Dell EMC OpenManage Network Manager

Architecture Overview



Notes and Cautions

A NOTE indicates important information that helps you make better use of your computer.

A CAUTION indicates either potential damage to hardware or loss of data and tells you how to avoid the problem.

© 2017 Dell Inc.

Trademarks used in this text: Dell™, the DELL EMC logo, PowerEdge™, PowerVault™, PowerConnect™, OpenManage™, EqualLogic™, KACE™, FlexAddress™ and Vostro™ are trademarks of Dell Inc. Microsoft®, Windows®, Windows Server®, MS-DOS® and Windows Vista® are either trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries. Red Hat Enterprise Linux® and Enterprise Linux® are registered trademarks of Red Hat, Inc. in the United States and/or other countries.

Other trademarks and trade names may be used in this publication to refer to either the entities claiming the marks and names or their products. Dell Inc. disclaims any proprietary interest in trademarks and trade names other than its own.

2017-6

Rev. A01

Contents

Overview 1
User Interface: Synergy 2
Synergy Technologies 2
Synergy Core Components 3
Extension Strategy 5
Branding Strategy 5
Application Process: Redcell. 6
Component-Based Architecture 6
Redcell J2EE Technologies 8
Oware Services 10
Mediation Services 13
High Availability 14
Contact Information. 15

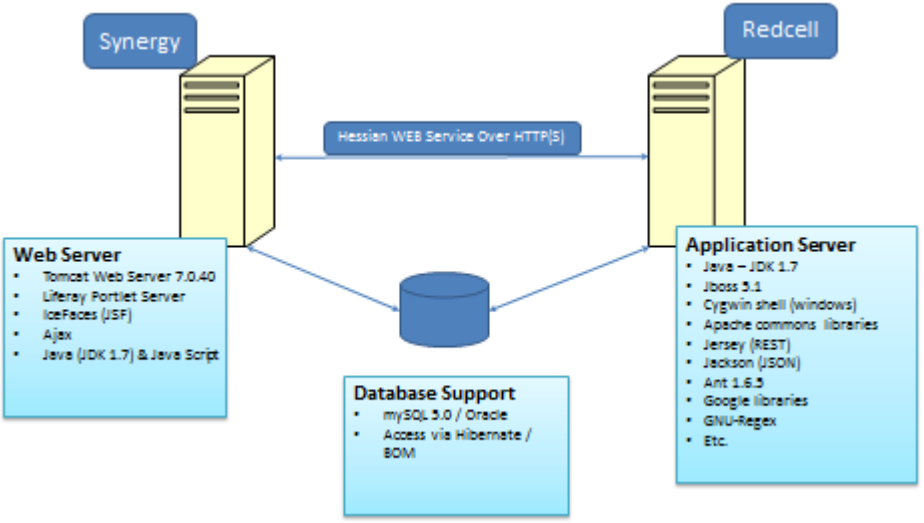
1

Overview

Redcell Synergy is a distributed infrastructure management platform built around J2EE technologies to provide a rich WEB-based user experience. Application features, business processing, and web portal technologies are all Java-based (JDK 1.7) which allows the platform to be deployed on any OS.

The platform utilizes plug-in style architecture and componentization so that new application features and new resource models can easily be added to a deployed package. Following this component-based approach, packages can also be customized to suite the specific needs of the deployment. Components either provide support for new application functionality (OCPs) or they provide support for a new resource type (DDPs) which are responsible for integrating to the device/ vendor/library specifics.

In its simplest deployment the solution consists of a single WEB server process (Synergy) and a single application server process (Redcell). The two processes communicate with each other via a Hessian 2 binary web service; both processes also access a database service.



Depending upon size of the environment, size of the user base or the need for high availability this architecture can scale through server clustering and distributed mediation services to support extremely large deployments.

User Interface: Synergy

Synergy is a WEB based UI component used to expose business information and processes to the end user. It runs on a Tomcat 7.0.40 web server which is bundled and installed as part of the Synergy package.

