CHAPTER 3

Exalogic Software Architecture
This chapter describes the types of software deployments that Exalogic is intended to host. Special focus is given to the Oracle Fusion Middleware platform, which has been specifically tested and optimized for the Exalogic environment. Much of Fusion Middleware’s deployment architecture is defined by the core runtime, WebLogic Server, so it is given particular emphasis. Lastly, this chapter describes how to monitor all of these middleware applications on Exalogic by using Oracle Enterprise Manager.

Target Applications
Generally speaking, Oracle supports any software on Exalogic that is certified to run on either Oracle Linux 5.5 or Solaris 11 (64-bit), the two operating systems that Oracle supports on the Intel-based Exalogic hardware. Consequently, you can use Exalogic to consolidate literally thousands of different software products, both Oracle and non-Oracle. It’s important to keep in mind, however, that Exalogic was designed and engineered primarily to host Oracle middleware applications. For these deployments, there can be significant performance gains on Exalogic, assuming you follow the recommended deployment described in this chapter.

TIP
The performance of non-Oracle applications will likely benefit from the dense computing and network power of Exalogic. However, it is always wise to conduct an initial benchmark to compare the application’s performance on Exalogic to the earlier hardware platform.

Java Enterprise Edition Review
For the past decade, the dominant force in the enterprise middleware space has been Java EE. This specification creates a contract between application developers and middleware software, freeing companies from being locked in to a specific vendor’s products. More importantly, Java EE allows developers to focus on implementing business requirements (order processing, billing,
claim processing, employee workflows, and so on) and spend less time worrying about infrastructure. Programmers no longer need to write code that manages threads and sockets, or code that handles load distribution and failover.

While there is no “typical” implementation of a custom application using Java EE, an example can help illustrate some of the burdens that Java EE attempts to remove from application developers. Suppose you need to build an auction system. First, you will likely need a web site that allows customers to find auctions, place bids, and configure notifications. In Java EE, the application code that generates a web site is referred to as a web application. Next, these auction tasks will need to be integrated into partners’ web sites and systems. Java EE supports the creation of web service applications, which use XML and other standards to facilitate interactions with other Java and non-Java applications. Finally, to support auction notifications, you will need a third application that collects and processes the notifications as a batch process, perhaps every 30 minutes. With Java EE, the Enterprise JavaBean (EJB) and Java Message Service (JMS) standards provide a simple solution for this common use case. If you put all of these pieces together, your final auction system resembles Figure 3-1.

To host Java EE applications on Exalogic or any server platform, you require a compliant application server, or “container.” Java EE containers supply applications with the necessary runtime infrastructure services, including support for various protocols, database access, messaging, security, and high availability. Oracle WebLogic Server is the recommended container for production Java EE applications. It performs and scales extremely well, particularly in Exalogic environments.

**FIGURE 3-1. A Java EE auction system**
Oracle Fusion Middleware

Although Java EE certainly facilitates the creation of custom enterprise applications, it tends to concentrate on the low-level infrastructure. However, there are invariably additional application requirements that tend to arise over and over that Java EE does not directly address. Consider the following requirements:

- How do I turn our web site into a corporate portal that lets my end users customize the site and add their own web content to it?
- How do I create and orchestrate some corporate workflow, such as hiring a new employee? Can I even allow end users to customize this workflow?
- How do I implement a centralized security manager that provides account management, single sign-on, and access control to the rest of our applications?

Oracle Fusion Middleware includes several products to help address these challenges. These products essentially leverage Java EE capabilities, such as web applications, web services, EJBs, and JMS, to provide higher-level functions. Most Fusion Middleware products are organized and packaged into suites, although the individual component products can often be licensed and installed individually. These product suites include the following:

- **SOA Suite**  Build and manage distributed, service-oriented applications that consist of reusable services, workflows, and business rules.
- **WebCenter Portal**  Build and manage collaborative web portals with sophisticated navigation, customization, and social computing features.
- **WebCenter Content**  Build applications that capture, manage, and surface documents, images, and other types of content.
- **Identity Management**  Use a centralized security infrastructure to manage users, roles, entitlements, and/or single sign-on tokens.
Oracle Fusion Applications

You might be wondering why you should bother with application development at all, when in many cases you can buy a complete, packaged software solution that already meets your business requirements:

- Customer relationship management
- Risk management and compliance
- Supply chain management
- Human resources
- Financial analysis
- Procurement

Fusion Applications represents Oracle’s latest evolution of its popular packaged applications, including E-Business Suite, PeopleSoft, JD Edwards, and Siebel. The primary difference between these older releases and Fusion Applications is the reliance on Fusion Middleware. Most Oracle Fusion Applications rely on multiple Oracle Fusion Middleware components, such as SOA Suite and Identity Management.

WebLogic Server

As mentioned, WebLogic Server is Oracle’s production-ready, Java EE application server. It can host any Java EE–compliant application. But more important, it is the only supported application server for most Oracle Fusion Middleware components as well as for most Oracle Fusion Applications. Therefore, understanding WebLogic’s architecture and capabilities is a critical part of understanding Exalogic’s software architecture.

Servers and Machines

An instance of WebLogic Server is often simply referred to as a “server.” From the host operating system’s perspective, a server is a running process listening to one or more ports. Each server process can host multiple Java EE applications along with other WebLogic resources that the applications require, such as database connections or message queues.
WebLogic Server is itself a Java application and therefore running it requires a Java Virtual Machine (JVM). For running a server on Oracle Linux, both the Oracle JRockit and Oracle HotSpot JVMs are supported; however, JRockit is recommended because of its better performance and administrative capabilities. The HotSpot JVM is recommended for Solaris on Exalogic. When you install WebLogic, you indicate which JVM it should use by default.

