Sun Java™ System

Sun Java Enterprise System Deployment Planning White Paper

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817-5759-10

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Deployment Planning

This white paper provides an introduction to planning large scale deployments based on Sun Java™ Enterprise System. It presents some basic concepts and principles of deployment planning and introduces a number of processes that you can use as a starting point when designing enterprise-wide deployments.

If you are evaluating Java Enterprise System or planning to create and deploy large scale applications based on Java Enterprise System, use this paper as a guide to the deployment planning process.

This chapter provides a brief overview of Java Enterprise System and introduces concepts on deployment planning that are discussed in later chapters. This chapter contains the following sections:

About Java Enterprise System

The Java Enterprise System is a software infrastructure that provides services to support enterprise-strength applications distributed across a network or Internet environment. The following table lists the components of the Java Enterprise System and the infrastructure services they provide.

Table 1-1 Java Enterprise System Components

System Component	Services Provided
Application Server	Provides Java 2 Platform, Enterprise Edition (J2EE™ platform) container services for Enterprise JavaBeans™ (EJB) components, such as session beans, entity beans, and message-driven beans. The container provides the infrastructure services needed for tightly-coupled distributed components to interact, making it a platform for the development and execution of e-commerce applications and web services. The Application Server also provides web container services.
Calendar Server	Provides calendar and scheduling services to end users and groups of end users. Calendar Server includes a browser-based client that interacts with the server.
Directory Proxy Server	Provides security services for Directory Server from outside a corporate firewall. Directory Proxy Server provides enhanced directory access control, schema compatibility, routing, and load balancing for multiple Directory Server instances.
Directory Server	Provides a central repository for storing and managing intranet and Internet information such as identity profiles (employees, customers, suppliers, and so forth), user credentials (public key certificates, passwords, and pin numbers), access privileges, application resource information, and network resource information.
Identity Server	Provides access management and digital identity administration services. Access management services include authentication (including single sign-on) and role-based authorization for access to applications and/or services. Administration services include centralized administration of individual user profiles, roles, groups, and policies.
Instant Messaging	Provides secure, real-time communication between end users, such as instant messaging (chat), conferencing, alerts, news, polls, and file transfer. The service includes a presence manager that tells users who is currently on line and includes a browser-based client that interacts with the server.

Table 1-1 Java Enterprise System Components *(Continued)*

System Component	Services Provided
Message Queue	Provides reliable, asynchronous messaging between loosely-coupled distributed components and applications. Message Queue implements the Java Message Service (JMS) API specification and adds enterprise features such as security, scalability, and remote administration.
Messaging Server	Provides secure, reliable, high-capacity store-and-forward messaging that supports email, fax, pager, voice, and video. It can concurrently access multiple message stores and provides content filtering to help reject unsolicited email and prevent virus attacks.
Portal Server	Provides key portal services, such as content aggregation and personalization, to browser-based clients accessing business applications or services. Portal Server also provides a configurable search engine.
Secure Remote Access	Provides secure, Internet access from outside a corporate firewall to Portal Server content and services, including internal portals and Internet applications.
Web Server	Provides J2EE platform web container services for Java web components, such as Java Servlet and JavaServer Pages™ (JSP™) components. The Web Server also supports other web application technologies for delivering static and dynamic web content, such as CGI scripts and Active Server Pages.
Sun Cluster	Provides high availability and scalability services for the Java Enterprise System, the applications that run on top of the Java Enterprise System infrastructure, and the hardware environment in which both are deployed.

Java Enterprise System Suites of Services

Java Enterprise System deployments typically fall into two general categories, those consisting primarily of services provided by Java Enterprise System and those that integrate a significant number of custom developed services and third party applications. You can think of the former type of deployment as an *80:20 deployment* (80% of the services are provided by Java Enterprise System) and similarly, the latter as a *20:80 deployment*.

NOTE Actual enterprise deployments can vary greatly in the amount of custom developed services they require.

Java Enterprise System is particularly suited to 80:20 deployments because of its rich set of services it. For example, it is relatively easy to deploy an enterprise-wide communications system or an enterprise-wide portal system.

However, for deployments requiring custom development, Java Enterprise System provides the ability to create and integrate custom developed services and applications.

The following table groups Java Enterprise System components into suites that can deliver enterprise deployments. Some components are in more than one suite.

Table 1-2 Java Enterprise System Suites of Services

Suite	Java Enterprise System Components
Network Identity Services	Identity Server Directory Server Web Server
Enterprise Portal Services	Portal Server Secure Remote Access Identity Server Directory Server Application Server or Web Server
Enterprise Communication and Collaboration Services	Messaging Server Calendar Server Instant Messaging Identity Server Directory Server Application Server or Web Server
Web and Application Services	Application Server Message Queue Web Server
Availability Services	Sun Cluster 3.1 4/04 Sun Cluster Agents

Most of the suites in Table 1-2 above can deliver 80:20 type of deployments. For example, the Enterprise Communications and Collaboration suite can be used to create a deployment that provides email, calendar, and instant messaging services to end users, allowing them to aggregate and personalize the content. Similarly, the Network Identity and Enterprise Portal suites allow you to install and configure enterprise-wide applications without having to develop or integrate custom services.

The Availability Services suite provides high availability to large scale deployments of enterprise applications. Use the Web and Application Services suite if your enterprise application requires custom development of J2EE platform services that run in an application server or web server.

Because of the interoperability between Java Enterprise System services, you can create your own suite of services tailored to your particular enterprise needs.

Advantages of Java Enterprise System

Enterprise deployments have three keys to success.

- Time to deliver
- Cost of delivery
- Functionality

Java Enterprise System provides you with the tools to address each of these keys to success, as indicated in the following table.

Table 1-3 Java Enterprise System Advantages

Advantage	Description
Simple to use	Java Enterprise System provides a common installer, which makes it easy to install, configure, and upgrade. Java Enterprise System provides a shift from integrating independently developed, single products and middleware to a system of integrated platform services that can be deployed and configured with little customization.

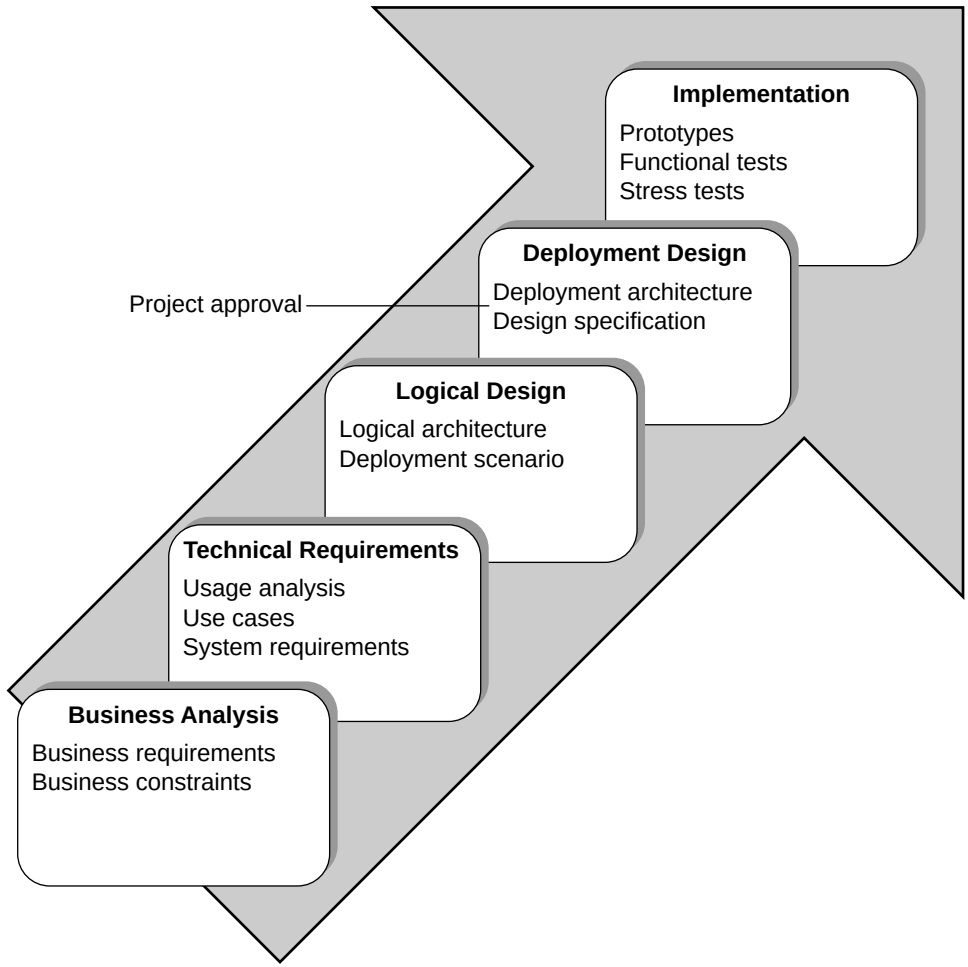
Table 1-3 Java Enterprise System Advantages *(Continued)*

Advantage	Description
Predictable	The Java Enterprise System release cycle takes into account compatibility among the Java Enterprise System components. When you upgrade to a new release, you can avoid incompatibility and misalignment between the components.
	Java Enterprise System components use a set of shared platform components, which makes it easier for services to interoperate.
	Java Enterprise System delivery model of scheduled releases provides predictability in deployment planning.
Affordable	Java Enterprise System single unit pricing model for commercial licenses reduces the complexity and cost of installing and upgrading a deployment. The single unit price includes support, maintenance, and consulting services.
	Other pricing models for OEM and education licenses are available.

About Deployment Planning

Successful deployment planning is the result of careful preparation, analysis, and design through a series of phases, as illustrated in the following figure.

Figure 1-1 Deployment Planning Phases



Each phase depicted in Figure 1-1 has its own set of analyses and procedures that results in specifications and designs carried forward to subsequent phases. The following sections in this chapter provide a summary description of each deployment planning phase.

Business Analysis Phase

During the business analysis phase you define the business goal of a deployment project and state the business requirements that must be met to achieve that goal. When stating the business requirements, consider any business constraints that might impact the ability to achieve the business goal. The business analysis phase results in a *business requirements document* that you later use in the technical requirements phase and against which you later measure the success of the deployment design.

For more information on the business analysis phase, refer to Chapter 2, "Business Analysis" on page 21.

Technical Requirements Phase

The technical requirements phase starts with the business requirements you create during the business analysis phase and translates these requirements into technical specifications that can be used to design the deployment architecture. During the technical requirements phase you prepare the following information:

- Analysis of user tasks and usage patterns
- Use cases that model user interaction with the planned deployment
- System requirements derived from the business requirements, taking into consideration the analysis of user tasks and usage patterns

The resulting set of *usage analysis*, *use cases*, and *system requirements documents* are inputs to the logical design phase.

During technical requirements analysis, you might also specify *service level requirements*, which are the terms under which customer support must be provided to remedy a deployed system failure to meet system requirements. Service level requirements are the basis for *service level agreements* signed during project approval.

For more information on the technical requirements phase, refer to Chapter 3, "Technical Requirements" on page 27.

Logical Design Phase

Deployment design begins in the logical design phase. In this phase, you design a logical architecture that represents the Java Enterprise System services and dependencies that satisfy the use cases you identified during the technical requirements phase.