Synergy Technologies

LifeRay Web Portal

Synergy is built upon LifeRay's WEB Portal product. A web portal allows information to be brought together from diverse sources in a uniform way on a single page. These diverse sources—called portlets—are given dedicated areas on the page to display information and expose application behavior. Each portlet is an independent application responsible for both its display rendering on the front-end and access business processes and data sources on the back-end.

The LifeRay WEB portal adheres to open standards for content, portlets, web services and front-end technologies, which reduces development costs and ensures interoperability with other portlet providers and portals. LifeRay has added multiple features that Synergy and Synergy users can take advantage of for a wide array of web applications and user experiences. These features include:

- Customized presentations based upon user status
- Access to web pages based upon user role and permissions
- Integration of existing web applications
- User collaboration through applications
- Site management capabilities to support multitenant requirements
- Extensive user authentication schemes

JSF & AJAX

JSF (JavaServer Faces) is a specification for building component-based user interface for web applications. As a stateful framework, JSF allows for richer, more natural application behaviors. JSF also abstracts away native browser technologies such as HTML, CSS and HTTP and instead presents a Java-based API for a more natural, seamless integration for developers.

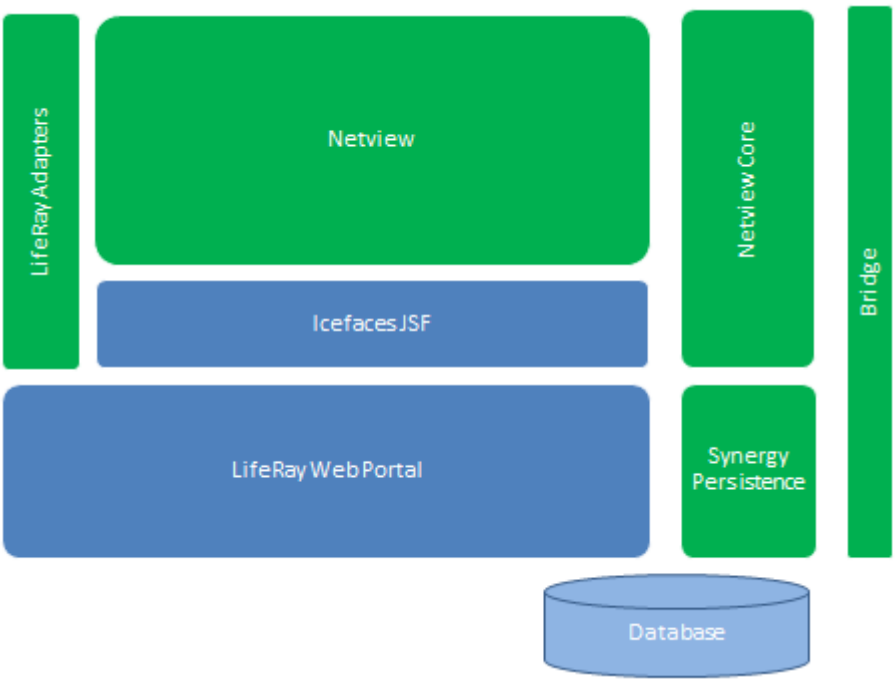
AJAX (push-server) is used in conjunction with JSF to provide asynchronous streamlined updates from the web server.

Synergy utilizes Icefaces (ver 1.8.2) which is an open source JSF provider following the JSF 1.2 specification.

Additional Libraries

- Apache Velocity – Java based template engine
- XStream – serialization of XML objects
- Hessian – binary web service

Synergy Core Components



Bridge

The Bridge component connects the two core processes, the web server and the application server, utilizing a Hessian 2 binary web service. The binary web service protocol makes web services usable without requiring a large framework and without relying on another alphabet soup of protocols. Because it is a binary protocol, it is well-suited to sending binary data without any need to extend the protocol with attachments.

Application Server APIs

- Bridge gives us the ability to expose any application server API with a simple façade service.
- Exposes a clean well documented Java API that can be accessed using any Hessian client.
- Requires all the service objects (interface objects) to be Serializable

Netview

The Netview component is the core UI application. It includes both specific application (portlet) implementations as well as standardized portlets that are built against common application abstractions registered within the application server. These standardized portlets provide common look and feel, require minimal configuration and registration and can easily be extended to provide more customized capabilities.