A physical piece of hardware, such as an Exalogic compute node, is not limited to running a single server process. Especially in the case of Exalogic, you can expect to run many servers on each compute node to utilize all the available CPU cores and memory. WebLogic documentation and tools use the term “machine” to designate these physical hardware boundaries. As part of configuring a WebLogic server, you can explicitly associate it with a machine. Like servers, WebLogic machines are simply given arbitrary configuration names, such as “MachineA” or “el01cn01.” Machine names need not map to a network host name. Put another way, a WebLogic machine is simply a logical collection of WebLogic servers running on the same physical device, as shown in Figure 3-2.

The definition of machines may seem unnecessary, and strictly speaking it is optional. However, several WebLogic capabilities rely on machine definitions, each of which will be addressed later in the “Node Manager” section of this chapter. Specifically, you must explicitly group servers using machines in order to use WebLogic’s Node Manager feature and to support some of WebLogic’s high availability features.

**FIGURE 3-2.** WebLogic servers and machines
TIP

On Exalogic, a WebLogic machine maps to a compute node. Define a separate WebLogic machine for each compute node that will run WebLogic servers.

Domains

WebLogic servers are organized and grouped together through the definition of machines and also through the definition of domains. A WebLogic domain is a set of servers that is configured, monitored, and managed as an administration unit. A server can belong to one domain only. Keep in mind that this is only an administrative boundary, not a functional one. A web application deployed in one domain can still, for example, communicate with an EJB or web service application that is deployed to another domain.

One (and only one) server in each domain is flagged as the domain’s administration server, as shown in Figure 3-3.

By default, the administration server in a domain is named AdminServer. An administration server supports all the same capabilities supported by any other server, along with some additional duties. To help distinguish the administration server from other servers in the domain, the remaining servers are often referred to as “managed” servers.

First, the administration server acts as the owner of the domain’s configuration files. These files describe all the functional aspects of each server, including IP addresses, port numbers, application deployments,

FIGURE 3-3. A WebLogic domain
database connections, message queues, and so on. As part of starting up, a managed server contacts the domain’s administration server and requests the configuration. Each managed server also caches a copy of these files, should the administration server be unavailable at a later time.

TIP
Deploy your business applications on managed servers. Do not deploy applications to the administration server.

Second, the administration server automatically hosts a web application called the WebLogic Administration Console. It can be accessed from a browser using the URL http://<adminserver>:<port>/console. The console provides a GUI to configure, manage, and monitor all servers in this domain from a single location. As modifications are made to the domain’s configuration, not only are the changes written to disk by the administration server, but they are also broadcast to all running managed servers. Figure 3-4 shows a typical interaction with the console—viewing the status of the servers in the domain.

FIGURE 3-4. The WebLogic Administration Console
How many domains should you create in your Exalogic data center? For larger organizations, different applications, especially those in different business units, are often managed by distinct IT groups. In this scenario, it makes sense to configure a separate domain for each major business unit. For example, you would separate the HRDomain from the OrderProcessingDomain. Furthermore, even within a single business unit, we recommend that each project phase be associated with its own domain. For example, consider an application that is first deployed to HR-QADomain and then to HR-ProdDomain. As we shall see in later chapters, tools such as Oracle Enterprise Manager let you manage and monitor multiple domains from a single interface.

On Exalogic’s shared file system, each domain is stored in a separate directory. The domain directory contains the domain’s configuration files along with server logs and other runtime files such as transaction and message logs. A domain directory also contains shell scripts that you can use to start and stop the domain’s administration server and managed servers. These scripts can be useful for initial domain testing and troubleshooting, but the preferred approach for starting and stopping servers is the WebLogic Node Manager, which will be discussed shortly.

**TIP**

Maintain each domain directory in a part of the file system separate from the location of the WebLogic Server product installation. Your product installation should effectively remain read-only and can be reused by many WebLogic domains.

**Clusters**

As a quick review, recall that you need to organize and group your WebLogic servers using two different constructs:

- **Domains**  
  Servers that are administered as a unit

- **Machines**  
  Servers that are hosted on the same hardware

A third way in which servers are grouped together is a *cluster*. A cluster defines a set of servers that work together to provide high availability for the applications and services deployed to the servers. Generally speaking,
all servers in a cluster are configured identically (ignoring minor exceptions such as the listen address and listen port). Cluster members periodically communicate with one another via a heartbeat mechanism. Servers then know which other servers in the cluster are available and unavailable at any moment in time. Figure 3-5 illustrates the WebLogic cluster architecture.

Another important characteristic of a cluster is that it appears to clients as a single server. High availability is then transparent both to clients and the applications being accessed. Finally, clusters provide a method of scalability for applications, because a single WebLogic server supports a finite capacity. You can quickly create additional capacity by adding more members to the cluster.

TIP

In general, you can deploy WebLogic applications to a single server or an entire cluster. Always target applications to a cluster, even if it initially only consists of a single managed server. This helps avoid some of the challenges involved in upgrading a single-server environment to a clustered one.

High availability within a cluster falls into two general categories: load distribution and failover. The mechanisms by which cluster members provide these features vary according to the type of application. For example, when an EJB client remotely accesses an EJB within a cluster, the client transparently
downloads and uses a WebLogic “stub” object that intelligently distributes requests across the cluster members, and the client also transparently fails over if a member becomes unreachable. The web and web service applications, on the other hand, rely on HTTP, where the option to use a WebLogic stub is not possible. Instead they need to include a proxy server between the client and the application hosted by WebLogic. This cluster-aware proxy performs similar behavior to that of the EJB stub. See Figure 3-6 to compare these two types of deployments in a cluster. Several hardware and software options are available to implement a WebLogic cluster proxy, including Oracle HTTP Server, Oracle iPlanet, and Oracle Traffic Director.