The logical architecture, together with the system requirements document, characterize a *deployment scenario*. The logical architecture does not specify the actual hardware required to implement the deployment scenario.

For more information on the logical design phase, refer to Chapter 4, "Designing the Logical Architecture" on page 41.

Deployment Design Phase

In the deployment design phase, you create a *deployment architecture* that represents a mapping of the deployment scenario to a physical environment. The physical environment is the network infrastructure for the deployment and includes computing nodes, hardware requirements for each node, firewalls, and other devices on the network.

The process of mapping consists of sizing the deployment to specify the actual hardware needed to fulfill system requirements and determining a strategy to optimize the deployment architecture to meet budget considerations.

Approval for a deployment project typically follows the creation of the deployment architecture. During *project approval*, the cost of the deployment is assessed, and if approved, contracts for implementation of the deployment are signed, and resources to build the project acquired.

A detailed *design specification* is also part of the deployment design phase. The design specification provides details needed to implement the deployment architecture, such as the actual hardware, operating systems, network design, and other aspects of a physical environment. The detailed design specification also includes specifying directory services data structures needed to *provision users* for access to system services. Depending on the processes and policies for your deployment project, design specification occurs before or after project approval.

For more information on the deployment design phase, refer to Chapter 5, "Designing a Deployment Architecture" on page 51.

Implementation Phase

During the implementation phase, you build out the deployment architecture. Depending on the nature of your deployment project, this phase includes some or all of the following steps:

- Creating and deploying pilot and/or prototype deployments in a test environment
- Designing and running functional tests to measure compliance with system requirements
- Designing and running stress tests to measure performance under peak loads
- Creating a production deployment, which might be phased into production in stages

Once a deployment is in production, you need to continue to monitor, test, and tune the deployment to ensure that it fulfills the business goals.

For more information on the implementation phase, refer to Chapter 6, “Implementing a Deployment Design” on page 73.

Chapter 2

Business Analysis

This chapter provides some guidelines on how to analyze a business problem, identify its business requirements and constraints, and articulate a business goal.

Business analysis starts with stating the business goals for the deployment project. You then analyze the business problems you must solve and identify the *business requirements* that must be met to achieve the business goals. Consider also any *business constraints* that limit your ability to achieve the goals. The business requirements and constraints that you identify are a basis for a *business requirements document* that you later use to derive system requirements during the technical requirements phase.

No simple formula exists that identifies business requirements—you determine the requirements based on collaboration with your customer and on your own knowledge about the business domain. The guidelines presented here provide one way that you can begin your business analysis.

This chapter contains the following sections:

Business Requirements

A business problem statement is much like an executive summary of the project, outlining the project's ultimate goal. Within the business problem statement you make the business case for the project (why the project is necessary or desirable to do) and define the scope of the project (what is in bounds and out of bounds for the project). You also decide which features of the project are critical to its success.

The result of business requirements analysis should be a document that defines how a deployment satisfies the business goals. The following table lists topics typically addressed during business requirements analysis.

Table 2-1 Topics for Analyzing Business Requirements

Topic	Description
Business goals	Clearly articulate the goals of the project. A clear understanding of the goals helps focus design decisions. Here are a few example goals: • Enterprise collaboration, including features such as messaging, address book, instant messaging, and calendar services • Enterprise portal, to allow users to aggregate and personalize content, and to provide access to e-mail, calendar, instant messaging, and other enterprise services • Enterprise resource scheduler, to schedule conference rooms, offices, and other shared physical resources • Enable online commerce Contrasting the goals for a planned deployment with current operations can help later determine design decisions.
Type of deployment	Identify which of the following types of deployments you envision: • Business to Customer • Business to Employee • Business to Business • Enterprise Employee to Employee Communications • Some combination of these types Understanding the type of deployment brings to focus specific design issues inherent with that type.
Scope	Clearly state the scope of the project. Make sure you identify area that can be solved and avoid “open-ended” statements that make the goal either unclear or unreachable. Poorly defined scope can lead to deployment design that insufficiently addresses business needs.
Stakeholders	Identify individuals and organizations that have a vested interest in the success of the deployment. All stakeholders should actively participate in defining the business goals and requirements.
Critical qualities	Identify areas that are critical to success. This allows for analysis of the design with respect to the most important criteria.

Table 2-1 Topics for Analyzing Business Requirements *(Continued)*

Topic	Description
Target users	Identify the types of users the deployment targets. For example: • Current and previous employees • Active customers • Membership site • General public • Administrators
Benefits to the users	State the expected benefits to the users of the deployment. For example: • Remote access to company resources • Enterprise collaboration • Reduced response time • Reduced error rate • Simplification of daily tasks • Sharing of resources by remote teams • Increased productivity Clearly stating the expected benefits helps drive design decisions.
Service level agreements	Define the level and extent of customer support you must provide should the deployment fail to meet specific system requirements. Typically, a service level agreement is signed during project approval, based on service level requirements defined during analysis of technical requirements.
Security issues	Goals that you previously identified might have implicit security issues that you do not need to list in the problem statement. However, it can be useful to call out specific security goals essential to the deployment. For example: • Access to proprietary information to authorized users • Role-based access to confidential information • Secure communication between remote locations • Invocation of remote applications on local systems • Secure transactions with third party businesses
Priorities	State the priorities of your goals. Large and complex deployments might require phased implementation. Limited resources might require elimination or modification of some goals. By clearly stating the priorities, you can provide guidance to decisions that might need to be made for your deployment design to garner acceptance.

Business Constraints

Business constraints play a significant role in determining the nature of a deployment project. The key to a successful deployment design is finding the optimal way to meet business requirements within known business constraints.

The following table lists typical business constraints that might affect deployment design. Individual deployment projects might have business constraints particular to their own situation.

Table 2-2 Topics for Analyzing Business Constraints

Topic	Description
Time frame or schedule	The schedule for deployment can affect design decisions that you make. Aggressive schedules might result in scaling back of goals, changing priorities, or adopting an incremental solution approach. Within a schedule, there might be significant milestones that deserve consideration as well.
Budget considerations	Most deployments must adhere to a specific budget. This budget should always be considered during the design process to avoid cost overruns. When considering the budget, keep in mind not only the cost of completion of the project, but the resources required to maintain the project over a specific lifetime.
Resources	Consider all resources necessary for a successful deployment, not just the capital expenditures. This includes the following: • Existing hardware and network infrastructure Reliance on existing infrastructure can affect the design of a system. • Development resources needed to implement the deployment design Limited development resources, including hardware, software, and human resources, might suggest incremental deployment. You might have to reuse the same resources or development teams for each incremental phase. • Maintenance, administration, and support Analyze the resources available to administer, maintain, and support users on the system. Limited resources here might impact design decisions you make.
Cost of ownership	In addition to maintenance, administration, and support, you should analyze other factors that affect the cost of ownership. For example, hardware and software upgrades that might be necessary, the footprint on the power grid, telecommunications cost, and other factors influencing out-of-pocket expenses.

Table 2-2 Topics for Analyzing Business Constraints *(Continued)*

Topic	Description
Company standards and policies	Make sure to understand the standards and policies of the organization requesting the deployment. These standards and policies might affect technical aspects of the design, product selection, and methods of deployment.
Company change management	Company procedures for change management might dramatically affect the deployment methods and time table.
Return on investment	Each deployment should provide a return to the customer on their investment. Analysis of return on investment typically involves measuring the financial benefits gained from the expenditure of capital. Estimating the financial benefits of a deployment involves a careful analysis of the goals to be achieved by the deployment in comparison with alternate ways of achieving those goals, or in comparison with the cost of doing nothing at all.
Regulatory requirements	Regulatory requirements vary greatly, depending on the nature of the deployment.

Incremental Approach to Deployment

Typically, you view a deployment as a whole, comprehensive system. However, you often achieve the comprehensive system incrementally with measured steps.

The incremental approach provides these advantages:

- You can adapt to requirement changes due to business growth
- You can leverage existing infrastructure as you transition to your ultimate deployment implementation
- You can accommodate capital expenditure requirements
- You can leverage a small supply of human resources
- You can allow for partnership possibilities

When adopting an incremental approach, you typically design a road map that provides milestones that lead to the ultimate, comprehensive solution. Additionally, you might have to consider short term solutions for phases that will be implemented later in the plan.

No matter what approach you take, you should always design deployments to allow room for change and growth.

Chapter 3

Technical Requirements

This chapter discusses some of the processes and procedures that occur during technical requirements analysis.

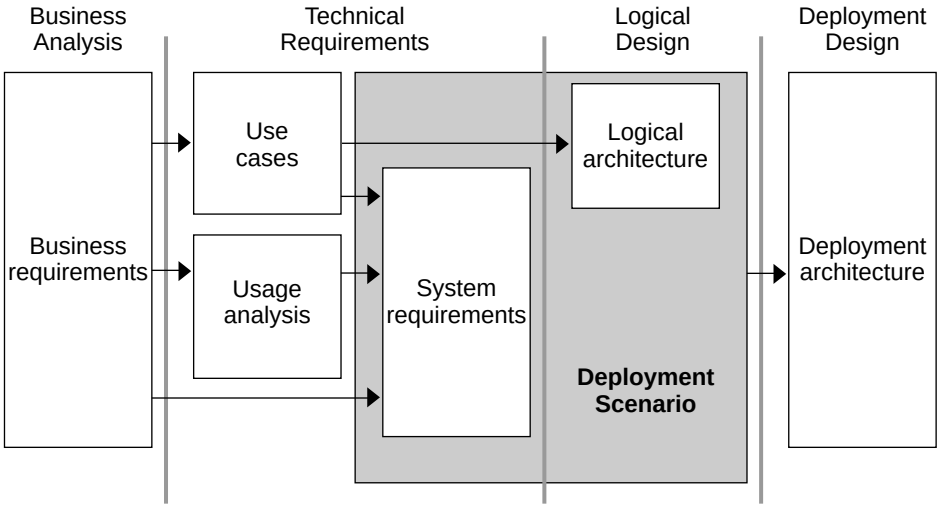
Technical requirements analysis begins with the business requirements documents created during the business analysis phase. Using the business requirements as a basis, you perform the following steps:

- Perform a *usage analysis* to aid in determining expected load on the deployment
- Create a set of *use cases* that model typical user interaction with the deployment
- Create a set of *system requirements* that are derived from the business requirements, use cases, and usage analysis

The use cases are also the basis for designing the *logical architecture* in the design phase. The logical architecture and the system requirements together form the *deployment scenario*, which later is an input to the deployment design phase.

The following figure shows the technical requirements phase in relation to the business analysis, logical design, and deployment design phases.

Figure 3-1 Technical Requirements Phase and Other Deployment Planning Phases



As with business analysis, no magic formula for technical requirements analysis exists that generates the usage analysis, use cases, and system requirements. Technical requirements analysis requires an understanding of the business domain, business objectives, and the underlying system technology.

This chapter contains the following sections:

- "Usage Analysis"
- "Use Cases" on page 30
- "System Requirements" on page 31

Usage Analysis

Usage analysis involves identifying the various users of the deployment you are designing and determining the usage patterns for those users. The information you gather provides an idea of the expected load conditions and is later used to determine performance requirements and other system requirements. Usage analysis information is also useful when assigning weights to use cases, as described in "Use Cases" on page 30.

