reporting service functionality as the embedded JReport Server, but without the requirement and overhead that it is run on a third-party Java EE server Management of the standalone JReport Server is done directly through the JReport tools. This scenario runs out-of-the-box; no additional configuration or setup is required. Additionally, the JReport Server in this mode can be run as an operating system service so that it is automatically started when the operating system starts.

The standalone JReport Server can also operate as a clustered set of JReport Servers:



Benefits: This scenario provides an easy way to set up and mange clusters of JReport Server, without using a third-party Java EE server. The JReport Server Monitor can be used to directly manage the clustering.

A standalone JReport Server, single or clustered, can be accessed from the Java application or web application through HTTP or RMI calls.

Conclusion

This paper described a variety of web application deployment scenarios and how the JReport Server integrates with each of them. This flexibility of the JReport Server architecture is why it is the leading embedded reporting solution for Java applications.

Our experienced application design consultants can further advise which deployment scenario will best meet your application requirements or specifically, how JReport can be seamlessly embedded into your application environment. Please visit us at http://www.jinfonet.com for more information.

Order your evaluation copy of JReport at: http://www.jinfonet.com/eval

More Information

Schedule a Free Evaluation to Review Your Specific Reporting Needs

Contact Us

Tel:	+1 240-477-1000
Fax:	+1 240-465-0355
Website:	http://www.jinfonet.com
General Info:	info@jinfonet.com
Sales:	sales@jinfonet.com
Support:	support@jinfonet.com
Marketing:	marketing@jinfonet.com