Failover can also take the form of data replication. Consider, for example, an EJB or web application that maintains a user’s shopping cart in server memory. If the server that hosts that shopping cart fails, the shopping cart is lost. Although the user can automatically be redirected to another available server, the user will have to start over again to re-create his shopping cart. For many types of applications, the loss of this in-memory session data is unacceptable. Cluster members address this issue by replicating sessions, so that at any given moment there is no single point of failure. WebLogic supports replicating session data across members within the cluster, or replicating externally to a highly available file system or database.

Getting Started with WebLogic

After installing the WebLogic Server product, you are ready to create your first domain. The primary tool used to create a new WebLogic domain, and also to define the initial servers, machines, and clusters to be contained in

**FIGURE 3-6. Cluster load balancing and failover**
the domain, is the Fusion Middleware Configuration Wizard. It interactively
guides you through the process of creating your domain’s configuration. The
output of the Configuration Wizard is a new domain folder on the shared
file system. You are then free to start the domain’s administration server and
make further modifications to the domain (such as adding a new server)
using online tools such as the administration console. As you get more
proficient with WebLogic domains, if you want to improve automation
of the domain provisioning process, you can employ tools such as the
WebLogic Script Tool (WLST) or Oracle Enterprise Manager.

TIP
Remember that all product binaries,
configurations, and applications should be
placed on the ZFS storage appliance. This
includes WebLogic installations and domain
configurations. A recommended approach for
organizing this shared storage is covered next
in Chapter 4.

To create a new domain, you must select at least one template. Templates
define the default topology, applications, and settings for the domain. A base
WebLogic Server installation includes a few simple templates, and each
separate Fusion Middleware product installation (WebCenter or Identity
Manager, for example) will include additional templates. Figure 3-7 lists some
of the FMW templates you may be asked to choose from while running the
Configuration Wizard. Refer to the Deployment Guide for a specific Fusion
Middleware product in the Oracle documentation to determine the template
or templates you should choose in the Configuration Wizard.

The general tasks that you perform with the Configuration Wizard are as
follows:

■ Select one or more templates.
■ Provide the destination on the file system.
■ Enter credentials for the default administrative account.
■ Confirm the JVM that should be used to run this domain.
Enter the listen address and port of the administration server.

Enter the name, address, and port of each managed server.

Define any clusters and their member servers.

Enter the name, node manager address, and node manager port of each machine, along with the target servers.

Configure any template resources such as JDBC data sources or JMS destinations.

Notice that each server is initially configured with a single IP address and port number. But you may recall that an Exalogic compute node participates in multiple networks and is therefore assigned several IP
addresses (BOND0, BOND1, eth0, and so on). So which should you choose? If you leave the Listen Address field blank, a server will actually attempt to bind to all available interfaces and addresses it can find on the host. Although this default behavior is convenient, it does not provide the most optimized configuration on Exalogic. Instead, it is recommend that you specify the BOND0 IP address (or host name) for each server when creating a new domain. In the next section, we will see how to optimize the domain further to take advantage of all of the available Exalogic networks.

Although not specific to Exalogic, another issue to consider that is related to server network configuration is port number assignments. When running multiple servers on the same compute node, each server must be configured with a unique combination of IP address and port number. Although one approach is to use the same address and different port numbers on each server, this is generally not the approach that is recommended. Instead, create virtual, or “floating,” IP addresses for the same network interface (BOND0). The use of floating IP addresses also enables you to take advantage of certain WebLogic features, such as node manager’s server migration capability. For example, a floating server address could be temporarily transferred from one compute node to another in response to a failure or during hardware maintenance.

**TIP**

*Use floating IP addresses for all server listen addresses.*

Here is an example of configuring a floating IP address on Linux for BOND0:

```bash
ifconfig bond0:1 192.168.100.100 netmask 255.255.255.0 up
arping -U -I bond0 192.168.100.100
```

**Optimizing for Exalogic**

The performance of a standard WebLogic domain configuration will certainly benefit from the processing power, memory, and network bandwidth provided on the Exalogic hardware. In addition, the JRockit and HotSpot JVMs both include Exalogic-specific performance optimizations. However, you should also consider some simple modifications to your domain to take full advantage of the unique characteristics of Exalogic.
Server Memory (Heap)

The heap is the amount of memory available to the JVM for storing Java objects. The JVM itself also requires some memory of its own, but for application servers like WebLogic, the size of the heap is usually the critical consideration. When any JVM process is started, you can specify the starting heap size as well as the maximum heap size. The following example configures a starting heap of 100MB and a maximum heap of 1GB:

```
java -Xms100m -Xmx1g MyJavaProgram
```

As the current heap size approaches the maximum, the JVM must dedicate more and more CPU time to garbage collection—the discovery and cleanup of Java objects that are no longer in use. Worse still, if no heap is available at all during the creation of a new object, the JVM sends an “out of memory” error to the code that attempted to create the object. If this scenario occurs during execution of your application code, it may simply cause this one request to fail and not impact other users. But an “out of memory” error during WebLogic’s internal processing can result in a server hanging or crashing.

By default, all WebLogic servers specify minimum and maximum heap sizes of 256MB and 512MB, respectively. However, many applications require significantly more RAM. Remember that each compute node runs a 64-bit OS and has 96GB of RAM. The optimal amount of heap for a given server is going to vary according to the types of applications it is hosting, the expected workload, and so on. As always, it is a good idea to determine the best heap size experimentally using JRockit Mission Control, Oracle Advanced Diagnostics for Java, or another software package designed for diagnostics and tuning. However, a good starting point is 4GB per server. Modify your domain’s start scripts to change the default heap settings.