During usage analysis you should interview users whenever possible, research existing data on usage patterns, and also interview builders and administrators of previous systems. The following table lists topics to consider when performing a usage analysis.

Table 3-1 Usage Analysis Topics

Topic	Description
Number and type of users	Identify how many users your deployment must support, and categorize those users, if necessary. For example: • A Business to Customer deployment might have a large number of visitors, but only a small number of users who register and engage in business transactions. • A Business to Employee deployment typically has to accommodate each employee, but some might need access from outside the corporate network. • In a Business to Employee deployment, managers might need authorization to areas that regular employees should not be able to access.
Active and inactive users	Identify the usage patterns and ratios of active and inactive users. Active users are those users logged into the system and are interacting with the system's components. Inactive users can be users who are not logged in or users who log in but do not interact with the system's components.
Administrative users	Identify users that access the deployed system to monitor, update, and support the deployment. Determine any specific administrative usage patterns that might affect system requirements. For example, administration of the deployment from outside the firewall.
Usage patterns	Identify how users of various types will access the system and provide targets for expected usage. For example: • Are there peak times when usage spikes? • What are the normal business hours? • Are users distributed globally? • What is the expected duration of user connectivity?
User growth	Determine if the size of the user base is fixed or if the deployment expects growth in the number of users. If the user base is expected to grow, try to create reasonable projections of the growth.

Table 3-1 Usage Analysis Topics *(Continued)*

Topic	Description
User transactions	Identify the type of user transactions that must be supported. These user transactions can be translated into use cases. For example: • What tasks will users perform? • When users log in, do they remain logged in? Or do they typically perform a few tasks and log out? • Will there be significant collaboration between users that requires common calendars, web-conferences, and deployment of internal web pages?
User studies and statistical data	Use pre-existing user studies and other sources to determine patterns of user behavior. Often, enterprises or industry organizations have user research studies from which you can extract useful information about users. Log files for existing applications might contain statistical data useful in making estimates for a system.

Use Cases

Use cases model typical user interaction with the deployment you are designing, describing the complete flow of an operation from the perspective of an end user. Prioritizing design around a complete set of use cases ensures a continual focus on the delivery of expected functionality.

Each use case can include quantitative estimates about user behavior, which you can later use to determine system requirements for performance, availability, and other qualities of service. Use cases are also the starting point for designing the logical architecture, as described in Chapter 4, “Designing the Logical Architecture” on page 41.

You often assign relative weights to use cases, with the highest weighted use cases representing the most common user tasks. The weighting of use cases helps to determine system requirements.

Use cases can be described at two levels.

- Use case diagrams

  Graphical depiction of the relationships among actors and use cases.

- Use case reports

  Descriptions of individual use cases, including primary and alternative flows of events.

System Requirements

System requirements describe the quality of service a deployed system must provide to meet the business requirements arrived at through business analysis. You typically use the usage analysis and use cases together with the business requirements to derive system requirements.

The following table lists system qualities that are often used to specify system requirements.

Table 3-2 System Qualities Affecting Deployment Design

System Qualities	Description
Availability	A measure of how often a system's resources and services are accessible to end users, often expressed as the *uptime* of a system.
Latent Capacity	The ability of a system to handle unusual peak load usage without additional resources.
Performance	The measurement of response time and latency with respect to user load conditions.
Scalability	The ability to add capacity (and users) to a deployed system over time. Scalability typically involves adding resources to the system but should not require changes to the deployment architecture.
Security	A complex combination of factors that describe the integrity of a system and its users. Security includes authentication and authorization of users as well as the secure transport of information.
Serviceability	The ease by which a deployed system can be administered, including tasks such as monitoring the system, repairing problems that arise, and upgrading hardware and software components.

The system qualities that affect deployment design are closely interrelated. Requirements for one system quality might affect the requirements and design for other system qualities. For example, higher levels of security might affect performance, which in turn might affect availability. Adding additional servers to address availability issues might affect maintenance costs (serviceability).

Understanding how system qualities are interrelated and the trade-offs that must be made is the key to designing a system that successfully satisfies both business requirements and business constraints.

The following sections take a closer look at the system qualities that affect deployment design, providing guidance on factors to consider when formulating system requirements. There is also a section on service level requirements, which are a special set of system requirements used to create service level agreements.

Availability

Availability is a way to specify the *uptime* of a deployed system. It is typically measured as the percentage of time that the system is accessible to users. The time the system is not accessible (*downtime*) can be due to the failure of hardware, software, the network, or any other factor (such as loss of power) that causes the system to be down. In most cases, scheduled time for service (maintenance and upgrades) is not considered downtime.

Typically you measure availability by the number of "nines" you can achieve. For example, 99% availability is two nines. Specifying additional nines significantly affects the deployment design for availability. The following table displays the result of adding additional nines of availability to a system that is running 24x7 year-round, which is a total of 8,760 hours.

Table 3-3 Downtime for a System Running Year-round (8,760 hours)

Nines	Percentage Available	Downtime
Two	99%	88 hours
Three	99.9%	9 hours
Four	99.99%	45 minutes
Five	99.999%	5 minutes

Fault Tolerant Systems

Availability requirements of four or five nines typically require a system that is *fault tolerant*. A fault tolerant system must be able to continue service even during a hardware or software failure. Typically, fault tolerance is achieved by redundancy in both hardware (such as CPUs, memory, and network devices) and software providing key services.

A *single point of failure* is a hardware or software component that is not backed up by redundant components. The failure of this component results in the loss of service for the system. When designing a fault tolerant system, you must identify potential single points of failure and eliminate them.

Fault tolerant systems can be expensive to implement and maintain. Make sure you understand the nature of the business requirements for availability and consider the strategies and costs of availability solutions that meet those requirements.

Sun Cluster 3.1 4/04

Sun Cluster 3.1 4/04 software provides a high availability solution for deployments that require a highly available, fault tolerant system. Sun Cluster 3.1 4/04 couples servers, storage, and other network resources to provide a failover process that is achieved quickly and with little interruption of services to the users of the system.

Prioritizing Availability of Services

From a user perspective, availability often applies more to each service provided by the deployed system rather than the availability of the entire system. For example, if instant messaging services become unavailable, there usually is little or no impact on the availability of other services. However, the availability of services upon which many other services depend (such as Directory Server) has a wider impact on the system.

It is helpful to list availability needs according to an ordered set of priorities. The following table prioritizes the availability of different types of services.

Table 3-4 Prioritizing Availability of Services

Priority	Service Type	Description
1	Strategic	Services essential to operation. For example, many services depend on Directory Server.
2	Mission critical	Services that must be available at peak load. For example, database services to applications defined as mission critical.
3	Must be available	Services that must be available, but can be available at reduced performance. For example, Messaging Server availability might not be critical in some business environments.
4	Can be postponed	Services that must be available within a given time period. For example, Instant Messaging availability might not be essential in some business environments.
5	Optional	Services that can be postponed indefinitely.

For information on various design strategies to implement availability requirements, refer to "Sizing for Availability" on page 61.

Latent Capacity

Latent capacity is the ability of a deployment to handle unusual peak load usage without the addition of resources. Typically, you do not specify system requirements directly around latent capacity, but this system quality is a factor in determining availability, performance, and scalability requirements.

Performance

Determining performance requirements is the process of translating business requirement expectations on performance into system requirements. The business requirement typically expresses performance in non-technical terms that specify response time. For example, a business requirement for web-based access might state the following:

> Users should expect a reasonable response time upon login, typically no greater than four seconds.

Starting with this business requirement, examine all use cases to determine how to express this requirement at a system level. Take into account the user load conditions, as determined during usage analysis. Express the performance requirement for each use case in terms of *response time under specified load conditions* or *response time plus throughput*. You might also specify the allowable number of errors.

Here is one example of how to specify system requirements for performance.

> Response for user login must be no greater than four seconds throughout the day, measured at 15 minute intervals, with fewer than 3.4 errors per million transactions.

The performance requirements are closely related to availability requirements (how failover impacts performance) and latent capacity (how much capacity there is to handle unusual peak loads).

Scalability

Scalability describes the ability to add capacity and users to a system over time. Scalability usually requires the addition of resources, but should not require changes in the design of the deployment architecture or loss of service due to the time required to add additional resources.

As with availability, scalability applies more to individual services provided by a system rather than to the entire system. However, for services upon which other services depend, such as Directory Server, scalability can have system-wide impact.

You do not necessarily specify scalability requirements with system requirements unless projected growth of the deployment is clearly stated in the business requirements. During the deployment design phase, the deployment architecture should account for scaling the system even if you do not specify scalability requirements.

Determining scalability requirements is not an exact science. Estimating the growth of a system involves projections, estimates, and guesses that might not be fulfilled. Here are three keys to building a scalable system.

- Adopt a strategy of high performance design.

  During the specification and design of performance requirements, include latent capacity to handle loads that might increase over time. Also, maximize availability within the budget constraints. This strategy allows you to absorb growth and better schedule milestones for scaling the system.

- Implement your deployment in stages.

  Incremental implementation helps with scheduling the addition of resources.

- Implement extensive performance monitoring.

  Monitoring performance of a deployment helps determine when to add resources to the deployment.

The following table lists some topics to consider for scalability.

Table 3-5 Scalability Considerations

Topic	Description
Usage analysis	Understand the usage patterns of the current (or projected) user base by studying existing data. In the absence of current data, analyze industry data or market estimates.

Table 3-5 Scalability Considerations *(Continued)*

Topic	Description
Design for reasonable maximum scale	Design with a goal towards the maximum required scale for both known and possible demand. Often, this is a 24 month estimate based on performance evaluation of the existing user load and reasonable expectations of future load. The time period for the estimate depends largely on the reliability of projections.
Set appropriate milestones	Implement the deployment design in increments to meet short term requirements with a buffer to allow for unexpected growth. Set milestones for adding system resources. For example: • Capital acquisition Such as quarterly or yearly • Hardware lead time For example, one to six weeks • Buffer (10% to 100%, depending on growth expectations)
Incorporate emerging technology	Understand emerging technology, such as faster CPUs and Web servers, and how that can affect the performance of the underlying architecture.

Security Requirements

Security is the quality of a system that affects the integrity of the system and its users, including the integrity of the user's transactions and associated data. As with other system requirements, the business requirements, usage analysis, and use cases drive the analysis for security requirements.

Analysis for security requirements fall under the following categories:

- Authentication
- Authorization
- Identity Management

Authentication, authorization, and identity management, together with an enterprise-wide policy for enforcement of sound security practices, provide confidence that transactions are secure and that data stored on a site cannot be compromised.

NOTE Security requirements affecting the integrity of the infrastructure (for example firewall software and network design) are typically not considered during system requirements analysis. Instead, these security issues come into play during deployment design.

Authentication

Authentication is how users identify themselves to a system and also how the system identifies itself to the users. Authentication is a key part of the system integrity that protects the system from unauthorized access.

You should understand user requirements to select the best authentication scheme for the deployment. For example, a Business to Customer deployment might allow users to register using a username/password combination. These users rely on a server certificate issued by a trusted certificate authority, such as VeriSign, to authenticate the selling system over a secure transport.

A Business to Employee deployment might instead provision employees from an existing user base. From within the company firewall, access is allowed to known secure locations. From outside the firewall, access to secure locations is through proxies that perform the authentication and redirection inside the company firewall.