The Netview component also includes numerous reusable widgets. In many cases these widgets are simply extensions of the IceFaces widgets providing tailored behavior for the domain—eg. IP Address text fields. In other cases the widgets are complex composite widgets that include numerous core widgets. These composite widgets speed up development time and ensure that there is common look and feel across the various applications (portlets).

The Netview component has a direct dependency upon the Icefaces JSF implementation. Other reusable UI functionality has been packaged in separate components so that it can be reused with other presentation providers. As a result of this dependency, any java script needed to customize widget behavior is also packaged within Netview.

Netview-Core

Netview core is series of commonly used frameworks, functions, utilities and services. Including:

- Manager Support
- View Support
- Popup Support
- Menu Support
- Listener Support
- Message Provider
- Registry
- Licensing
- Extension Framework
- Branding Framework

Liferay Adapters

The Liferay Adapters component provides a façade to internal Liferay functions as needed. For example, Synergy's Netview component hooks into LifeRay's user authentication to extract user permissions from the application server at login time.

Synergy Persistence

The synergy persistence component isolates all access to the synergy database which is used to store portal settings and other UI-based information.

LifeRay has its own database lportal which it accesses internally to store site, page, user configuration, etc.

Extension Strategy

The core applications within Synergy are packaged in the Netview component. To support 3rd party application development, Synergy provides an extension strategy which allows 3rd party applications to be picked up and merged into the netview war as if the application was developed natively. The extension strategy is supported via annotations—any portlet carrying the @RegisterExtension annotation and implementing IExtensionProvider will be added to the Synergy registry.

Branding Strategy

Synergy also supports a branding strategy that allows packages to be branded for specific target customers. The branding is supported via annotations. In the absence of a custom-branding class, Synergy uses a default branding implementation class. Branding includes the ability to specify company logo, page and portlet layout as well as which applications are available/ exposed to the end user.

Application Process: Redcell

In a Redcell Synergy deployment, Redcell provides the application features and business logic necessary to manage a networking/computing infrastructure. Redcell uses a component-based architecture in which multiple application packages (OCPs) and driver packages (DDPs) constitute the domain feature-set of the deployment. Additional applications and drivers can be added or upgraded in the deployment as new feature requirements are needed.

Redcell is developed on a J2EE computing platform, JBoss v5.1. J2EE is a widely-used enterprise computing platform for developing large-scale, multi-tiered, scalable applications. Like Redcell, the platform itself incorporates a modular component- based architecture focusing on reusability and loose coupling. Key J2EE technologies utilized within Redcell include:

- Enterprise Java Bean (EJB)
- Java Messaging Services (JMS)
- J2EE Connector Architecture (JCA)
- Java Database Connectivity (JDBC)
- Java Naming and Directory Interface (JNDI)
- MBeans & Java Management Extensions (JMX)
- Java Servlets

Redcell components access these J2EE services through a core platform component—Oware—which acts as an abstraction layer providing middleware capabilities. In addition to technologies listed above, Oware is also responsible for providing:

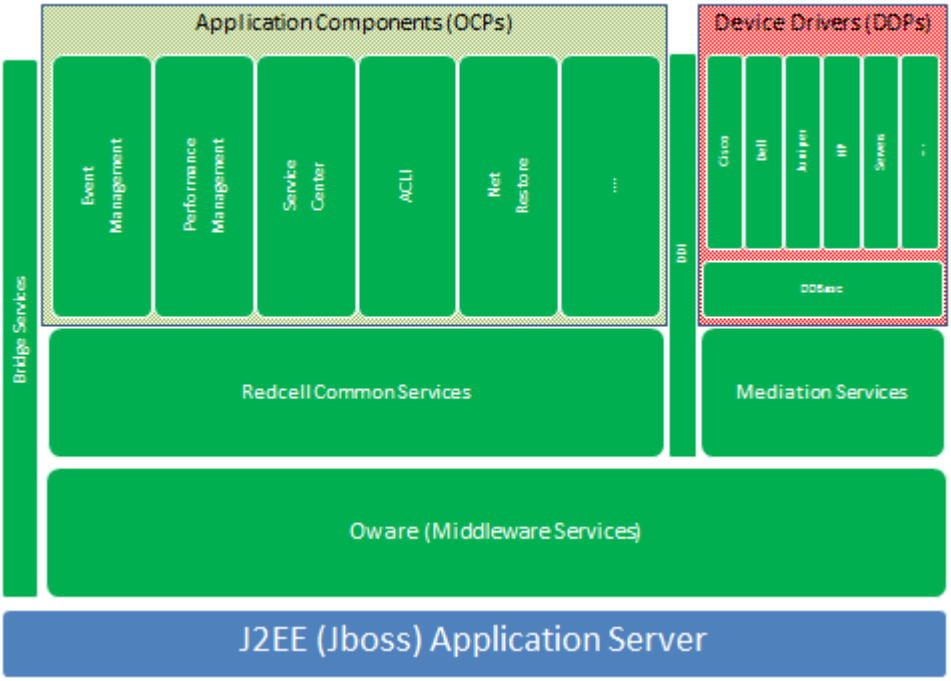
- Java Authentication and Authorization Service (JAAS)
- ORM services
- WEB Services
- Scheduling services
- Job services
- Licensing services

Component-Based Architecture

Redcell Synergy leverages a component-based architecture which allows easy addition and integration of new functionality.

Redcell is a component-based architecture which provides normalized application functionality across a heterogeneous resource environment. Components within Redcell are classified as application components (OCPs or Oware Component Package) or device driver components (DDPs or

Device Driver Package). Application components are responsible for defining a normalized interface for which the individual device driver components provide implementations, if the device supports the application feature. This same concept is extended to other constructs where vendor- or protocol-specifics come into play. This strategy offers both a consistent user experience and the reuse of the core application competency, whereby only a small shim layer is required to support new vendor or protocol semantics.



Component packages—whether application OCPs or driver DDPs are simply jar files that contain the necessary artifacts for the component. All components have a standardized directory structure which is tailored to the Redcell build and runtime environments to enable the following built-in features:

- Property loading
- Library/Jar loading
- Device driver profile loading
- Path management
- Installation support
- Configuration loading

- MIB loading
- Localization support
- Database migration
- Database seeding
- Extended maven build time

Redcell J2EE Technologies

EJBs

J2EE defines three types of EJBs—entity beans, message-driven beans, and session beans—Redcell takes advantage of session beans via an abstracted RuleEngine bean.

The RuleEngine EJB is a stateless session bean whose main purpose is to provide synchronous and asynchronous execution of business logic which can easily be triggered through remote method invocation (RMI) onto separate application server processes. The RuleEngine EJB is built around an abstract interface IOWRule. The primary method—run()—takes an abstract interface IOWEvent as its parameter.

Resource consumption and the number of concurrent EJB threads can be controlled through configuration on the application server, ensuring that server does not exhaust resources during times of heavy application activity.

JMS

JMS provides a common programming interface in Java for messaging—irrespective of the underlying messaging system—allowing messages to be sent and received using a point-to-point or publish/subscribe paradigm.

Redcell uses JMS predominantly for asynchronous rule execution. The parameter event to the rule (IOWEvent) is dispatched onto a message queue. Once the event is de-queued the associated rule is executed as a RuleEngine EJB. Since the RuleEngine EJB is stateless, Redcell reuses the same session bean implementation for message-driven beans.

Redcell also makes use of JMS within its Job notifications sub-system, which allows applications to be notified when a job executing asynchronously completes.

Other uses of JMS within the Redcell application server are dependent upon higher level application components which are free to configure and utilize JMS as needed.

JCA & JDBC

JCA provides a standard mechanism for connecting the application server to an Enterprise Information System. Within Redcell this is used in conjunction with JDBC to provide access to the various databases where JCA connection management capabilities create various connection pools around specific application functionality to ensure a scalable application environment that can support a large number of infrastructure resources and clients.

