

SSL, LOAD BALANCERS, REWRITE, REDIRECT AND MORE ADVANCED CONFIGURATION

*Dan Norris, dnorris(at)piocon.com, Piocon
Matt Topper, matt.topper(at)oracle.com, Oracle*

INTRODUCTION AND TERMINOLOGY

Oracle Fusion Middleware is a complex product with many individual components that must all work together. Basic installation is relatively straightforward, but few enterprise environments use the basic installation. After many enterprise deployments, some common themes surface in the configuration types that most sites use. This paper will offer instructions on configuring some of the more complex deployments available including SSL, load balancers, persistency, mod_rewrite, and logging.

To support the discussion that follows, these definitions may be helpful:

Term	Definition
OAS	Oracle Application Server
DAS	Delegated Administration Service
SSO	Single Sign-On
OID	Oracle Internet Directory (LDAP v3 server)
Midtier or Middle Tier	The tier in OAS where applications are usually deployed. This is also the tier where Portal, Discoverer, BIEE, Forms, Reports, and OC4J-based applications are deployed.
Infra or Infrastructure Tier	The tier in OAS where SSO, DAS and OID are housed and served. A single infrastructure tier can be utilized by multiple middle tiers. Infrastructure tiers can also be configured for redundancy.
SSL	Secure Sockets Layer

In the sections that follow, you will learn about:

- Enterprise deployment types
- Load balancing configuration
- SSL configuration
- Rewriting URLs

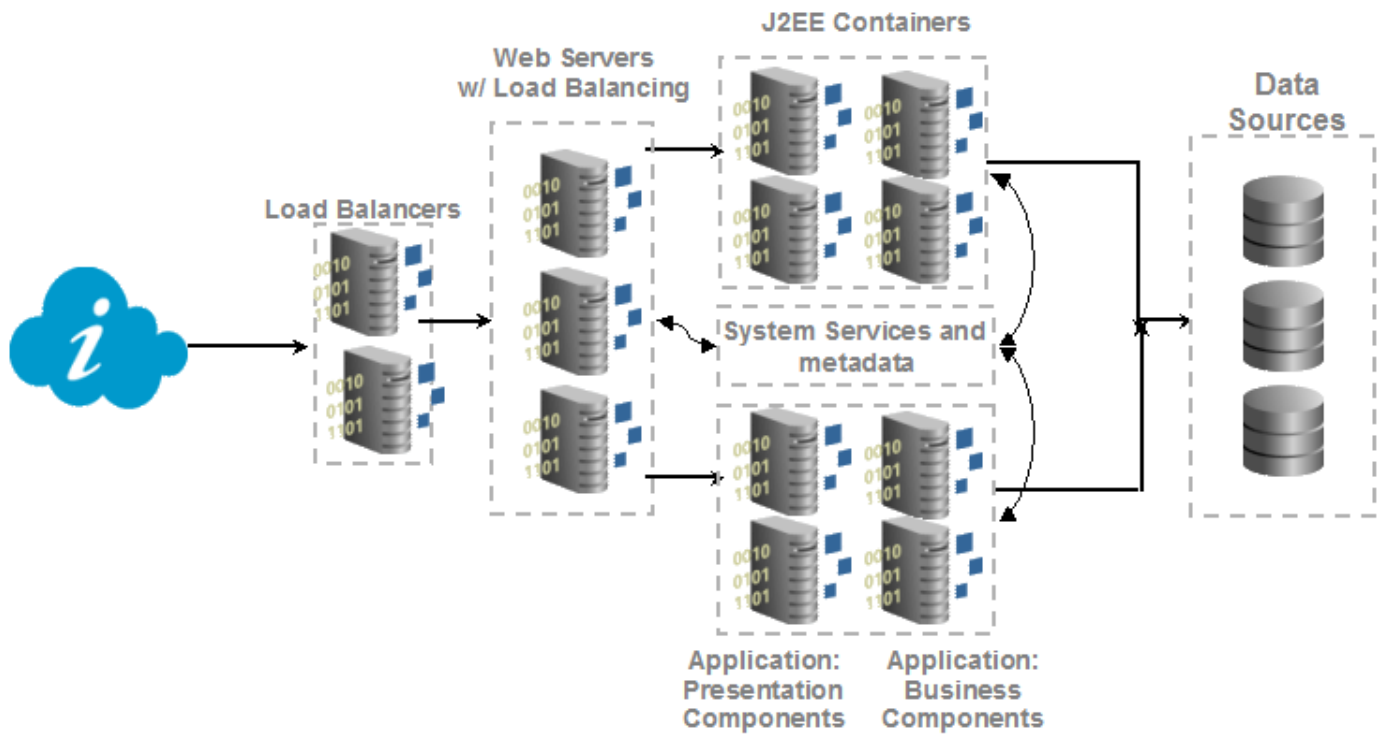
ENTERPRISE DEPLOYMENT TYPES

Part of the complexity of the Oracle Application Server product is understanding the architectural components and how they depend on one another and communicate with one another. Additionally, there are many different version numbers for the various components and some version numbers included only certain parts of the environment, making it more difficult to create a supported, working architecture plan.

Oracle has created sample deployment types in the standard documentation. In addition to the standard deployment architectures, deployment variants including cold failover clusters, OID replication, third-party integration with SSO, using load balancers, and more. In this paper, we'll be discussing portions of some of these variants.