Authorization

Authorization is the recognition of specific privileges to authenticated users. For example, users with administrator authority have access to parts of a deployed system that are inaccessible to ordinary users.

Authorization also plays a role in deployments implementing single sign-on (SSO). Authenticated users to a deployment can have access to multiple services without having to sign on more than once.

Identity Management

A deployed system must have a way to add, modify, or delete users who will be accessing system services. Depending on your needs, identity management can be accomplished by an authorized administrator or by the users themselves by means of a *delegated administration* interface. Deployments for medium or large enterprise should consider a delegated administration design. Delegated administration improves customer satisfaction and reduces the costs of system administration.

Serviceability Requirements

Serviceability is the ease by which a deployed system can be administered, including tasks such as monitoring the system, repairing problems that arise, and upgrading hardware and software components.

When planning requirements for serviceability, consider the topics listed in the following table.

Table 3-6 Topics for Serviceability Requirements

Topic	Description
Downtime Planning	Identify maintenance tasks that require specific services to be unavailable or partially unavailable. Some maintenance and upgrades can occur seamlessly to users, while others require interruption of service. When possible, schedule with users those maintenance activities that require downtime, allowing the users to plan for the downtime.
Usage Patterns	Identify the usage patterns of a deployment to determine windows of opportunity for maintenance. For example, on systems where peak usage is normally during normal business hours, the windows of opportunity occur in the evening or weekends. For geographically distributed systems, identifying these times can be more challenging.
Availability	Serviceability is often a reflection of your availability design. Strategies for minimizing downtime for maintenance and upgrades revolve around your availability strategy. Systems that require a high degree of availability have smaller windows for maintenance, upgrades, and repair. Strategies for handling availability requirements affect how you handle maintenance and upgrades. For example, on systems that are distributed geographically, servicing can depend on the ability to route workloads to remote servers during maintenance periods. Also, systems requiring a high degree of availability might require more sophisticated solutions that automate restarting of systems with little human intervention.
Diagnostics and Monitoring	You can improve the stability of a system by regularly running diagnostic and monitoring tools to identify problem areas. This can avoid problems before they occur, help balance workloads according to availability strategies, and improve planning for maintenance and downtime.

Service Level Requirements

Service level requirements are a set of system requirements that specify the conditions under which customer support must be provided. Service level requirements are the basis for service level agreements, which are typically signed during project approval.

As with system requirements, service level requirements derive from business requirements and represent a kind of guarantee to the customer about the overall system quality that the deployment must meet. Because the service level agreement is a contract between you and the customer, there should be no ambiguity in the specification of service level requirements. The service level requirements define exactly under what conditions the requirements are tested and precisely what constitutes failure to meet the requirements.

Chapter 4

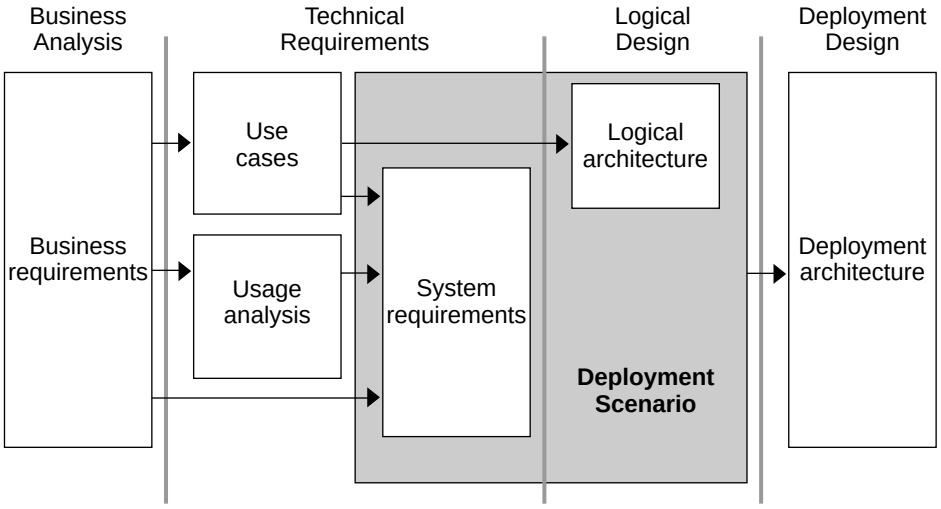
Designing the Logical Architecture

This chapter discusses a process for creating a logical architecture and provides an example of the process using a set of use cases representative of those found in a communications deployment for a medium-sized enterprise.

The logical architecture identifies the Java Enterprise System components (and their dependencies) that provide the software services needed to meet the business goals of a deployment. Typically, use cases developed during the technical requirements phase indicate which software services are required. However, the information on software services can often be obtained directly from the business requirements derived during the business analysis phase.

The logical architecture, together with system requirements determined during requirements analysis, represent a deployment scenario. The deployment scenario is the basis for designing the deployment architecture. The following figure shows the relationship of the logical design phase to the business analysis, technical requirements, and deployment design phases.

Figure 4-1 Logical Design in Relation to Other Deployment Planning Phases



This chapter contains the following sections:

Deployment Planning Example

To help illustrate the deployment planning process, this section introduces use cases for an example deployment based on the communications needs of a typical medium-sized enterprise. This example deployment continues in later chapters of this white paper to illustrate various steps of deployment planning.

CAUTION The use cases, logical architecture, deployment architecture, and design specification for the example deployment are simplified versions of steps in a deployment planning process.

The example has been simplified for illustrative purposes. The design for the example is incomplete and has never been built or tested. Do not use the example as a blueprint for any deployment you are planning.

The example deployment begins with a set of use cases that are derived from typical business requirements for a communications deployment. The following table summarizes these use cases.

Table 4-1 Use Cases for Example Deployment

Use Case	Description
#1 Single sign-on	From Web browser, user logs on to system (username/password) to access enterprise services, which could be any of the following: • Custom Portal Web page • Web-based e-mail page • Calendar interface • Secure Web page
#2 Open personal portal screen	From Web browser, user navigates to personal portal screen.
#3 Through portal, user checks e-mail	From portal interface, user checks for new e-mail messages.
#4 Through portal, user checks secure web page	From personal portal interface, user checks a secure project status page.
#5 Through portal, user checks calendar	From portal interface, user checks daily appointments.
#6 Manage calendar	From Web-based calendar client, user schedules appointments.
#7 Manage e-mail	From e-mail client, user reads and sends e-mail.

From these use cases, you can derive the services needed for the logical architecture, as described in following sections.

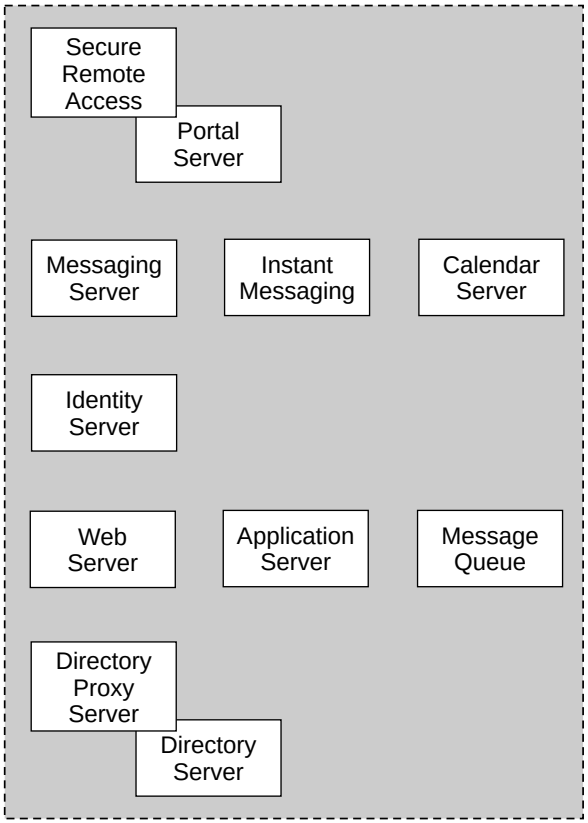
Java Enterprise System Services

Design of the logical architecture begins with an analysis of the use cases, which should help you determine the services required for the deployment. Using your knowledge of Java Enterprise System and previous design experience, lay out an initial logical design of Java Enterprise System components that provide the services identified by the use cases.

When laying out the components, consider the logical flow of data within the system and the dependencies between the components that provide the services. Your logical design should reflect these dependencies, which impact the flow of data between components in the design.

The following figure shows components provided with Java Enterprise System. Use this figure with Table 4-2 on page 45 to understand the interdependencies of Java Enterprise System components. In general, components at the bottom of the figure provide support to components above them.

Figure 4-2 Java Enterprise System Components



The following table lists the actual interdependencies among Java Enterprise System components.

Table 4-2 Java Enterprise System Component Interdependencies

Java Enterprise System Component	Provides Support To	Depends On
Application Server	Identity Server Portal Server	Message Queue
Calendar Server	Portal Server (for calendar channel)	Directory Server Identity Server (for single sign-on) Messaging Server (for Calendar Server e-mail notification service)
Directory Proxy Server	None	Directory Server
Directory Server	Administration Server Calendar Server Directory Proxy Server Identity Server Instant Messaging Messaging Server Portal Server	None
Identity Server	Portal Server If configured for single sign-on: Calendar Server Instant Messaging Messaging Server	Directory Server Application Server or Web Server
Instant Messaging	Portal Server	Directory Server
Message Queue	Application Server	Directory Server (optional)
Messaging Server	Calendar Server Portal Server (for messaging channel)	Directory Server Web Server Identity Server
Portal Server	Secure Remote Access	Directory Server Application Server or Web Server If configured to use Portal Server Channels: Calendar Server Messaging Server Instant Messaging
Secure Remote Access	None	Portal Server
Web Server	Identity Server Portal Server	None

For example, to lay out the Java Enterprise System components for the example communications deployment, analyze the use cases listed in Table 4-1 on page 43. The following table lists the components directly required for the deployment, as indicated by the use cases.

Table 4-3 Java Enterprise System Components to Support Example Use Cases

Java Enterprise System Component	Use Cases
Portal Server	#1 Single sign-on
	#2 Open personal portal screen
	#3 Through portal, user checks e-mail
	#4 Through portal, user checks secure web page
	#5 Through portal, user checks calendar
Calendar Server	#1 Single sign-on
	#5 Through portal, user checks calendar
	#6 Manage calendar
Messaging Server	#1 Single sign-on
	#3 Through portal, user checks e-mail
	#7 Manage e-mail

You also need to determine which Java Enterprise System components are needed to support the components listed in Table 4-3 above. The following table lists these additional components.