TIP

*In almost all cases, you will see better performance results running a cluster of multiple servers with 4 to 8GB RAM each, rather than running a single huge server that is allocated almost all of the compute node’s available RAM.*
Network Channels

The servers within a domain support several different forms of communication along with various protocols. Consider these different scenarios:

- The server accepts an HTTP request for a web or web service application.
- The server accepts a T3 request for an EJB application.
- The server accepts a T3 request to publish or consume a JMS message.
- An application running on a server accesses a database by using JDBC and a database vendor-specific wire protocol.
- During startup, a managed server contacts the admin server by using T3 to obtain its configuration.
- While using the console to monitor the domain, the admin server communicates with managed servers using T3.
- Cluster members send heartbeats and data replications to each other.
- The admin server accepts a Simple Network Management Protocol (SNMP) request.

**NOTE**

T3 is a WebLogic-specific wire protocol that is used by default for JVM-to-JVM communication and applies to several of the preceding scenarios.

Initially, each of your servers has been configured with a single listen address and port. This is referred to as the server’s default channel. Without any additional configuration, all of these scenarios are handled by this default channel. WebLogic is capable of accepting and multiplexing all the supported protocols via the same listen port. However, recall that an Exalogic machine includes different networks: private (BOND0), client (BOND1), and management. Ideally, each server should be configured so
that it utilizes the Exalogic network that is best suited for every one of these scenarios. For example, you can use the client network for external application requests and use the private network for administrative and cluster communication. Figure 3-8 illustrates the recommended WebLogic network configuration on Exalogic.

WebLogic’s network channel feature allows you to isolate different types of communication and assign them to specific addresses and ports. If a network channel is not explicitly created for a protocol, the default channel will be used instead. Earlier we recommended that you use a floating IP address on the private network as each server’s default channel. Consequently, you need to create additional network channels only for scenarios that involve the Exalogic client or management networks. Network channels are created for each server individually. In the Administration Console, select a server and open the Protocols and then Channels tabs, as shown in Figure 3-9.

**TIP**

Create an HTTP network channel on a domain’s administration server that is associated with the Exalogic management network. This will let you remotely access the Administration Console from a browser.
All the different communication scenarios can be implemented using a combination of the default channel and custom channels, with one exception. To identify a separate channel to use for data replication within a cluster, you must perform the following:

1. Create a T3 protocol channel on each cluster member; use an address on the private Exalogic network.

2. Update the cluster and specify the name of the channel to use for replication.

For the specific case of cluster replication on the private network, WebLogic can also take advantage of the Sockets Direct Protocol (SDP) as a faster alternative to TCP. The use of SDP reduces the latency involved in these server-to-server communications, which is especially significant for large session objects (a shopping cart with ten items, for example).
Enable this optimization on each server’s replication channel individually, as shown next.

Finally, you can enable one more Exalogic-specific optimization for replication traffic. As it turns out, a single network channel is not able to utilize all the available bandwidth on the 40Gb InfiniBand network. Instead, Oracle recommends that you define multiple replication channels for each server, which the servers will then use in parallel to transmit data. Luckily, you are not required to create all of these additional channels manually on each server in the cluster. Instead, edit the cluster settings of each server (via the Configuration and Cluster tabs) and provide a range of ten port numbers, such as “7005-7015.” Upon server restart, WebLogic will automatically start ten replication channels, all using the same IP address but different port numbers. This field is shown next.

Server Optimizations
WebLogic provides several more performance optimizations when running on Exalogic. These facilitate a great increase in server throughput, responsiveness, and scalability. Included are improvements to WebLogic’s mechanisms for request handling, networking, memory management, and thread management. Although you can enable each of these optimizations individually and for each server, Oracle recommends that you simply enable all of them together, and for all servers in a domain. Within the Administration Console, select domain-wide
settings (click your domain name) and locate the Enable Exalogic Optimizations checkbox, as shown next:

When this attribute is turned on and the servers are restarted, one of the first changes that takes place is in regard to threading. By default, WebLogic creates a fairly modest number of server threads and then increases this number as the workload on the server increases. In other words, a server dynamically optimizes its thread count. However, a server running on Exalogic has a much greater than average amount of processing power behind it, so it creates a much larger initial thread pool during startup. Similarly, if more threads are needed later, the server increases the size of the thread pool at a much faster rate. The process of creating threads is a relatively expensive one, so it makes sense to optimize sooner rather than later when the server is already busy handling application requests. Essentially, this optimization increases the default size of the initial thread pool and also increases the number of threads that are incrementally created when there is a need for additional threads.

The second group of optimizations relates to network I/O. On an InfiniBand network, the maximum transmission unit (MTU) is 64KB versus about 1.5KB for Ethernet. To take advantage of this throughput, the server’s internal network I/O engine is optimized to use larger message buffers and to perform multiple reads and writes in parallel. Furthermore, in general the server optimizes its calls to the OS kernel based on the assumption that it in turn is using an InfiniBand driver. These WebLogic optimizations will be discussed in even greater detail in Chapter 7.

Node Manager

Configuring and optimizing the collection of servers in the domain is only a start. Now we must consider how best to manage the server lifecycle. In earlier sections, we mentioned that a domain is represented on the file system as a folder, and that its contents include shell scripts to start and stop servers in the domain. Therefore, you can certainly configure your host operating system so that these scripts are executed as the compute node
boots and shuts down. However, this approach is not recommended, because it does not account for all possible scenarios in a server’s lifecycle. For example, you may want to do the following:

- Remotely start or restart a server.
- Suspend or bring down a server for maintenance, without shutting down the entire operating system.
- Move or migrate a server from one host to another for maintenance purposes.
- Enable failed servers to be automatically restarted.
- Automatically kill and restart a failed or hung server.