APP SERVER PRODUCTS AND VERSIONS

Another area of frequent confusion for application server practitioners is the myriad versions and editions available with Oracle Application Server. You'll find a brief summary of the most recent versions of Oracle Application Server available at the time of this writing (March, 2008).



ORACLE APPLICATION SERVER (10.1.2)

Oracle Application Server 10.1.2 is the most recent complete release of all Oracle Application Server components including the infrastructure tier and middle tier. Since this release, many of the middle tier components have been updated via subsequent releases that have distinct version numbers, but only contain middle tier components. These subsequent releases may utilize the 10.1.2 infrastructure tier for single sign-on and centralized configuration and cluster management.

ORACLE SOA SUITE 10.1.3

Oracle SOA Suite 10.1.3 is a middle-tier-only installation that contains the tools necessary to deploy applications that are based on Oracle's Service Oriented Architecture. These tools include:

- Oracle ADF
- Oracle BPEL Process Manager
- Oracle Business Rules
- Oracle Enterprise Service Bus
- Oracle Web Services Manager
- Oracle TopLink
- XML Query Service

Additionally, some of these components are available (both technically and for licensing purposes) as separate standalone components.

ORACLE WEBCENTER SUITE 10G (10.1.3)

Oracle WebCenter 10.1.3 is an application framework similar in some ways to Oracle Portal in that it allows developers to easily create dynamic, high-productivity work environments that harness SOA's strength. The components in WebCenter are the following:

- Oracle ADF
- Oracle TopLink
- Oracle XML Query Services
- Oracle WebCenter Portal Services

- Oracle WebCenter Content Integration Services
- Oracle Content Database

Some of these components are available in other suites and packages as well.

ORACLE PORTAL (10.1.4)

Oracle Portal is an integrated portal product that enables customers to build and deploy portals using a standards-compatible platform. It has ability to plug in many different data sources and different source types ranging from data supplied by SQL queries to obtaining HTML snippets from other websites and presenting it in portal pages. Oracle Portal also has some capability to manage content and files may be uploaded and indexed, though Oracle Content Management (a relatively recent acquisition as compared to Portal's heritage) is generally better suited to the task of content management on a large scale.

The Oracle Portal 10.1.4 release is somewhat unique since it is only offered as an upgrade. So, customers desiring to use 10.1.4 have to install 10.1.2 and then upgrade to version 10.1.4. However, given the significant new features available first in version 10.1.4, it will certainly be an attractive release for new customers and existing customers will be driven to upgrade to it as well. While Oracle Portal 10.1.4 is an upgrade, the 10.1.4 upgrade does not upgrade the entire application server to the 10.1.4 version—just the Oracle Portal product. The rest of the application server remains at version 10.1.2.

WEBCACHE DEPLOYMENTS

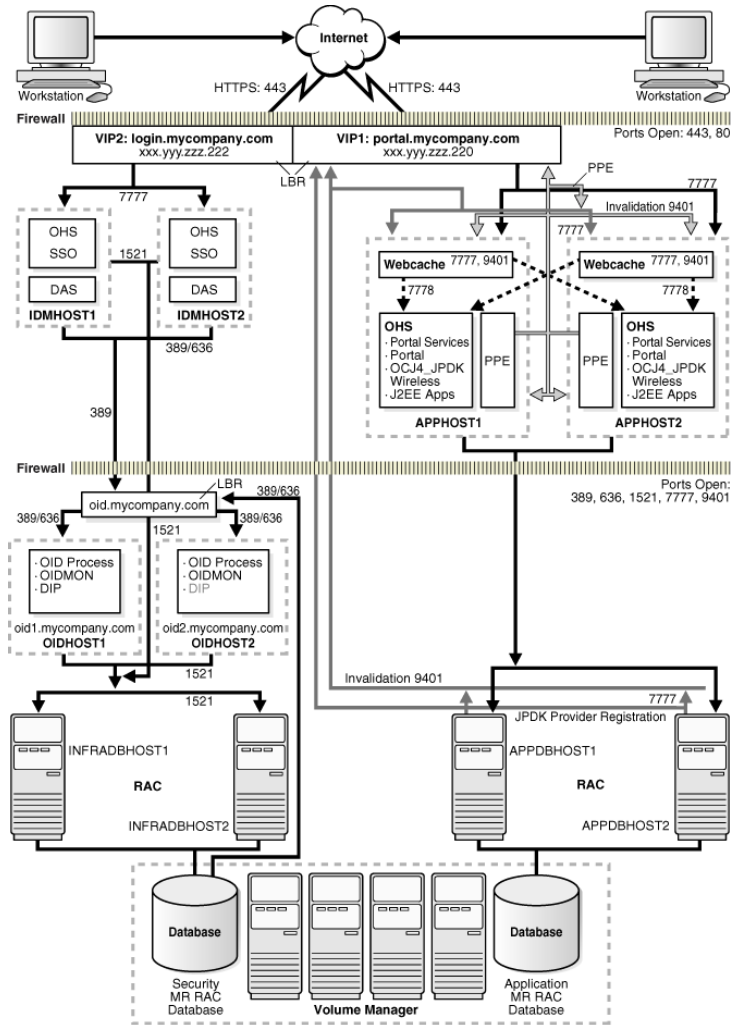
The Oracle WebCache is an application server component that provides in-memory caching for partial or entire sites based on a set of caching rules. By caching in memory, the end user's response time is significantly lower. WebCache is commonly deployed and integrated with Oracle Portal and sometimes other custom applications as well.