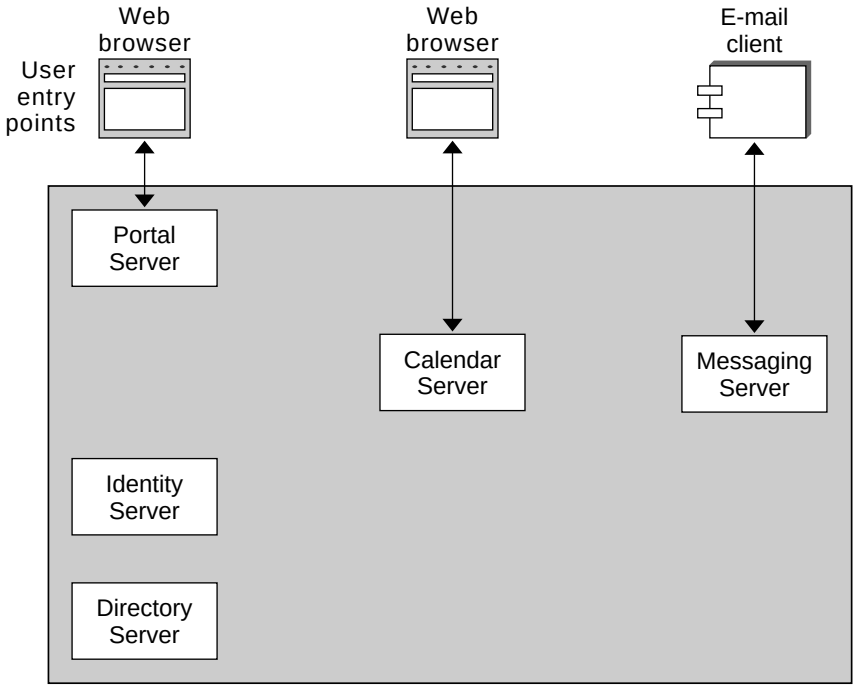
Table 4-4 Additional Components to Support Example Use Cases

Java Enterprise System Component	Support Provided
Identity Server	Provides support to Portal Server.
	Provides single sign-on support to Calendar Server and Messaging Server.
Directory Server	Provides support to Identity Server and Portal Server.
Application Server or Web Server	Provides support to Identity Server and Portal Server. (Identity Server and Portal Server must run inside a web container.)

Logical Architecture for the Example Deployment

Figure 4-3 below shows the layout of components for the example deployment, indicating user entry points to the deployment. The figure places the service requiring the most support (Portal Server) at the top and lists supporting components beneath it, roughly reflecting the dependencies between the components (as described in Table 4-2 on page 45). The figure does not depict the component providing the web container to support Portal Server and Identity Server because this dependency does not reflect the flow of data in the deployment.

Figure 4-3 Java Enterprise System Components in a Logical Architecture

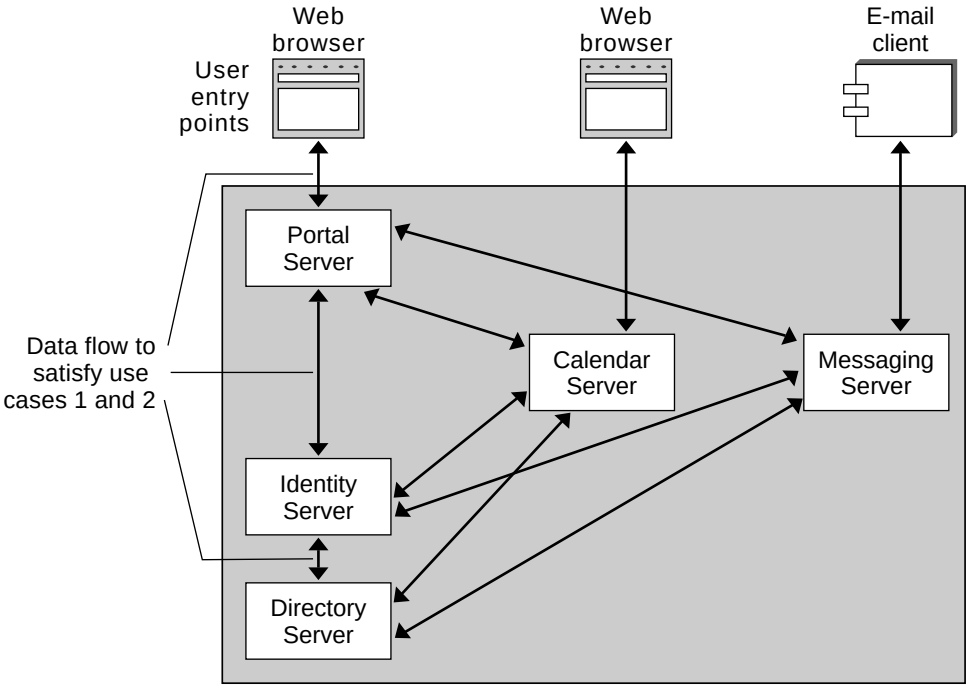


Data Flow for the Example Deployment

Study the use cases to determine the logical flow of data between services in the logical architecture, and indicate this flow in the layout. The flow of data between the services in a system plays an important role when sizing for performance and availability, as described in "Sizing a Planned Deployment" on page 52.

The following figure depicts the flow of data for the example deployment. The data flow is determined from the use cases for the deployment as well as the Java Enterprise System service dependencies.

Figure 4-4 Logical Flow of Data for the Example Deployment



Logical Architecture for Example Deployment

Figure 4-4 calls out the data flow that satisfies both use cases 1 and 2. This data flow represents the following:

- User login request from the Web-based client
- Portal Server's dependency on Identity Server to provide authentication services

- LDAP information provided by Directory Server to Identity Server

The remaining data flows in Figure 4-4 are similarly derived from the use cases and server dependencies.

Deployment Scenario

The completed logical architecture design and the system requirements derived during requirements analysis constitute the deployment scenario. The deployment scenario is the starting point for designing the deployment architecture, as explained in Chapter 5, "Designing a Deployment Architecture."

Chapter 5

Designing a Deployment Architecture

This chapter provides information on how to design a deployment for performance, security, availability and other system qualities. The chapter also provides information on optimizing the deployment design.

A deployment architecture depicts the mapping of a logical architecture to a physical environment. The physical environment includes the computing nodes in an intranet or Internet environment, CPUs, memory, storage devices, and other hardware and network devices.

Designing the deployment architecture involves *sizing the deployment* to determine the physical resources necessary to meet the system requirements specified during the technical requirements phase. You also *optimize resources* by analyzing the results of sizing the deployment to create a design that provides the best use of resources within business constraints.

After a deployment architecture design is complete the actual cost of the deployment is assessed during *project approval*. Once the project is approved, contracts for completion of the deployment need to be signed and resources to implement the project acquired.

A *detailed design specification* occurs before or after project approval. The detailed design specification is used in the implementation phase to build out the design.

This chapter continues using the example deployment from Chapter 4 to illustrate various steps in the process of designing a deployment architecture.

This chapter contains the following sections:

Sizing a Planned Deployment

Sizing a planned deployment is the process of determining the set of hardware resources necessary to fulfill the system requirements and ultimately satisfy the business goals. As with other aspects of planning and designing a deployment, sizing is not an exact science and cannot be prescribed with formulas and recipes. Successful sizing is the result of a combination of past design experience, knowledge of systems architecture, domain knowledge, and applied creative thinking.

Sizing revolves around the system requirements you previously determined for the following system qualities, as described in "System Requirements" on page 31. The business requirements, usage analysis, and use cases from the earlier phases of deployment design also play a role in sizing a system.

When performing a sizing exercise, the use cases and usage analysis help determine the resources necessary to support the use cases. You typically start with the heaviest weighted use cases (representing the most common transactions) and proceed to the least weighted ones. This use of weighted use cases helps allocate resources according to the expected stress on the system.

The following sections provide some general guidance on how to size a deployment for the following system qualities:

- Performance
- Security
- Availability
- Serviceability

Sizing for Performance

Sizing for performance and load requirements is an iterative process that estimates the number of CPUs and corresponding memory required to support the services in the deployed system. When estimating the number of CPUs required to support a service, consider the following:

- Use cases and corresponding usage analysis that apply to the service
- System requirements determined during analysis for technical requirements
- Past experience with the Java Enterprise System components providing the service

- Consultation with Sun professional services, who have experience with sizing various types of deployment scenarios

The process of sizing for performance typically consists of the following steps. The ordering of these steps is not critical—it simply provides a way to consider the factors that affect the final result.

1. Determine a baseline CPU estimate for components identified as user entry points to the system.
2. Make adjustments to the CPU estimates to account for dependencies between components.
3. Make adjustments to the CPU estimates to reflect security, availability, scalability, and latent capacity requirements.

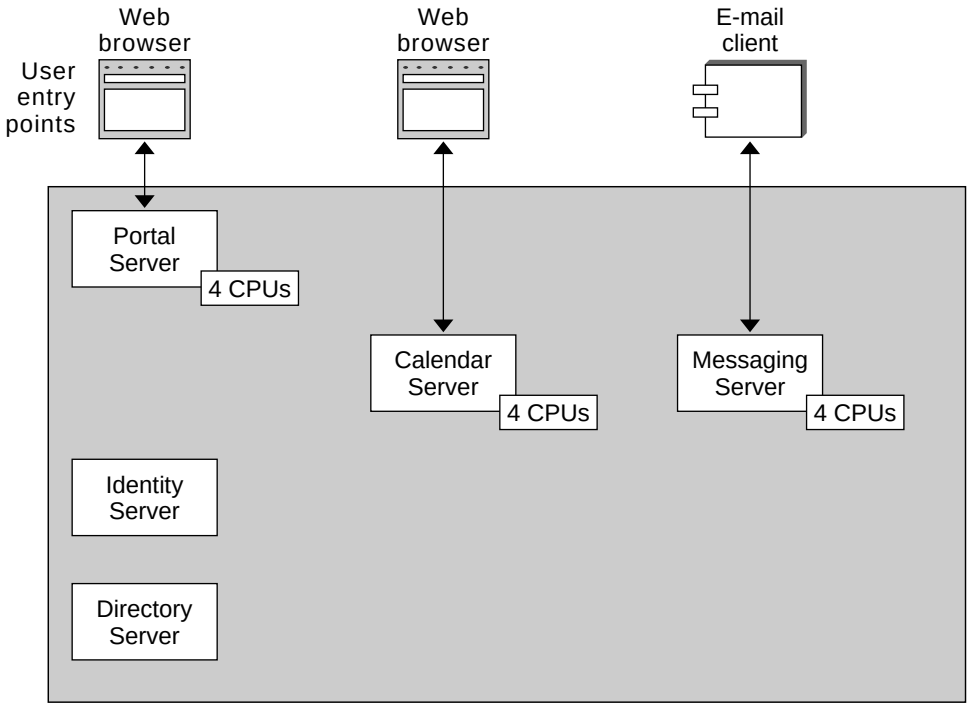
Determine Baseline CPU Estimate for User Entry Points

Begin by estimating the number of CPUs required to handle the expected load on each component that is a user entry point. Note the estimates on your layout design of the logical architecture.

The following figure uses the example deployment introduced in "Deployment Planning Example" on page 42, depicting initial CPU estimates for components that are user entry points. These estimates represent figures that might result from analysis of the system requirements, use cases, and usage analysis.

CAUTION This white paper does not instruct you on the specifics of sizing for performance. The CPU and memory figures used in this manual are arbitrary estimates for illustration only. They do not represent any specific implementation advice, other than to illustrate a process you might use when designing a system.

Figure 5-1 Baseline CPU Estimates for Components Providing User Entry Points



Logical Architecture for Example Deployment

Adjust CPU Estimates for Service Dependencies

The components providing user entry points require support from other Java Enterprise System services. To continue specifying performance requirements, adjust the performance estimates to take into account support required from other components.