Ideally, you should be able to perform all of these tasks from a single location. WebLogic’s node manager fulfills all of these needs.

**Node Manager Architecture**

Similar to a WebLogic server, a node manager is a Java process that listens for requests on a specified address and port. However, the node manager is limited to handling remote server lifecycle requests. Consequently, it has a very negligible CPU and memory footprint compared to a server. For each machine (compute node) that will run at least one server, you run just a single node manager process. You need only a single node manager per machine (compute node), regardless of to which domains each of the host servers belongs. A node manager responds to remote requests to start, stop, suspend, resume, kill, or migrate a server. These lifecycle requests can be issued by a domain’s administration server (via the Admin Console), Oracle Enterprise Manager, or by the WLST. The diagram in Figure 3-10 illustrates the node manager architecture.

Node manager can also be used to start administration servers remotely. However, since the administration server itself hosts the administration console, you are limited to using one of the other mentioned tools to issue the start command. It’s also important to keep in mind that while node manager will, by default, automatically restart failed servers, it will monitor only the local server processes that it started. If, instead, a server is started through some other means such as a shell script, this process will remain outside of the node manager’s control.
Node Manager Configuration

The node manager for each compute node must be configured individually. The overall setup process for each node manager is as follows:

1. Create a node manager home folder on the storage appliance.
2. Create a start script and register it with the OS init process.
3. Create and initialize a nodemanager.properties file.
4. Use WLST to enroll one or more domains with the node manager.

Regarding the storage of node manager files, a recommended convention is to create a separate home folder for each node manager and give it the name of the hosting compute node, such as “el1cn15” (rack #1, compute node #15). For convenience, a sample node manager start script is found in the WebLogic installation at /server/bin, which you can copy into each node manager home folder.

**TIP**

*Maintain node manager home folders within a separate file system from that used for the corresponding WebLogic Server installation. Your product installation should effectively remain read-only.*
Similarly, when you launch node manager the first time, if a properties file is not present, it will automatically generate one that lists all available properties and their default values. At a minimum, the properties file should configure a listen address and port, and also specify that the node manager start and stop servers by using the default domain scripts. With this file, you can also tune other parameters related to node manager logging, health monitoring, auto-restart, and security. By default node manager requires a Secure Sockets Layer (SSL) connection, but to avoid some additional setup you can disable this. Here is a simple nodemanager.properties example:

```
NodeManagerHome=/u01/SalesApps/nodemanager/el1cn15
ListenAddress=el1cn15
ListenPort=5056
SecureListener=false
StartScriptEnabled=true
StopScriptEnabled=true
```

If you plan to use the domain’s administration server (via the WebLogic Admin Console) to start server processes remotely, you will also need to configure the connections between the administration server and each node manager. In the Admin Console, select the machine on which the node manager is running and click the Node Manager tab. Supply the host name and port of the node manager (the same values used in nodemanager.properties), and also disable SSL if applicable.

Finally, for each domain that includes servers that will be run on the target compute node, you must register, or enroll, the node manager with that domain’s administration server. To accomplish this task, you can manually create and configure some additional node manager and domain files. Alternatively, you can run a WLST command from a terminal or script, which generates these files for you. The latter approach is generally preferred since it is less error prone, although it does require that your domain’s administration server be running:

```
cd /u01/app/FMW/Middleware/wlserver_103/common/bin
./wlst.sh
connect('user','password','10.100.10.100:7001')
nmEnroll('/u01/SalesApps/domains/SalesDomain',
   '/u01/SalesApps/nodemanager/el1cn15 ')
exit()
```
Remote Start
After node manager is configured and running on each compute node that participates in your domain, you can begin issuing commands to the node manager to start servers in that domain (and later stop, kill, or migrate servers). WebLogic gives you the ability to control individual servers, all servers in a target cluster, or all servers in the entire domain. If you prefer the command line, you can send remote requests to a node manager from the same WLST interface discussed earlier in the chapter. Alternatively, if the administration server is running, you may perform this same functionality from the Administration Console. In general, you can control server lifecycle through the Control tabs, which are found in various places throughout the interface. To view the screen in Figure 3-11, for example, first click the name of the domain.

Migration
At this point, you are able to bring your servers up and down from a single remote location. If any of these servers fail, the node manager running on the Exalogic compute node will automatically try to restart them, if desired.

<table>
<thead>
<tr>
<th>Settings for MyDomain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
</tr>
<tr>
<td>Servers</td>
</tr>
</tbody>
</table>

Use this page to change the state of the servers in this WebLogic Server domain. Control and managing servers in Standby mode requires the domain’s administration port.

Customize this table

<table>
<thead>
<tr>
<th>Servers (Filtered – More Columns Exist)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
</tr>
<tr>
<td>Server ID</td>
</tr>
<tr>
<td>AdminServer (admin)</td>
</tr>
<tr>
<td>WLSD1</td>
</tr>
<tr>
<td>WLSD2</td>
</tr>
</tbody>
</table>

FIGURE 3-11. Start all servers in a domain using the console.
But now consider a situation in which the entire compute node has failed or is being taken offline for maintenance. If other compute nodes have available capacity, why not restart or migrate the servers from the unavailable compute node? Even more important than the performance considerations of a lost compute node are scenarios in which the application can no longer function. For example, consider a server with pending insurance claim messages on a failed compute node.