WebCache is a component that must be considered when creating advanced configurations, especially those involving clustering or load balancing. The WebCache is the first component access by a user HTTP request and is also involved in some recursive requests depending on the configuration. When clustering and load balancing, WebCache invalidations must be propagated to all WebCaches to ensure that no invalid content is presented from the caches. Generally speaking, as long as this “rule” is followed, WebCache deployments aren't too difficult to manage.

When configuring name-based VirtualHosts, the webcache configuration will need to be modified to properly route requests for each of the host names you're hosting unless wildcards can be employed effectively. You'll find good references for proper virtual host configuration in the OAS documentation.

HTTP SERVER DEPLOYMENTS

The Oracle HTTP Server is very closely derived from Apache HTTPD and configuration parameters are largely the same. The HTTP server configuration is one of the most common areas where deployment issues arise in enterprise deployments due to the multiple locations for virtual hostnames, virtual IP addresses, name-based virtual hosting and SSL configurations. Oracle's products support use of name-based virtual hosting, but it is sometimes challenging to determine the proper configuration for



each component since many of them depend on one another and improper configuration with one component may cause the HTTP server configuration to behave differently or become ineffective.

You'll find the HTTP server configuration in `$ORACLE_HOME/Apache/Apache/conf` and for those familiar with Apache HTTPD, the files will be very familiar: `httpd.conf` and its included conf files like `ssl.conf`, `oracle_apache.conf` and many others. Read carefully through the included files as there are several levels of includes and you'll find parts of the Apache configuration tucked away in many directories other than `$ORACLE_HOME/Apache/Apache/conf`.

OC4J DEPLOYMENTS

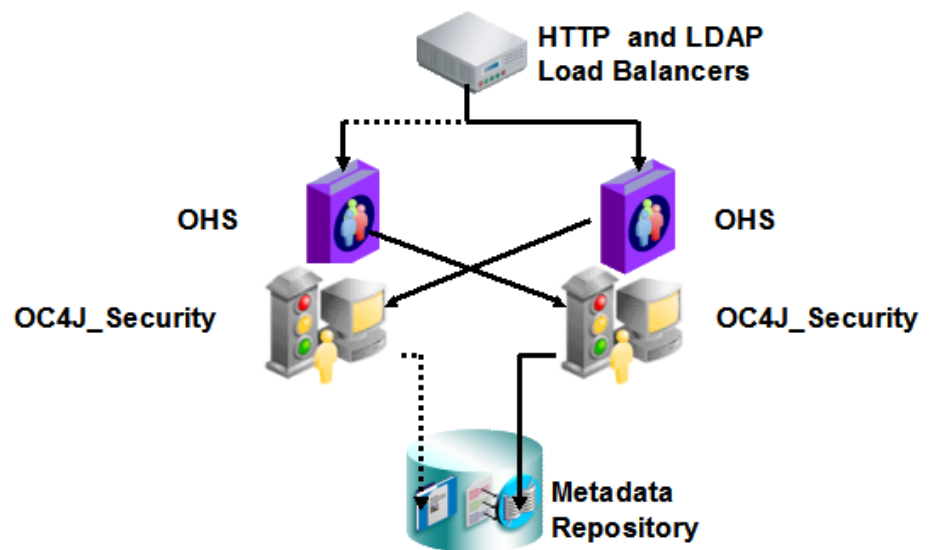
Oracle Containers for Java (OC4J) is Oracle's J2EE container that forms the core of all Java functionality in the application server. The standards followed by OC4J require it to provide many state variables and infrastructure services to Java applications. These variables are set by server configuration parameters and many are derived from various properties of the ways in which it is called.

The biggest challenges with the OC4J configuration in enterprise deployments are ensuring that the OC4J is properly deriving the parameters for things like the hostname, protocol, and port number. These are the same parameters used extensively in virtual hosting configurations. When using OC4J in configurations that also use `mod_rewrite`'s functionality to rewrite requests, there are sometimes challenges due to some bugs in the OC4J/J2EE combinations in use. Metalink BUG 5726819 documents one issue we've encountered when using `mod_rewrite` with OC4J applications. See the end of this paper for a list of other helpful Metalink notes.

LOAD BALANCING

Load balancing is a configuration that typically uses specialized network appliance devices to handle incoming requests and pass them off to multiple back end servers using a variety of different algorithms. Most load balancers will offer methods such as round robin, ratio (priority), fastest, least connections, and least sessions. In the sections that follow, we'll discuss the important configuration aspects of each setup.

Starting with 10.1.2, the documentation includes some sample configurations for common load balancer appliances in Appendix A of the Enterprise Deployment Guide (see URL in reference list later in this document).



MULTIPLE MIDDLE TIERS

When load balancing multiple middle tiers, the hardest part for most customers is understanding how to direct users to a single, virtual IP address on a load balancer and still have the middle tiers respond appropriately. The key to the middle tier configuration is usually the configuration of virtual hosts as well as the configuration of the Port and Listen directives. When using SSL, additional considerations are necessary; these will be covered later.