In the example, examine the logical flow of data, as illustrated in Figure 4-4 on page 48, and make adjustments for components providing support to other components. The following table summarizes the adjustments to the CPU estimates. In your estimates, you can specify fractional CPUs. When performance estimates are complete, the CPU counts are totaled and rounded up.

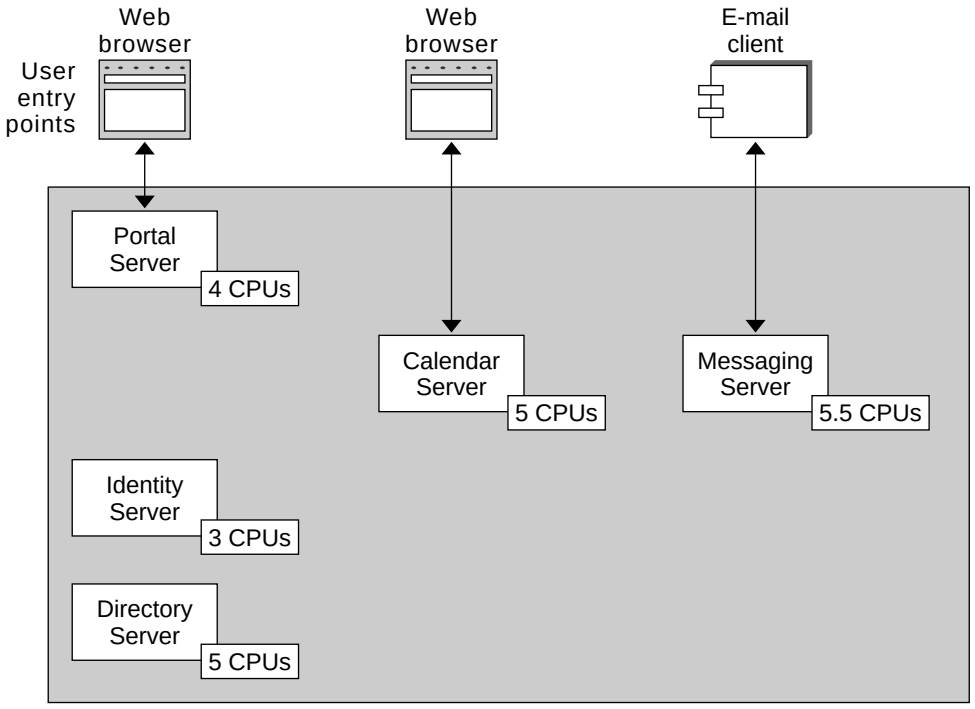
As with the estimates in the previous section, the performance estimates in the following table are arbitrary values for illustration purposes only.

Table 5-1 CPU Estimates for Supporting Services

Service	Estimate	Description
Portal Server	None	Does not provide support to other services.
Calendar Server	1 CPU	Provides support to: • Portal Server's calendar channel
Messaging Server	1.5 CPUs	Provides support to: • Portal Server's messaging channel • Calendar Server's e-mail notification service
Identity Server	3 CPUs	Provides support to: • Portal Server • Calendar Server • Messaging Server
Directory Server	5 CPUs	Provides support to: • Identity Server • Calendar Server • Messaging Server

The following figure updates the estimates for performance, based on the information in Table 5-1.

Figure 5-2 CPU Estimates adjusted for Supporting Services



Logical Architecture for Example Deployment

Adjust CPU Estimates for Latent Capacity, Scalability, and Availability

Once you complete sizing estimates for performance, round up the figures for CPUs. Typically, you round up CPUs to the next even number. When rounding up CPU estimates, consider the following factors:

- Latent capacity

  Increase CPU estimates to improve the ability to handle peak loads.

- Scalability

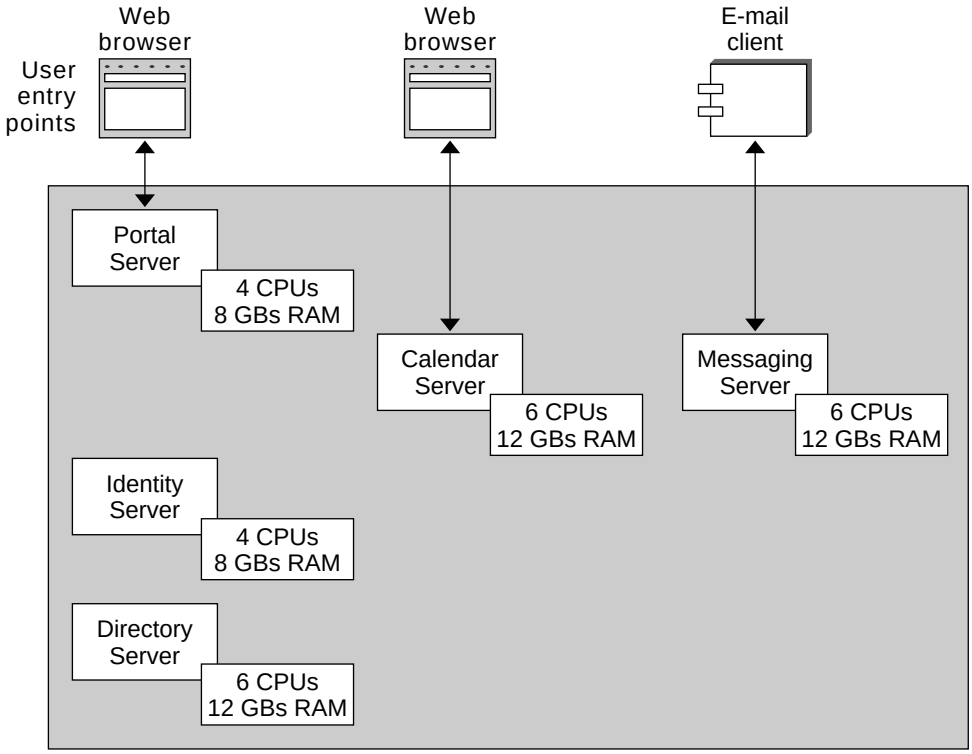
  Increase CPU estimates to make sure your deployment does not need to be scaled prematurely. Look at the anticipated milestones for scaling and projected load increase over time to make sure you allow enough latent capacity to reach the milestones.

- Availability

  Adjust CPU estimates to account for replication of services that might be required for availability or failover.

The following figure adjusts the CPU estimates for the example deployment. The figure also specifies memory requirements for each CPU. The example assumes each CPU requires 2GB of memory. These memory specifications for the example are arbitrary figures for illustration purposes. The calculation of memory required for each CPU is beyond the scope of this white paper.

Figure 5-3 Performance Figures Including Memory Requirements



Logical Architecture for Example Deployment

Sizing for Security

When sizing a deployment, security issues become a factor in the following ways:

- Secure transport of data
- Authentication of users

Secure transport of data involves handling transactions over a secure transport protocol such as Secure Sockets Layer (SSL). Authentication of users can also require handling transactions over a secure transport.

Transactions handled over a secure transport typically require additional computing power to first, establish a secure session (known as the *handshake*) and second, to encrypt and decrypt transported data. Depending on the encryption algorithm used (for example, 40-bit or 128-bit encryption algorithms), the additional computing power can be substantial.

For secure transactions to perform at the same level as non-secure transactions, you must plan for additional computing power. Depending on the nature of the transaction, and the Java Enterprise System services that handle it, secure transactions might require four times (or more) computing power.

When estimating the performance requirement to handle secure transactions, first analyze use cases to determine the percentage of transactions that require secure transport. If the performance requirements for secure transactions are the same as for non-secure transactions, modify the CPU estimates to account for the additional computing power needed for the secure transactions.

In some usage scenarios, secure transport might only be required for authentication. Once a user is authenticated to the system, no additional security measures for transport of data is required. In other scenarios, secure transport might be required for all transactions. An estimate of five to ten percent of transactions requiring secure transport is reasonable in many cases.

For example, when browsing a product catalog for an online e-commerce site, all transactions can be non-secure until the customer has finished making selections and is ready to “check out.” Additionally, many of these e-commerce sites relax the latent response requirement for secure transactions. However some usage scenarios, such as deployments for banks or brokerage houses, require most, if not all, transactions to be secure and apply the same performance standard for both secure and non-secure transactions.

Calculating Performance for Secure Transactions

This section continues the example deployment to illustrate a worksheet for calculating CPU requirements for a use case that includes both secure and non-secure transactions.

To calculate the CPU requirements, in the worksheet make the following calculations:

1. Start with a baseline figure for the CPU requirements, such as you calculated in the previous section, "Sizing for Performance" on page 52.
2. Calculate the percentage of transactions that require SSL, and compute the CPU requirements for the SSL transactions.
3. Adjust the CPU calculations for the non-secure transactions.
4. Tally the secure and non-secure requirements to calculate the total CPU requirements.

The worksheet in Figure 5-4 is based on additional use cases and usage analysis for the Portal Server. The additional use cases and usage analysis assume the following:

- All logins require secure authentication
- All logins account for 10% of the total Portal Server load
- The performance requirement for secure transactions equals the performance requirement for non-secure transactions.

For the purposes of this example, to account for the extra computing power to handle SSL transactions, the number of CPUs to handle these transactions will be increased by a factor of five. As with other CPU figures in the example, this is an arbitrary figure for illustration purposes only.

Figure 5-4 Worksheet for Calculating CPU Estimates for Secure Transactions

Baseline estimate for all Portal Server transactions: 4 CPUs

1. Calculate CPU estimates for SSL transactions:
Ten percent require SSL
`.10 x 4 = 4`
SSL transactions require 5x CPU power
`5 x .4 = 2 CPUs`
	2 CPUs

2. Adjust CPU estimates for non-SSL transactions: Ninety percent are nonsecure `.9 x 4 = 3.6 CPUs required`	3.6 CPUs

3. Total CPU estimate:
`SSL 2`
`Non-SSL 3.6`
`5.6 CPUs`
	5.6 CPUs

Specialized Hardware to Handle SSL Transactions

Specialized hardware devices, such as SSL accelerator cards and other appliances, are available to provide computing power to handle establishment of secure sessions and/or encryption and decryption of data. When using specialized hardware for SSL operations, computational power is dedicated to some part of the SSL computations, typically the “handshake” operation that establishes a secure session.

This hardware might be of benefit to your final deployment architecture. However, because of the specialized nature of the hardware, it is best to estimate secure transaction performance requirements first in terms of CPU power, and then consider the benefits of using specialized hardware to handle the additional load.

Some factors to consider when using specialized hardware are whether the use cases support using the hardware (for example, use cases that require a large number of SSL “handshake” operations) and the added layer of complexity this type of hardware brings to the design. This complexity includes the installation, configuration, testing, and administration of these devices.

Sizing for Availability

After you complete sizing for performance you can begin sizing your system for availability. This is where you designate specific servers to host the components in the logical architecture and design load balancing, redundancy, and failover strategies for the various Java Enterprise System components.

Study the use cases and usage analysis to determine which availability solutions to consider. The following items are examples of the type of information you gather to help determine availability strategies:

- How many nines of availability are specified?
- Are there performance specifications specific to failover situations (for example, at least 50% of performance during failover)?
- Do the use cases and usage analysis identify times of peak and non-peak usage?
- What are the latent performance requirements?
- Are there geographical considerations?

For each component, analyze the use cases to determine a best-fit solution for failover and load balancing requirements. Also, consider the use cases and usage analysis to determine the best way to load balance services.