Oware defines three connection pools against the primary database owbusdb:

- corepool – the core pool is used for internal platform management.
- userpool – the user pool is used to support general business logic executed within a rule engine.
- jobpool – The job pool is used to support jobs and job messaging.

Other application components can define their own connection pools or connection to other 3rd party EIS as needed based upon their specific behavior.

MBean & JMX

An MBean (Management Bean) represents an application resource running within the application server which exposes management properties to the user through the JMX console. The management properties are specific to the function of the MBean but typically include performance statistics, resource usage, error messages, etc. The MBean can also expose properties that can be changed real time through the JMX console to help tune application resource performance.

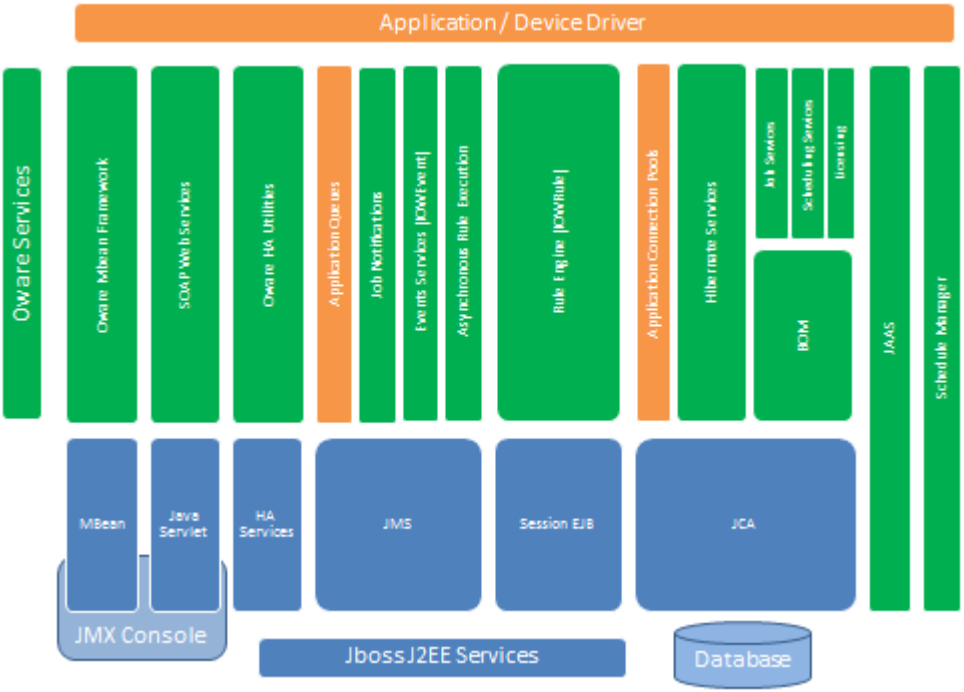
Applications can define their own MBeans using simple annotations provided by and processed through Oware. The default MBean configuration is included as part of the application component and automatically picked up as part of the startup process of the application server.

Java Servlet

Embedded within JBoss is a Tomcat web server which is used to support Java Servlets. A Java Servlet is used to handle client-based HTTP requests to the application server. Oware deploys a servlet to support SOAP-based WEB services.

Other application components can define their own Java Servlets as needed based upon their specific behavior.

Oware Services



JAAS & Licensing

Oware exposes standard JAAS capabilities that applications can use to control access to data and application functionality. Those capabilities are exposed through user management services, in which the current user/ subject gets associated with the current execution thread, and through security services, which provides an API that applications can use to verify that the current subject has the required permission. Applications have the ability to define their own permissions so that access control can be tailored at a granular level sufficient to support any sort of operational user role.

Oware also provides for licensing capabilities in which a set of application capabilities are encrypted and controlled through a license key. Permissions are integrated into the licensing scheme. To assign permission to a user that permission must be specified within a valid license, providing full control over which features can be used within a specific deployment.