The Port directive should be set to the port number that the client browser uses to access the site. This will usually not be the same port number that the HTTP server is listening on, but will be the port number that the load balancer listens on. In most cases, this will be port 80.

The Listen directive should be set to the port (in some cases, it may be set to the `ipaddress:port`) that the HTTP server listens on. This will almost never be the same as the Port directive since there will likely be a load balancer in "front" (looking from the perspective of the end user browser) of the HTTP server. So, it doesn't really matter what port the HTTP server listens on as long as the Port directive is set properly. The OC4J containers also inherit the value of the Port parameter.

Besides the HTTP server configuration parameters Port and Listen (which may appear multiple times if you have multiple VirtualHost sections configured), some applications may also require reconfiguration. The only default application from the

middle tier that requires reconfiguration is the Oracle Portal product. To reconfigure the Oracle Portal product, you will use the `ptlconfig` tool and a configuration file.

MULTIPLE INFRASTRUCTURE TIERS

Balancing load among multiple infrastructure tiers is very similar to load balancing for middle tiers where the HTTP server is concerned. The `Port` and `Listen` directives in the HTTP server are pretty much the same. The primary differences are in reconfiguring the applications that typically reside in the infrastructure tier. Those applications are Delegated Administration Service (DAS) and Single Sign-On (SSO).

There is no special step required to reconfigure the DAS application. However, the URL for the DAS application is stored in a location in OID. It is relatively easy to change the URL, once you find the location where it is stored. The attribute to change is `orcldasurlbase` in the entry with DN: `cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext`. Once that change is made to handle the DAS location change, you will need to login to Oracle Portal and refresh the OID parameters under `General Settings > OID/SSO`.

Finally, the SSO server must be updated to know its new URL location. The reconfiguration for SSO is performed by the `ssocfg.sh` script using a simple syntax. All of these changes and the overall configuration process are well-documented.

PERSISTENCY REQUIREMENTS, WAYS TO ENSURE PERSISTENCE

Some applications deployed in an Oracle Application Server require state to be maintained. While Oracle Portal maintains state in the database, the DAS application keeps state in the middle tier. That means that you must either replicate the session information between all the OC4Js in the infrastructure tiers or ensure that all requests from an individual end user are routed to the same infrastructure tier. In the subsections that follow, we discuss two methods that are often used when session replication is not configured.

LOAD BALANCER PERSISTENCY

Load balancers are responsible for handling requests from clients and routing those requests to back end servers that will satisfy the requests. When an application creates and maintains state in the middle tier, it's important that each request from the end user is routed to the same middle tier server so that the state can be read and maintained by that user's interactions. Load balancer persistency is sometimes called "sticky sessions" and is often implemented with cookies planted by and read by the load balancer device to "remember" where that browser's session needs to be routed. Other times, the load balancer will keep a small database of all browser sessions and their associated back end destination.

MOD_OC4J PERSISTENCY

Persistency can also be an issue between the HTTP server and the OC4J servers that are associated with that HTTP server. The `mod_oc4j` module handles all the HTTP requests that are destined for OC4J containers and routes them appropriately. When multiple OC4Js have the same URLs available (that is, the same application is deployed in multiple OC4Js), `mod_oc4j` must decide which OC4J will be asked to satisfy the request. In a configuration where the OC4Js are not clustered, the `mod_oc4j` must ensure that the request is routed properly to the same OC4J.

In order for `mod_oc4j` to handle sticky sessions properly you must review the `mod_oc4j` configuration parameters. In recent versions, this has become somewhat trickier due to the lack of static configuration parameters for OC4J-resident applications. In current releases, all `mod_oc4j` applications register with `mod_oc4j` dynamically at startup time whereas in previous releases (9iAS), the `mod_oc4j` configuration was stored statically in the `mod_oc4j.conf` file. See the articles listed at the end of this paper for pointers on properly handling the `JSESSIONID` cookie when using multiple OC4Js and multiple applications. The default configuration will likely not serve the needs of those environments that use multiple web applications in a single or multiple containers.

SSL

Secure Sockets Layer (SSL) is commonly used with HTTP to encrypt traffic between the client browser and the first contact endpoint on the server side (this is usually the only link that crosses the "unfriendly" internet. SSL is a protocol extension that encrypts all traffic in a request. While it is commonly used with HTTP, other protocols like LDAP also can use SSL.

In an OAS environment, SSL is usually used to encrypt HTTP traffic between client and server. With some enterprise deployments, the server-side SSL endpoint can be one of many different options. In this section, we'll describe each of the common deployment options and some of the benefits and disadvantages of each.

In OAS environments, using SSL with Oracle HTTP Server, Oracle WebCache or Oracle Internet Directory (with authentication) requires an Oracle Wallet, managed by the Oracle Wallet Manager utility.

MIDDLE TIER SSL

With the middle tier, SSL is commonly used only if the applications hosted there have some sensitive data or purpose. Since the majority of user-generated traffic will be to the middle tier, SSL is not usually used unless necessary since it can impose a significant performance penalty in most deployments. It is also possible to create a slightly more complex application environment by having the application use SSL for some application modules, but not for others. If SSL is required for a large deployment, it is most common to offload the SSL encryption to an accelerator device which is a common component for load balancers to incorporate.