The availability strategy you choose must also take into consideration serviceability requirements, as discussed in "Serviceability Issues" on page 66. Try to avoid complex solutions that require considerable administration and maintenance in favor of systems that are easy to manage.

Directory Design for Complex Systems

Complex deployments for a large number of users might require a directory design for Directory Server that can affect the availability strategy. This is because the LDAP directory design might affect availability strategy for Identity Server and Messaging Server, which in turn might affect other system qualities.

If you are designing a complex deployment, consider creating a preliminary directory design to aid in the availability design. Later, during detailed design specification or development phases, provide the complete directory design.

Hardware and Software Failures

Your availability design should provide protections for both hardware and software failures. Software failure typically has a higher cost than hardware failure. There is higher mean time between software failures than between hardware failures. Additionally, software failures are harder to diagnose and repair and require higher administration and maintenance costs to prevent.

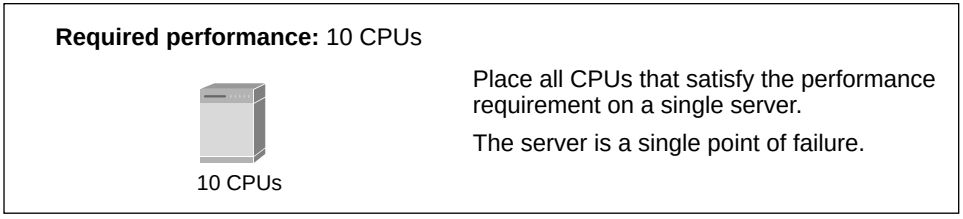
General Approaches to Availability

This section provides some general ways you can design for availability requirements. Specific availability designs are outside the scope of this white paper.

Single Server System

Place all your computing resources for a service on a single server. If the server fails, the entire service fails.

Figure 5-5 Single Server



Sun provides high-end servers that provide the following benefits:

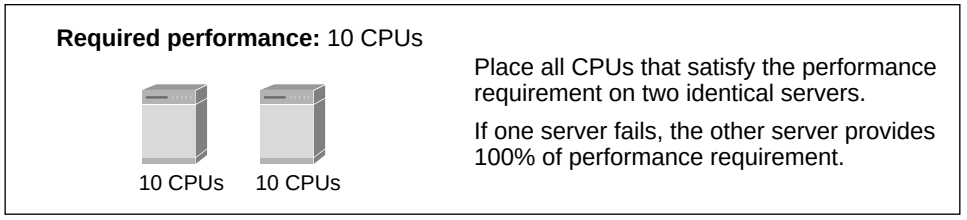
- Replacement and reconfiguration of hardware components while the system is running
- Running multiple applications in fault-isolated domains on the server
- Upgrading capacity, performance speed, and I/O configuration without rebooting the system

A high-end server typically costs more than a comparable multi-server system. However, a single server provides savings on administration, monitoring, and hosting costs for servers in a data center. However, load balancing, failover, and removal of single points of failure is more flexible with multi-server systems.

Horizontally Redundant Systems

There are several ways to increase availability with parallel redundant servers that provide both load balancing and failover. The following figure illustrates two replicate servers providing an *N+1 availability* system. An N+1 system has an additional component to provide 100% capacity should one server fail.

Figure 5-6 Two Replicate Servers

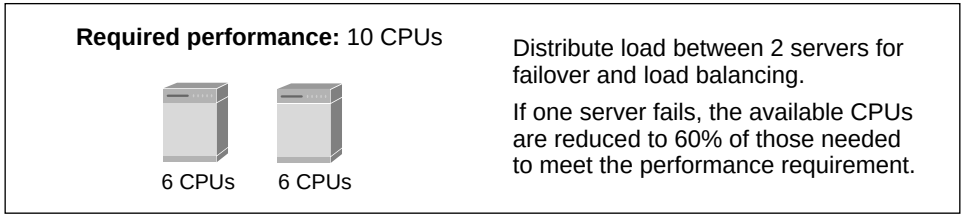


The computing power of each server in Figure 5-6 above is identical. One server alone handles the performance requirements. The other server provides 100% of performance when called into service as a backup.

Advantages of a replica server design is 100% performance during a failover situation. Disadvantages include increased hardware costs with no corresponding gain in overall performance.

The following figure illustrates a scenario that distributes the performance between two servers for load balancing and failover.

Figure 5-7 Distribution of Load Between Two Servers

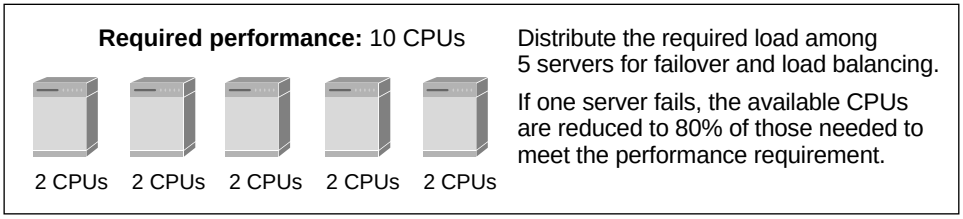


In Figure 5-7 above, if one server fails, all services are still available, although at a percentage of the full capacity. The remaining server provides 6 CPUs of computing power, which is 60% of the 10 CPU requirement.

An advantage of this design is the additional 2 CPU latent capacity when both servers are available. Also if one server fails, all of the services are available, but possibly at diminished performance.

The following figure illustrates the distribution between a number of servers for performance and load balancing.

Figure 5-8 Distribution of Load Between *n* Servers



Because there are five servers in the design illustrated in Figure 5-8, if one server fails the remaining servers provide a total 8 CPUs of computing power, which is 80% of the 10 CPU performance requirement. If you add an additional server with 2 CPUs capacity to the design, you effectively have an N+1 design. If one server fails, 100% of the performance requirement is met by the remaining servers.

This design includes the following advantages:

- Added performance if a single server fails
- Availability even when more than one server is down
- Servers can be rotated out of service for maintenance and upgrades
- Multiple low-end servers typically cost less than a single high-end server

However, administration and maintenance costs can increase significantly with additional servers. There are also hosting costs for servers in a data center. At some point you run into diminishing returns by adding additional servers.

Sun Cluster

For situations that require a high degree of availability (such as four or five nines), you might consider Sun Cluster as part of your availability design. A cluster system is the coupling of servers, storage, and other network resources. The servers in a cluster continually communicate with each other. If one of the servers goes

offline, the rest of the devices in the cluster isolate the server and fail-over any application or data from the failing node to another node. This failover process is done achieved relatively quickly with little interruption of service to the users of the system.

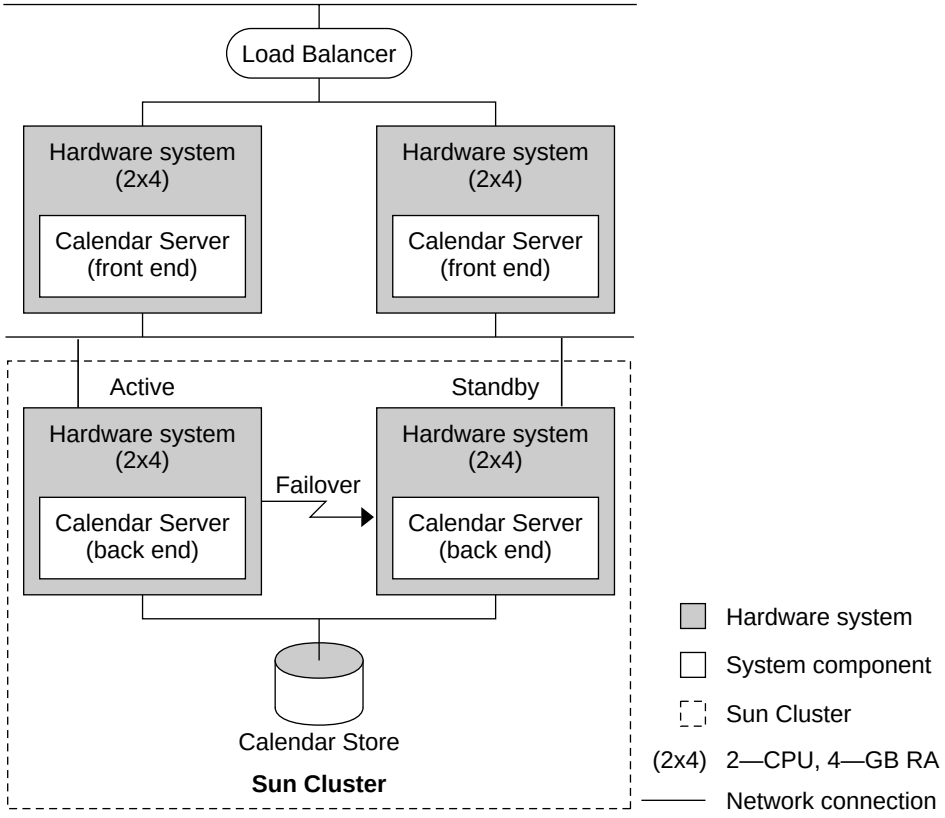
Sun Cluster requires additional dedicated hardware and specialized skills to configure, administer, and maintain.

Availability Design for Sample Deployment

The following figure shows an availability design for the calendar service portion of the example deployment, which was introduced in Chapter 4, "Designing the Logical Architecture." The figure depicts an availability solution for the Calendar Server piece of the logical architecture for the example deployment. An analysis of the complete availability solution for the example deployment is beyond the scope of this white paper.

The sizing exercise earlier in this chapter determined that Calendar Server requires 6 CPUs and 12 GB of memory, as depicted in Figure 5-3 on page 57. The following figure shows the front end of Calendar Server deployed on two servers for load balancing incoming and outgoing requests. The back end of Calendar Server is deployed on a separate server, and is replicated in a Sun Cluster 3.1 4/04 for failover. For failover purposes, the CPU and memory required for the Calendar Server back end is replicated in the Sun Cluster 3.1 4/04.

Figure 5-9 Availability Design for Calendar Server in Example Deployment



Serviceability Issues

When designing for availability, you must also consider the administration and maintenance costs of your solution. These costs are often overlooked in a design because they are not specifically tied to the purchase of hardware. Rather, they can be hidden, ongoing costs that reflect the complexity of your design.

For example, your design might include a large number of horizontally redundant servers that provide a high degree of availability. But if you do not factor in the costs to set up and configure the servers, continually upgrade the software, and monitor the health of the system the availability gain can be compromised.

When designing for serviceability, consider the following administration and maintenance costs:

- Setup and configuration

- Monitoring
- Upgrading server hardware
- Upgrading server software
- Automation of failover

Sizing for Scalability

Scalability describes the ability to add capacity to your system, usually by the addition of system resources, but without changes to the deployment architecture. This section discusses topics to consider when designing for scalability.

During requirements analysis, you typically make projections of expected growth to a system based on the business requirements and subsequent usage analysis. These projections of the number of users of a system, and the capacity of the system to meet their needs, are often estimates that can vary significantly from the actual numbers for the deployed system. Your design should be flexible enough to allow for variance in your projections.

Latent Capacity

Latent capacity is one aspect of scalability where you include additional performance and availability resources into your system so it can easily handle unusual peak loads. Latent capacity is one way to build safety into your design.