Since all of our server resources utilize a shared storage appliance, any server can potentially run on any of the compute nodes. An advantage of using the node manager architecture is that it enables you to support migration in these cases, including both manual and automatic migration. Keep in mind that automatic server migration in WebLogic is available only to clustered servers. Specifically, cluster members monitor each other’s availability, and, in response to a machine failure, an active member (called the cluster master) instructs the node manager on one of the remaining active compute nodes to restart the failed managed servers. The administration server does not directly participate in the migration process, although you can initiate a manual migration from the administration server (via the console or WLST). The WebLogic server migration architecture is illustrated in Figure 3-12.

To support this server migration architecture on Exalogic, you must perform the following additional configuration tasks:

1. Update nodemanager.properties to support address migration.
2. Enable automatic migration on an entire cluster or individual servers.
3. Provide a list of machines that can participate in migration.

FIGURE 3-12. Migrate a server across machines.
As part of executing a migration, the node manager is responsible for ensuring that the IP addresses required by the target server are available on the host machine. In fact, migration is one of the primary motivations for using floating IP addresses in the first place. There is no guarantee that a server will always run on the same compute node. Although node manager can dynamically add and remove the required IP addresses for a server, it must also know the associated interface and subnet mask for these addresses. Provide these parameters using nodemanager.properties. For a given range of IP addresses, indicate the interface name and subnet mask, as shown in this example:

```
bond0=192.168.100.1-192.168.100.50,NetMask=255.255.255.0
bond1=10.0.0.1-10.0.0.50,NetMask=255.255.255.0
UseMACBroadcast=true
```

**Coherence**

Consider a typical application that retrieves persistent data, such as an order or a report, from a relational database or similar source. This type of interaction is often slow, so to increase performance, the application caches the retrieved data in memory to avoid unnecessary interactions with the database. In the case of WebLogic, applications can use server memory as a cache, but this approach has its limitations. Suppose the amount of data to be cached is significant (for example, gigabytes). Adding more application servers to your cluster may be a solution, but these servers are designed to perform complex application processing, not to be simply a memory cache.

Instead, Oracle offers a dedicated caching solution called Coherence. Coherence is similar to WebLogic in that they are both Java servers and both support a clustering solution to provide scalability. However, Coherence is not designed to host Java EE applications and all of the services required by these applications. It is instead optimized solely for the purpose of distributed, in-memory caching of data objects. A single product installer is available from Oracle that contains both WebLogic and Coherence, and a license for WebLogic Suite includes a license for using Oracle Coherence.

**Coherence Clusters**

Recall that with WebLogic, servers are explicitly defined as members of a cluster. And, unless overridden, cluster members communicate with each other through each of the server’s default listen addresses and ports.
Coherence clusters, on the other hand, are defined more loosely and have a slightly more flexible topology. As Coherence servers start, they essentially search the network for other members of the same Coherence cluster. If a port is in use, the server will automatically try a different one. Furthermore, as data is sent to a Coherence server for caching, this data is automatically distributed, or partitioned, across the current cluster members to provide scalability and high availability. If a cluster member later becomes unavailable, the data is then repartitioned, similar to WebLogic session replication. Figure 3-13 depicts the basic Coherence architecture.

So who uses Coherence to cache data? In the context of a Fusion Middleware deployment, most Coherence “clients” are Java EE applications, but other custom Java or C programs can also utilize a Coherence cluster. However, unlike traditional client-server applications, Java programs that connect to a Coherence cluster are actually considered members of the cluster as well. In fact, they utilize the same protocol to discover Coherence servers on the network as the Coherence servers do themselves and automatically perform load balancing and failover. Additionally, these clients can be storage-enabled, meaning that they also participate in the maintaining and partitioning of cached data.

Within an Exalogic deployment, all Coherence cluster members (including clients such as WebLogic Server) are often co-located in the same data center, in which case it makes sense for these members to utilize the private InfiniBand (IB) network (BOND0) in their configuration settings. If, instead, certain cluster members are not on the IB fabric, then cluster members may instead bind to the client network. Another important consideration is Coherence’s default use of

---

**FIGURE 3-13.** Cache partitioning within a Coherence cluster
UDP multicast (WebLogic clusters do not use multicast by default). Your data center’s existing 10Gb Ethernet service network may block multicast traffic, in which case you must configure Coherence to use TCP instead.

**Coherence*Web**

Originally we introduced the Coherence solution by contrasting this to an all-WebLogic architecture. Many Java EE applications are stateful in nature and store information in memory on behalf of the client. Consider a company store application that maintains a user’s shopping cart. This server data is referred to as the client’s *session*.

If your goal with Coherence is to implement a centralized, consolidated caching solution on Exalogic, it may also make sense to offload application sessions from WebLogic to Coherence. With this model, you can separately scale and tune your processing and caching tiers. A convenient way to accomplish this is through a WebLogic plug-in for Coherence called Coherence*Web. When deployed to a WebLogic server, the server essentially acts as a storage-disabled member of a specified Coherence cluster and is now able to leverage the Coherence cluster to hold session objects. Coherence*Web does not require any changes to your existing Java EE applications.

**Coherence and WebLogic Domains**

Each Coherence server’s configuration is maintained in a set of files that are referenced when the server is started. Traditionally, these Coherence files were created and edited manually, and configured and managed independently from your WebLogic domain infrastructure. However, beginning with WebLogic 10.3.4, a domain can be used to administer both your Coherence and WebLogic configurations. For example, you may use the Administration Console or WLST to define new Coherence servers and clusters. Figure 3-14 shows how to view Coherence servers in a domain by using the console.

In addition, the node manager now supports managing Coherence processes. This includes the abilities to remotely start or stop Coherence servers, and to also perform health monitoring and automatic restart.
Now let’s return our attention to WebLogic Server clusters. Remember that the general purpose of a cluster is to provide transparent load distribution and failover. The clients of web applications are typically browsers and can’t download and run stub code as a Java application can. Instead, web browsers must interact with the web application via some intelligent proxy layer that is capable of making load balancing and failover decisions. This proxy layer is referred to as the web tier, to differentiate it from the application tier (WebLogic and other Fusion Middleware) and the data tier. The diagram in Figure 3-15 illustrates the concept of a cluster proxy.