In later versions of OAS, the application server includes a tool called SSLConfigTool to help with SSL configuration. The OAS 10.1.2.0.2 documentation discusses the tool and its usage in Chapter 14 of the OAS Administrator's Guide at http://download.oracle.com/docs/cd/B14099_19/core.1012/b13995/ssl_config_tool.htm.

TERMINATION AT WEBCACHE

Since the webcache is often the first OAS component accessed in many middle tiers, it is a logical place to terminate the SSL connection from the client. Configuration of an HTTPS listening endpoint in webcache is relatively easy in the administrative GUI, but you must have a valid wallet already created. As always, if you run OAS on a UNIX-like platform and want to run using one of the default low-numbered ports (like 80 or 443), you must also configure webcache to run as the root user using the `webcache_setuser.sh` script.

Once you configure a listening endpoint to handle the HTTPS requests, then you need to amend the site definitions and site-to-server mappings to properly route requests that originate on the HTTPS port to the proper back-end HTTP server(s). The requests to the back-end HTTP server(s) can be sent via HTTP calls (which terminates the SSL encryption at the webcache) or via HTTPS calls (which terminates the SSL from the client at the webcache and then recreates new HTTPS requests to the back-end server(s)). In the latter case where outbound requests from webcache to the back-end servers are sent via HTTPS, the webcache has to decrypt incoming client requests, process the request per the caching and routing rules, and then create a new encrypted request to send to the back-end server(s). Note that since webcache communication to back-end components usually occurs on local networks that are behind the border firewall, most sites choose not to encrypt communication between the webcache and the back-end HTTP servers.

Note that using the webcache to terminate SSL requires decryption in software. There are a few hardware vendors supported by Oracle for hardware offload of the SSL processing, but we have yet to encounter any customer that have deployed such a configuration or even seriously considered it. With all the encryption and decryption occurring in software, you can anticipate a significant amount of CPU usage by the webcache processes. In such deployments, it is not unusual for the webcache to be deployed on dedicated servers instead of being collocated with other OAS components on the middle tier.

TERMINATION AT HTTP SERVER

Terminating SSL traffic at the HTTP server is really only an option in environments that do not use the webcache component. The same considerations should be made for the HTTP server machine as discussed in the previous section regarding webcache's higher CPU utilization. Encryption and decryption operations used by SSL are implemented using stream ciphers which are generally considered some of the fastest encryption and decryption methods, but these ciphers still create a very significant CPU impact.

To terminate the HTTPS requests at the HTTP server, Oracle uses a separate port-based virtual host configuration to handle the requests to the additional listening port. This configuration is specified by default in the `$ORACLE_HOME/Apache/Apache/conf/ssl.conf` file should you need to make any manual configuration changes after using the SSLConfigTool utility.

TERMINATION AT LOAD BALANCER

For many enterprise deployments, especially large ones, it is common to terminate SSL at the load balancer. This capability is sometimes an add-on feature for the load balancer and other times it is implemented by a completely separate SSL accelerator appliance. These types of devices serve as the network endpoint for the user's HTTPS requests, "strip off" the SSL layer and pass along the remaining HTTP request (non-encrypted at this point) to the next component in line which is most likely the load balancer, webcache or HTTP server.

The load balancer module or separate appliance implements the SSL algorithms into hardware devices which are optimized to perform the stream cipher operations very quickly. These devices can offer performance specifications that result in very little overhead, especially when compared to using software SSL termination as described earlier in this section.

Configuration for hardware SSL termination devices is proprietary. However, it is important to note that when configuring OAS to allow another device to accept the first connection from the client browser, the Port and ServerName directives in the HTTP server configuration must match what the client browser enters to access the site. Additionally, the webcache configuration should include site to server mappings and site definitions that include host names and port numbers that are used by client browsers to access the site, even if the webcache doesn't necessarily listen on those host and port combinations. Finally, it is important to remember that every HTTP or HTTPS call has some overhead the first time a client accesses the server. This is the reason for HTTP keepalive—to avoid that overhead on successive, consecutive calls. With SSL, that initial overhead is much larger since you're not only making a TCP/IP handshake, but also setting up the encryption tunnel between client and server (which negotiates the encryption algorithms to use, their strength, etc). For this reason, it is especially important to make sure that you either keep the client routed to the same back-end server if SSL is passed through to the back end or terminate SSL on the load balancer to ensure that the SSL overhead is only encountered once per session.

When terminating SSL at a load balancer (also known generically as “reverse proxies”), the HTTP server still needs to know that the self-referencing URLs it creates are going to be appearing at a client browser that is using HTTPS to communicate on a port that is likely not the same as the default HTTP port. In order for the HTTP server to know that the client is using SSL (even though the HTTP server isn't using HTTPS directly), directives must appear in the configuration (usually within a VirtualHost section) to specify HTTPS is in use. For Oracle HTTP server, use the SimulateHttps directive which directs HTTPD to treat requests as though they were HTTPS requests even if they were received via HTTP. In order to use this directive, you'll first have to load the mod_certheaders module. The standard documentation provides good instruction on this configuration at http://download.oracle.com/docs/cd/B14099_19/web.1012/b14007/confmods.htm#sthref691.