ORM

Object-relational Mapping (ORM) services, which essentially map Java objects to a relational database schema are provided within Oware via Hibernate. Oware has wrapped or extended many of the standard Hibernate classes and integrated them into the other Oware services to simplify database access, to provide the appropriate transactional support across components, and to ensure resources are properly managed through lifecycle events.

Database access is predicated upon JCA configuration which is used to identify which connection pool to access which in turn identifies which database to access. This identification is exposed in the Oware services that wrap the native Hibernate API allowing applications to easily define and access their own database.

In addition to Hibernate, Oware uses a proprietary ORM and API called the BOM (Business Object Manager). The BOM API has been deprecated but is occasionally used when accessing older/legacy components. Oware provides the necessary transactional support to utilize both Hibernate and the BOM within the same transaction.

Oware also provides a DB evolve capability to create and alter any database tables as needed based upon Hibernate or BOM metadata. This is a standard capability built into the installation/upgrade process so that tables can be migrated or seeded automatically.

Both Mysql and Oracle databases are supported.

Web Services

Oware provides support for both REST-based and SOAP-based WEB services.

REST-based web services are the preferred technology as they provide a simpler integration for service implementers and client consumers alike. New RESTful web services may be added to the system by registering the location of the resource implementation(s) within a simple XML configuration file in the component package. During startup, the application server will scan classes at the registered location for JAX-RS annotations and add the new resources to the base REST servlet configured by Oware and hosted in JBoss.

Scheduling

Oware provides a scheduling service that allows any rule to be scheduled for future execution. Scheduled items can be **configured** to execute a single time or to execute on a recurring interval with an optional end time. The scheduler has minute level granularity with the application server checking for new scheduled items every 30 seconds.

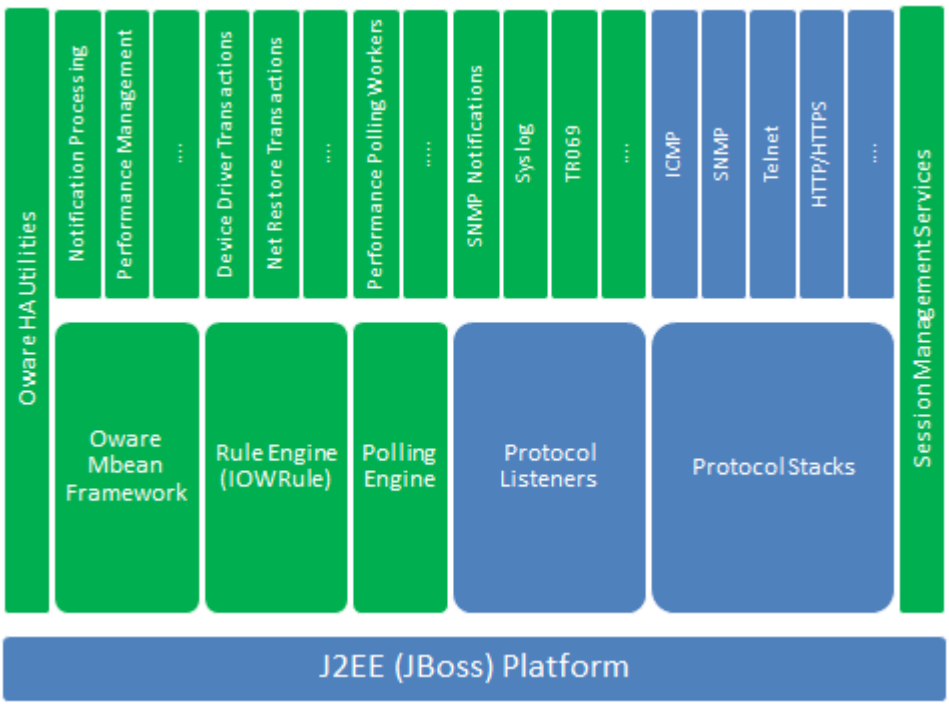
Oware also offers a polling service for applications that require processing on a more frequent basis.

Job services

Job services provide a standard mechanism to report on the current status of a long running job. Jobs are categorized and hierarchical so that sub-jobs can be attached by any component as a business process progresses across multiple applications or into multiple device drivers.