![Diagram of cluster proxy](image-url)
Generally speaking, WebLogic has only one requirement of the web tier, and technically it’s not a strict requirement. Once a user establishes a session with a web application, subsequent requests from that same user should be directed to the same server whenever possible. In other words, a user should be “pinned” to the server that is hosting his or her data. This expected behavior is also referred to as server affinity or session stickiness. At this point you might ask the question, “Doesn’t this introduce a single point of failure?” What happens to a user’s shopping cart if the server she was using fails? WebLogic provides a feature called session persistence to handle this scenario. Basically, if a server fails and the proxy directs the user to another server in the cluster, a backup copy of the shopping cart is retrieved and used without interruption to the user. However, optimum performance is achieved when the web tier is able to pin users to their assigned servers during normal processing.

Web tier proxies come in a range of flavors. Many hardware and software solutions exist, and several of each are certified for use with WebLogic clusters. One solution is to use Oracle’s Web Tier set of software utilities for this purpose. This brand includes the following:

- Oracle Traffic Director (OTD)
- Oracle HTTP Server (OHS), a derivative of the Apache web server
- Oracle (formerly Sun) iPlanet Web Server

Traffic Director

Oracle Traffic Director is the only option from the preceding list that is not a traditional web server. It cannot host web content or run scripts, for example. It is simply a high-performance HTTP load distributor that has been optimized for Exalogic’s InfiniBand network fabric. Therefore, unlike a hardware load balancer, it is specifically engineered to run on Exalogic itself and to distribute traffic locally within the same data center. Traffic Director can perform simple round-robin load balancing, but it also supports more complex routing rules and priorities. It can also cache frequently accessed content to increase performance, similar to other popular load balancing products.

A Traffic Director server, or instance, is configured with one or more server pools, each of which is a collection of HTTP URLs to direct traffic to. These can be any HTTP-based servers, but in the case of Exalogic they are
most likely to be WebLogic servers. Simply supply an initial list of servers in a WebLogic cluster to which Traffic Director will attempt to connect. Once Traffic Director is successfully connected to any one server, that server provides Traffic Director with the latest list of all running cluster members. This list is then dynamically updated with each response from the cluster.

The deployment architecture of OTD has many similarities to WebLogic Server. A group of instances is managed by a central administration server, which hosts a web-based administration console. Also a type of node manager is used to enable remote management capabilities for all instances on the same machine or node. OTD does not include a cluster configuration entity, however. Instead, two instances can be paired in an active/passive configuration for failover purposes. If the active instance fails, the backup instance will automatically bind to the same IP addresses that were previously in use by the failed instance.

Proxy Plug-ins
WebLogic Server supports several other cluster proxy solutions, including both Oracle and non-Oracle ones. Whichever product you choose, you will need to configure the WebLogic proxy plug-in. This plug-in understands how to communicate with WebLogic clusters and is responsible for orchestrating load balancing, sticky sessions, and failover.

For convenience, a copy of the plug-in comes packaged with Oracle HTTP Server. However, for other supported proxy server products you must explicitly install the plug-in. The exact installation and configuration steps vary slightly from proxy to proxy, but the overall process is the same. Following is an excerpt from mod_ohs_wl.conf, the file that’s used in Oracle HTTP Server to configure the plug-in:

```
<IfModule weblogic_module>
  WebLogicCluster 192.168.56.151:7001,192.168.56.152:7001
</IfModule>

<Location /companyportal>
  SetHandler weblogic-handler
</Location>
```

Simply supply an initial list of WebLogic servers that the plug-in will attempt to connect to. Once the plug-in is successfully connected to a
server, the server will respond with the latest list of all running members in
the same cluster, similar to the behavior in Traffic Director. The plug-in then
distributes additional application requests across the cluster in a round-robin
fashion.

TIP

It is recommended that you run Oracle HTTP
Server outside of the Exalogic machine on
separate hardware, particularly to meet existing
security and DMZ requirements. Proxy servers
can connect to the application tier using the
external EoIB network.

Enterprise Manager

As you may have noticed by now, a typical Exalogic environment contains
a lot of moving parts. In addition to the server, storage, and networking
hardware, there are all the operating system instances to consider. On top
of these operating systems, you are likely to run dozens or maybe even
hundreds of different application processes, including WebLogic servers,
other Fusion Middleware products, and Oracle applications such as Siebel,
E-Business Suite, and PeopleSoft. It can be a daunting task to administer,
maintain, and monitor the health of all of these pieces.

Oracle’s Enterprise Manager products provide administrators with a
large-scale management infrastructure along with a comprehensive and
unified interface for managing such a heterogeneous environment. Both of
the products introduced in this chapter, Grid Control and Ops Center, use
an agent-based infrastructure that allows you to perform most day-to-day
management tasks from a remote location. These Enterprise Manager agents
are deployed to each remote system that needs to be managed, and they
accept commands from the central Enterprise Manager management server.
Data is periodically gathered by the agents, published to the central
Enterprise Manager repository, and then used to trigger alerts and generate
reports. As you will see shortly, Grid Control focuses on software
monitoring, diagnostics, and management, while Ops Center focuses on the
same for hardware.
TIP

Although Enterprise Manager agents will be deployed to and run within Exalogic, Oracle recommends that you host the management servers and repositories outside of Exalogic. Management agents and servers can then communicate using the Exalogic management network. This approach lets you manage both the middle and data tiers together.