INFRASTRUCTURE TIER SSL

The infrastructure tier is frequently configured to use SSL since it handles authentication for the application server. By default, authentication is handled by Single Sign-On (SSO) which resides in the infrastructure tier. Additionally, a lot of personal information can be obtained via the Delegated Administration Service (DAS) which is also accessed in the infrastructure tier. With these sensitive applications residing in the infrastructure tier, SSL seems almost like a necessity and is certainly a best practice.

Since there is no webcache configured in the infrastructure tier, all SSL requests must be handled by HTTP server or an external SSL hardware accelerator or load balancer device.

To configure SSL for the infrastructure tier HTTP server, you can use the same SSLConfigTool which will assist with configuring the HTTP server properly and then properly reconfiguring the SSO and DAS applications as necessary. In older versions, you had to first configure the HTTP server manually by making changes to the httpd.conf and ssl.conf files. Once the configuration changes were in place, you had to run utilities to reconfigure SSO (ssocfg.sh) and the DAS configuration (by making changes to the appropriate LDAP entry and refreshing the DAS parameters in the Portal application).

If terminating SSL for the infrastructure tier on a load balancer or reverse proxy, you'll need to use the mod_certheaders directive SimulateHttps just like in the middle tier as discussed above.

SSL BETWEEN COMPONENTS WITHIN THE APPLICATION SERVER

In high-risk network environments and/or highly secure environments, it may be necessary to encrypt communications between components.

Though less common, SSL can be additionally configured:

- Between webcache and HTTP server
- Between SSO and OID
- Between DAS and OID
- Between HTTP and OC4J

These configurations are complex and not documented as well as more common configurations. Since these are so much less common, we won't get in to the details here, but wanted to mention the possibilities for completeness.

REWRITING URLS

There are many cases where rewriting URLs is necessary to ensure proper function of OAS and its applications. Rewriting URLs utilizes the mod_rewrite module from the core Apache HTTPD server and is configured using directives in the

httpd.conf file that include standard regular expressions. Deep knowledge of regular expressions isn't generally a prerequisite for OAS administrators, so decoding what a given regular expression does is usually one of the hardest parts of URL rewriting.

APPLICATION REQUIREMENTS (PORTAL, DAS, CUSTOM APPS, ETC.)

Oracle Portal and DAS have some requirements for rewrite rules. In default configurations, you'll find rewrite rules for default portal URLs and shortcut URLs for DAS like the ones below:

```
RewriteRule ^/oiddas/provisioning$ /oiddas/ui/oidapshome [R]
RewriteRule ^/oiddas/das$ /oiddas/ui/oiddashome [R]
RewriteRule ^/oiddas/eus$ /oiddas/ui/oideushome [R]
RewriteRule ^/oiddas/install$ /oiddas/ui/oidinstallhome [R]
RewriteRule (^/pls/portal$) /portal$1 [PT]
RewriteRule (^/pls/portal/.*) /portal$1 [PT]
```

Sometimes custom applications can also use URL rewriting to handle permanent changes in a URL's location without using browser redirection. For example, consider the rewrite rules below:

```
Alias /tpsredirect "D:\apps\wsroot\appsgateway\tps"
RewriteRule ^/$ https://tps.dannorris.com/tpsredirect [R,L]
RewriteCond %{REQUEST_URI} !^/tpsredirect
RewriteCond %{REQUEST_URI} !^/tps/
RewriteRule ^/(.*) /tps/$1 [PT]
```

The first line above sets up an alias to make sure that `/tpsredirect` will access a directory on the filesystem. Presumably, that directory has an index file that redirects the browser to another location. The first `RewriteRule` line says to see if the request matches `^/$` which means start at the beginning (^) look for a slash and then the end of the string (\$) immediately follows. That's basically a request of exactly `"/` which is the root page of the site. If someone requests that page, send them to the URL that follows. The `[R,L]` mean that we will send the browser a redirect (R) and then stop processing any further rules (this is the Last rule).

Assuming that first `RewriteRule` doesn't match, we continue processing with the `RewriteCond` statements. These two lines are conditions that determine if the `RewriteRule` that follows them will be executed. Consecutive `RewriteCond` statements are always ANDed together. These two lines say that if the request doesn't start with `/tpsredirect` and doesn't start with `/tps/`, then execute the `RewriteRule` that follows. The `RewriteRule` says that we'll match any request starting with a slash (^/) and the parenthesis around the `(.*)` mean that we will capture the remaining part of the request (since `.*` matches everything to the end of the line) and place it in a temporary variable. We'll then translate the incoming request to `/tps/$1` which means we'll take what we "captured" with the parenthesis earlier on the line and prepend `/tps/` to it. Finally, the `[PT]` says that we'll modify the request in this manner and then pass it through to Apache without telling the browser what we're doing inside the server configuration.