Mediation Services

Mediation services provide most of the standard protocol support required to communicate with the resources (devices) under management. This would include SNMP communication both northbound and southbound, CLI communication through Telnet/SSH sessions, Syslog, ICMP etc. In simple deployments these services are deployed within the same hardware as the application server process. In larger deployments, dedicated mediation servers can be deployed to off-load processing from the application server and to reduce or eliminate latency when communicating with the managed resources.



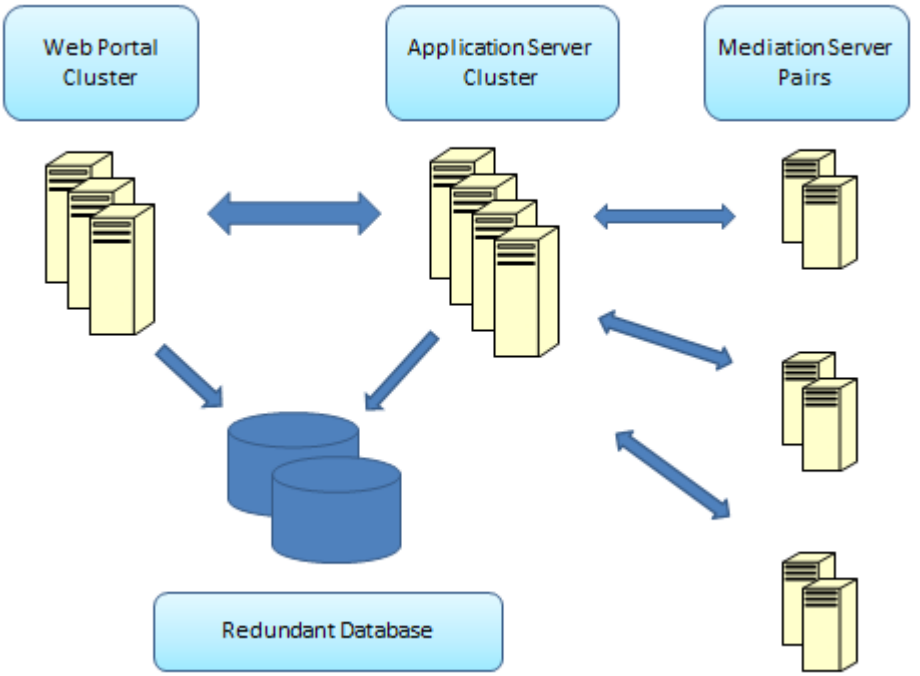
In high availability scenarios mediation servers would be deployed in active/standby pairs.

Applications can deploy their own mediation services specific to the application function that they are providing. For example Redcell's fault processing application (eventMgmt) deploys various protocol listeners and executes as much of the fault processing logic as it can out on the mediation server before sending the fault to an application server. The one caveat of mediation server processing is that no database connectivity/interaction is

available—this restriction enforced so that the database connection pools are not exhausted by device mediation tasks and connections are guaranteed to be available for application/user requests.

High Availability

High Availability (HA) ensures that there is no single point of failure within the system, from a deployment perspective. HA deployments typically require a cluster of application servers, a cluster of web portals, a redundant database configuration, and mediation server pairs (if using distributed mediation services).



Much of Oware's HA capability is provided by the underlying J2EE environment. Oware also provides additional utilities to simplify application development for an HA deployment—including the ability to test whether the current thread is running on the primary application server or a secondary server. (The notion of the primary application server can be paramount in scenarios where something should only be executed a single time, such as a scheduled item—in this case it is the responsibility of the primary application server to distribute the schedule execution load.)

Other utilities include both synchronized and asynchronous broadcast support across the cluster as well as a high availability MBean framework which simplifies the communication and coordination between paired servers.

Contact Information

North America – Corporate Headquarters
4805 Golden Foothill Parkway
El Dorado Hills, CA 95762

USA

T +1 916 673 1100
URL: www.doradosoftware.com
Email: info@doradosoftware.com