A careful analysis of use cases can help identify the scenarios that can create unusual peak loads (for example, a business to employee deployment that schedules mandatory webcasts). Use this analysis of unusual peak loads, plus a factor to cover unexpected growth, to design latent capacity that builds safety into your system.

You can also monitor how latent capacity is used in a deployed system to help determine when it is necessary to scale the system by adding resources.

Upgrading the Capacity of a System

Your system design should be able to handle projected capacity for the first 6 to 12 months of operation. Maintenance cycles can be used to add resources or increase capacity as needed. Ideally, you should be able to schedule upgrades to the system on a regular basis, but predicting needed increases in capacity is often difficult. Rely on careful monitoring of your resources as well as business projections to determine when to upgrade a system.

If you are performing an incremental deployment, where you defer deployment of parts of the system for business or technical reasons, you might schedule upgrading the capacity of the system to coincide with other upgrades that include new features of the system.

Optimizing Resources

Sizing a deployment is not just the estimation of resources to meet the system requirements. Sizing is also an exercise in both risk management and resource management. How a design handles risk management and resource management is often the key to meeting business goals.

Risk Management

Much of the information on which sizing is based, such as business requirements and usage analysis, is not empirical data but data based on estimates and projections. Before completing the sizing of a planned deployment, revisit the data and make sure your sizing design takes into account any reasonable deviations from the estimates or projections.

For example, if the projections from the business requirements underestimate the actual usage of the system, you run the risk of building a system that cannot cope with the amount of traffic it encounters. A design that under performs will surely be considered a failure.

On the other hand, if you build a system that is several orders more powerful than required, you divert resources that could be used elsewhere. The key is to include a margin of safety above the requirements, but to avoid extravagant use of resources.

Extravagant use of resources can also result in a failure of the design because t under utilized resources could have been applied to other areas critical to success. Additionally, extravagant solutions might be perceived as not fulfilling contracts in good faith.

Managing Resources

Managing resources is the process of analyzing all available sizing options and selecting the best fit solution that minimizes cost but still fulfills system requirements. This involves understanding the trade-offs for each design decision to make sure a benefit in one area is not offset by a cost in another.

For example, horizontal scaling for availability might increase overall availability, but at the cost of increased maintenance and service. Vertical scaling for performance might increase computing power inexpensively, but the additional power might be used inefficiently by some services.

Before completing your sizing strategy, examine your decisions to make sure you have balanced the use of resources with overall benefit to the design. This typically involves examining how system qualities in one area affect other system qualities. The following table lists some topics you might want to consider for management of resources.

Table 5-2 Resource Management Topics

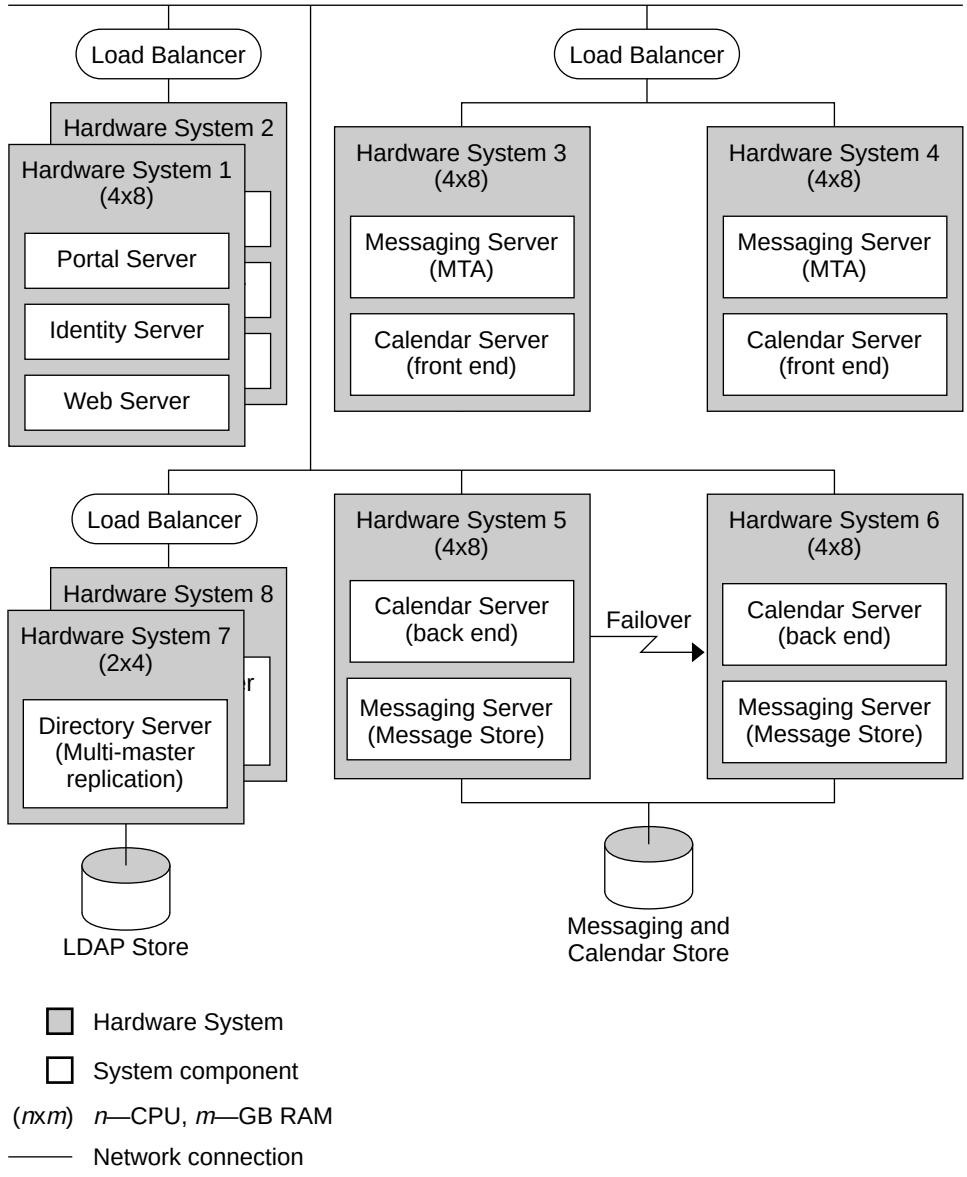
Topic	Description
Performance	For performance solutions that concentrate CPUs on individual servers, will the services be able to efficiently utilize the computing power. (For example, some services have a ceiling on the number of CPUs that can be efficiently used.)
Latent Capacity	Do you have a strategy to handle loads that exceed performance estimates?
	Are excessive loads handled with vertical scaling on servers, load balancing to other servers, or both?
	Is the latent capacity sufficient to handle unusual peak loads until the next milestone for scaling the deployment?
Security	Have you sufficiently accounted for the performance overhead required to handle secure transactions?
Availability	For horizontally redundant solutions, have you sufficiently estimated long term maintenance expenses?
	Have you taken into account scheduled downtime necessary to maintain the system?
	Have you balanced the costs between high-end servers and low-end servers.
Scalability	Have you estimated milestones for scaling the deployment?
	Do you have a strategy to provide enough latent capacity to handle projected increases in load up to the milestones for scaling the deployment?
Serviceability	Have you taken into account administration, monitoring, and maintenance costs into your availability design?
	Have you considered delegated administration solutions (allowing users themselves to perform some administration tasks) to reduce administration costs?

Example Deployment Architecture

The following figure represents a completed deployment architecture for the example deployment introduced earlier in this white paper. This figure provides an idea of how to present a deployment architecture.

CAUTION The deployment architecture in the following figure is for illustration purposes only. It does not represent a deployment that has been actually designed, built, or tested and should not be considered as deployment planning advice.

Figure 5-10 Example Deployment Architecture



Detailed Design Specification

After a deployment architecture is complete, there is a period for customer review, followed by, hopefully, project approval. In some cases the customer might redirect you to make changes to the deployment architecture before granting approval.

After project approval, you create a detailed design specification that is a starting point for implementation of the deployment. The design specification includes details on specific hardware resources and network devices, as well as a detailed LDAP directory specification.

Chapter 6

Implementing a Deployment Design

This chapter provides an overview of the steps necessary to implement a deployment design.

After the deployment architecture has been approved and a detailed design specification has been completed, you enter the implementation phase of deployment planning. During the implementation phase, you build out the deployment architecture. Depending on the nature of your deployment project, implementing a deployment design includes some or all of the following steps:

- Creating and deploying pilots or prototypes in a test environment
- Designing and running functional tests to measure compliance with system requirements
- Designing and running stress tests to measure performance under peak loads
- Creating a production deployment, which might be phased into production in stages

This chapter discusses the following sections:

Developing Pilots and Prototypes

Java Enterprise System deployments typically fall into two categories, those based primarily on services provided with Java Enterprise System and those that require a significant number of custom services that are integrated with Java Enterprise System services. You can think of the former type of deployment as an *80:20 deployment* (80% of the services are provided by Java Enterprise System) and similarly, the former as a *20:80 deployment*.

For 80:20 deployments, during the implementation phase, you typically develop a *pilot deployment* for testing purposes. Because 80:20 deployments use mature Java Enterprise System services that provide "out-of-the-box" functionality, pilot deployments move relatively quickly from development, testing, and modification steps, to production deployments.

20:80 deployments, on the other hand, introduce new, custom services that do not have the history of interoperability that comes with 80:20 deployments. For this reason, you create *prototype deployments*, which are proof-of-concept deployments that typically require a more rigorous development, testing, modification cycle before going to production.

NOTE Actual enterprise deployments can vary greatly in the amount of custom development of services they require. Whether you use pilot or prototype deployments for testing purposes depends on the complexity and nature of your deployment.

Testing Pilot and Prototype Deployments

The purpose of testing pilot and prototype deployments is to determine, as best as possible under test conditions, whether the deployment satisfies the system requirements and also meets the business goals.

Ideally, *functional tests* should model scenarios based on all identified use cases—a set of metrics should be developed to measure compliance. Functional testing can also involve a limited deployment to a select group of beta users to determine if business requirements are being satisfied.

Stress tests measure performance under peak loads. These tests typically use a series of simulated environments and load generators to measure throughput of data and performance. System requirements for the deployment are typically the basis for designing and passing stress tests.

NOTE Functional and stress tests are particularly important for large deployments where system requirements might not be well-defined, there is no previous implementation on which to base estimates, and the deployment requires a significant amount of new development.

Testing can indicate problems with the deployment design specification, and might involve several design, build, test iterations before you can roll out the deployment to a production environment. However, testing should never be the place where you discover problems with the deployment architecture. If you discover design problems at the testing stage for deployment architecture, then your analysis, planning, and design can be considered a failure.

Rolling Out a Production Deployment

Once the pilot or proof of concept deployment passes the test criteria, you are ready to roll out the deployment to a production environment. Typically, you roll out to a production environment in stages. A staged rollout is especially important for large deployments that affect a significant number of users.

The staged deployment can start with a small set of users and eventually expand the user base until the deployment is available to all users. A staged deployment can also start with a limited set of services, eventually phasing in the remaining services. Staging services in phases can help isolate, identify, and resolve any problems a service might encounter in a production environment.

Because testing never completely simulates a production environment, you should continue to monitor the deployed systems to identify any areas that require tuning, maintenance, or service.