VIRTUAL HOST CONSIDERATIONS

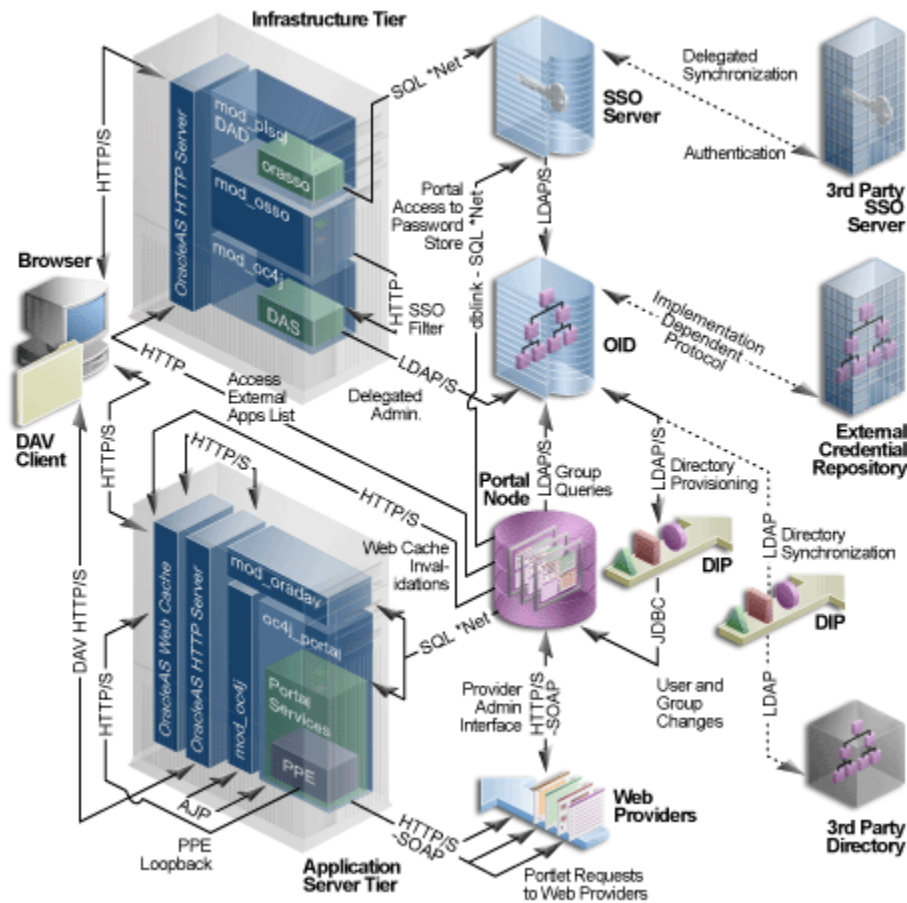
When using URL rewrites with `VirtualHosts`, it is important to remember that rewrite configurations, unlike some other configuration parameters, are not inherited by `VirtualHost` sections. That means that you'll need a `RewriteEngine` on directive in each `VirtualHost` where you wish to use some `RewriteRules`. When you set some `RewriteRules` in the base HTTP server configuration (not within any `VirtualHost`) and want to repeat those same statements inside a `VirtualHost`, you need to enter the following two directives inside the `VirtualHost` section:

```
RewriteEngine on
RewriteOptions inherit
```

These two directives will first enable the `RewriteEngine` for the specified `VirtualHost` and then will import all the `RewriteRules` from the base configuration into the `VirtualHost`. This allows you to configure `RewriteRules` "globally" and then just maintain them in a single location instead of having to copy and maintain them in each `VirtualHost`. While this may not seem like a serious issue when you maintain a few `VirtualHosts`, it becomes exponentially more difficult when you have more `VirtualHosts` to maintain.

ORACLE PORTAL ARCHITECTURE

This section is a brief introduction to Oracle Portal's architecture. The standard documentation does a sufficient job of explaining most of the major components and how they interact. However, most complex, internet-facing Oracle Portal deployments that involve crossing possibly multiple firewalls usually have trouble due to the architecture employed by Oracle Portal, so it deserves a revisit here.



The Portal architecture involves many moving parts. While some of the components are optional, most of the components shown in the figure above (borrowed from the documentation) are mandatory and must have communication with one another. The value of the above diagram is the protocol listing used by each of the components and in which direction communication travels. No doubt your network and firewall administrators will ask you to give them a list of all communications in and out of the application server including protocols and port numbers during a deployment. This diagram is a great start!

The least understood and unexpected communication path for most environments is the outbound HTTP/S requests that originate from the Portal database (the purple cylinder in the diagram above) to the middle tier webcache. To assemble all the components of a Portal page (which may consist of multiple portlets), Oracle launches separate HTTP/S requests for each portlet on the page. Those requests originate from the database. So, when deploying Oracle Portal in a complex and/or restrictive network environment, ensure that the database can perform HTTP/S requests to the application server middle tier. Most other communication paths shown in the diagram above are expected and generally provisioned in the network firewalls by most network administrators.

RAC ENVIRONMENTS

Oracle also makes it possible to provide database failover for all of its application server components. This is largely configured through manipulating TNS entries and JDBC connect URLs across each of the instances. The common RAC services of ICC (Implicit Connection Caching), TAF (Transparent Application Failover) and FCF (Fast Connection Failover) are available for both TNS- and JDBC-based connections. A great article on configuring this can be found at http://www.oracle.com/technology/tech/java/newsletter/articles/oc4j_data_sources/oc4j_ds.htm. For more information about configuring RAC connectivity and networking, you'll find many helpful whitepapers and archived webcasts online at the Oracle RAC SIG (<http://www.oracleracsig.org/>).