Generally speaking, Enterprise Manager is licensed separately from both the Exalogic hardware and software. In addition, the individual features of each Enterprise Manager product are often licensed separately as plug-ins called management packs.

Grid Control

With Oracle Enterprise Manager Grid Control you can perform any combination of the following tasks on your compute nodes, all from a single web-based console:

- Monitor operating system metrics and trigger alerts.
- Upload and execute scripts.
- Administer NFS mounts.
- Start, stop, and monitor all WebLogic Servers and Oracle applications.
- Monitor Java EE application availability and performance.
- Provision compute nodes to support specific applications.
- Validate compliance with known good configurations.
- Perform low-level JVM and WebLogic diagnostics.

This list represents just a subset of the functionality available through Grid Control. If you are also using Exalogic to host other Fusion Middleware products such as SOA Suite or Web Center, Grid Control provides additional administrative features that are specific to these products.
NOTE

Although this chapter refers to Grid Control, the latest version of the product has been renamed Cloud Control. The features described here are available in both versions.

Recall that each WebLogic domain has its own administration server that maintains its servers’ configurations and hosts its own copy of the web console. However, cases do arise, particularly in an Exalogic environment, in which it would be convenient to perform some task across multiple domains. A common scenario is checking the availability of all of your servers. With Grid Control, you can easily manage and monitor multiple domains from a single user interface.

Grid Control supports its own organizational units as well. Any software that can be managed/monitored is called a target. Targets also have implicit relationships to other targets. For example, a WebLogic domain target can span multiple host (OS) targets and also deploy application targets. You can also group related targets together and even create a custom dashboard for your system with metrics and reports. In fact, once agents have been installed and started on your compute nodes, Grid Control can automatically discover targets that it is able to manage and then automatically register them with the management server. On Exalogic, the output of this operation is a system group that consists of all compute node hosts and WebLogic domains, and that you can customize as needed.

In addition to providing a unified view of your Exalogic deployment, Grid Control has some additional Exalogic-specific features:

- Provisioning firmware, operating system, middleware, and Oracle applications on compute nodes
- Cloning an Exalogic deployment to allow easy scaling and to help transition from testing to production
- Performing functional, nonfunctional, and load tests on Exalogic deployments
- Managing patches and tracking changes

Finally, keep in mind that Grid Control is not limited to middleware administration. If your management network consists of relational databases,
either Oracle or non-Oracle, you can manage and monitor them with Grid Control as well. As you might expect, you can perform tasks such as provisioning a new database node, starting database processes, monitoring metrics, and executing SQL scripts.

**Grid Control Deployment**

Oracle Enterprise Manager Grid Control deployment requires an Oracle Enterprise Management Agent and the Oracle Management Service (OMS) installations. OMS is the central management control of Enterprise Manager and is normally deployed outside of Oracle Exalogic environment. The Management Agent is the software that is used to monitor each of the compute nodes. The deployment of a Management Agent that shares its software binaries, in this context, is called the Master Agent. In Exalogic, because access to the shared storage appliance is available to all compute nodes, only a single, full Oracle Enterprise Management Agent installation is required. Then you must run the Shared Agent Deployment Wizard tool on each compute node to configure the compute node to boot its agent from this shared installation. This remote agent deployment requires that some files be stored locally on the solid state disks of the compute nodes, but the majority of the agent binaries remain on shared storage.

In summary, setting up Oracle Enterprise Manager Grid Control in the Oracle Exalogic environment involves the following steps:

1. Install Grid Control Enterprise Manager System on an external system to Exalogic.

2. Install Oracle Management Agent on the Sun ZFS Storage 7320 appliance.

3. Install Oracle Management NFS Agents on each Exalogic compute node.

**Ops Center**

With Grid Control, you can manage the applications on your compute nodes. But how do you monitor and manage the underlying compute node hardware or your Exalogic storage and networks? For example, how can you assess the health of the InfiniBand network or update the firmware on the Exalogic switches? Or how can you coordinate an OS patch across all compute nodes?
These types of lower-level data center administrative tasks are addressed by Oracle Enterprise Manager Ops Center.

Similar to Grid Control, Ops Center can quickly discover your existing hardware and OS assets on your management network, given a range of IP addresses. Include the Integrated Lights Out Manager (ILOM) and management (eth0) addresses for the compute nodes as well as the management addresses for the storage nodes and switches. You will then be able to view the hardware specifications, sensor readings, and metrics for all of these components from a single, unified interface. Ops Center also has convenient shortcuts to launch the browser or command line interfaces for ILOM, a switch, or the storage appliance.

Arguably the most critical feature of Ops Center is its monitoring rules framework. Define conditions based on hardware or OS metrics that, when met, trigger alerts. A hardware fault is a common example. Furthermore, if you have an existing support contract with Oracle for these assets, these alerts can also trigger the creation of new support requests.

Summary

Oracle Exalogic is designed to provide high performance and throughput for Java- and JVM-based applications, particularly those built on Java EE and Oracle Fusion Middleware. This Exalogic software architecture consists of several layers, from the OS, to the JVM, to the Java EE container, WebLogic Server. Fusion Middleware then runs as applications and services on WebLogic. These WebLogic processes are grouped into administrative domains, which are then configured and optimized for Exalogic using the WebLogic Administration Console. The latest releases of WebLogic have been specifically engineered to run on Exalogic. Various network, I/O, and threading optimizations are available and require various levels of configuration.

A separate JVM process called the WebLogic node manager enables remote process management and also facilitates server migration. Migration is one of several clustering features in WebLogic, which allow your applications to scale and be highly available. A third type of JVM process is Coherence, which provides a distributed caching solution for very data-driven applications. Finally, you can manage and monitor all of these different software layers and processes from a single interface with Oracle Enterprise Manager.