For applications that utilize mod_plsql such as Application Express or Oracle Portal, things are treated a bit differently. Mod_plsql utilizes a connection pool, however, needs to have an additional attribute set called PlsqlConnectionValidation, which must be set within the DAD. This executes a quick query before each actual PLSQL query is executed on a connection

to validate if it's still open. If the query fails, it will close the connection, attempt to open a new one, again validate the connection and then execute the query. This obviously has major overhead and should be used only when absolutely necessary. Mod_plsql is able to utilize the same connection failover mechanisms for RAC back-end databases since it uses a full OCI client for connectivity and can use the same TNS entries as any other OCI client to harness the TAF, FCF, and other connection management features.

CONCLUSION

Oracle Application Server is described as a very powerful platform. In many cases, the word “powerful” can be codespeak for “complicated” and “difficult to configure and manage.” In this paper, we’ve described some of the most common advanced configurations and attempted to show how they can be performed in a standard installation. Every configuration is different, but by understanding the architectural components more thoroughly, you’ll find it much easier to know what configuration steps need to be performed and the related dependencies.

It is much more beneficial to learn the system architecture and use reference materials for the particular syntax of a parameter or directive than it is to learn a specific procedure. With advanced configurations, it is unlikely that any two systems are exactly the same, so procedures that are specific to a certain configuration will not be applicable to other configurations.

OVERALL TIPS AND TRICKS

In no particular order, here are some mostly high-level tips and tricks learned after years of working with OAS in complex configurations.

- Boil the configuration down to the least possible number of components to debug an issue. This usually involves stopping and/or disabling components that are not participating in the issue. In some cases, this technique is used to identify the components involved in the issue if you aren’t sure which components are involved. With a good backup in hand, keep in mind that you’ll rare break things worse by turning off components methodically.
- Keep careful notes of every configuration change. These can be in comments in configuration files, but it’s generally better to keep backup copies of configuration files and notes on what changed and when it was changed. When troubleshooting, it may take several iterations of testing before you zero in on the specific issue. Once you find the issue, you may forget all the other changes that need to be reversed if they didn’t contribute to the issue.
- Prepare test plans ahead of time. Make sure you know what has to work and what is optional. This may include testing from multiple network locations, killing off a server to test load balancing, using multiple browser products, checking multiple OS platforms, or adding stress to the system via a simulated load test. The testing should then just be a checklist that you can validate. Testing may take considerable time, but it is critical to a successful project, so plan to spend as much time as necessary testing all necessary combinations.
- When it seems that too many components are a part of the issue, take a step back and whiteboard those components and their interactions. Be sure to include protocols, ports, and parameters or directives that may affect their responses or interactions. Many times, drawing the particulars of a transaction or set of transactions and talking through it will yield new understandings about the possible or most likely areas of interest for debugging.

COMPILATION OF BUGS, ML NOTES, OTHER SOURCES OF MORE INFORMATION

This section simply lists some interesting bugs, notes and other documents we’ve found helpful in our experiences working with OAS in advanced configurations:

- Configuring High Available OracleAS infrastructure with F5’s BIG-IP Load Balancer, An Oracle-F5 Networks White Paper, Aug 2004
- Oracle BUG 5726819, OC4J fails to look up mod_rewrite-rewritten URLs of a web-app
- Metalink Note 453974.1 – Enable log entry output to default-web-access.log in 10.1.3.1 and higher
- Metalink Note 413861.1 – Unexpected session loss when JSESSIONID cookie usage exceeds browser maximum cookie level
- Metalink Note 345167.1 – HTTP Session info lost between two web applications when common “cookie-path” (e.g.”/”) for JSESSIONID
- Metalink Note 340294.1 – Session enabled application requests are wrongly routed to different JVM
- Metalink Note 292972.1 – How to override PATH attribute of the JSESSIONID cookie

- Metalink Note 166078.1 – How to track sessions in OC4J using the URL-rewriting mechanism
- Metalink Note 249411.1 – Integrating mod_rewrite with mod_oc4j into Oracle 9i Application Server 9.0.2.x and 9.0.3
- Metalink Note 231348.1 – Discoverer 9i processes die when the owner logs off the Windows server
- Oracle9i Application Server: mod_oc4j Technical Overview,
http://www.oracle.com/technology/products/ias/ohs/collateral/r2/mod_oc4j_wp.pdf
- Oracle Application Server Administrator's Guide 10g Release 2 (10.1.2), Chapter 14, Using the SSL Configuration Tool, http://download.oracle.com/docs/cd/B14099_19/core.1012/b13995/ssl_config_tool.htm
- Oracle Application Server Enterprise Deployment Guide 10g Release 2 (10.1.2),
http://download.oracle.com/docs/cd/B14099_19/core.1012/b13998/toc.htm
- Apache 1.3 URL Rewriting Guide, <http://httpd.apache.org/docs/1.3/misc/rewriteguide.html>

FROM THE LAWYERS

The information contained herein should be deemed reliable but not guaranteed. The authors have made every attempt to provide current and accurate information. If you have any comments or suggestions, please contact the authors at [dnorris\(at\)piocon.com](mailto:dnorris@piocon.com) and [matt.topper\(at\)oracle.com](mailto:matt.topper@oracle.com)