The Software AG Common Platform is OSGi-based and oﬀers the possibility to dynamically construct executable instances of various products. It enables applications to be remotely installed, started, stopped, updated, and uninstalled without the necessity of a reboot. Packages and classes can be managed in great detail.

Software AG Web Server based on Apache Tomcat

Toolkit for creating, conﬁguring, deploying, and managing web services named Software AG Web Services Stack. Software AG NERV. Spring Framework.

Software AG Web Services Stack is a toolkit for creating, conﬁguring, deploying, and managing web services. It handles the complex process of processing request and response messages between web services within Software AG products. You can specify individual conﬁguration seings for your web services. This enables you to modify their behavior at runtime and facilitate the correct invocation of the functionality they expose. You can conﬁgure the web services by providing advanced design seings, such as web services addressing, security, and transactional behavior (for example, the service should only be executed on HTTPS with encryption, and the client can only execute the service between 2 and 5 p.m. on Thursdays).

The Java Service Wrapper is an application developed by Tanuki Software, Ltd.

Some Software AG products use the Java Service Wrapper to:

Start and stop the Java Virtual Machines (JVM) in which they run.

You can conﬁgure Java startup parameters such as heap size and classpath.

Record the console output from the JVM in a log ﬁle.

This log includes stack traces that the JVM produces when a process throws an exception and any thread dumps you generate from the JVM.

The wrapper log is particularly useful when a webMethods product runs as a Windows service, because console output is not normally available to you in this mode.

The log ﬁle is named wrapper.log.

Monitor the JVM for various fault conditions and take a speciﬁed action when a fault occurs. You can do the following:

Detect a nonoperational (hung) JVM. After the Java Service Wrapper starts the JVM, it pings the JVM periodically to check whether it is operational. If the JVM does not respond to a ping within a speciﬁed interval, the Java Service Wrapper assumes that the JVM has stopped functioning and restarts it. Each Software AG product conﬁgures this feature diﬀerently; some disable it entirely.

Detect thread deadlocks in the JVM. A thread deadlock occurs when two or more threads try to lock resources in a manner that causes all threads to wait indeﬁnitely. The Java Service Wrapper can monitor the JVM for a deadlock condition and take a speciﬁed action (for example, restarting the JVM) when the condition occurs. For most Software AG products, this feature is disabled by default.

Detect speciﬁed messages in the console output. The Java Service Wrapper can monitor the console output and take a speciﬁed action when a given string of text appears. This feature is often used to watch for out-of-memory messages. Enable you to generate a thread dump when the JVM is running as a service under Windows.

Uncomment the sections that deﬁne the transport receiver and transport sender with name=“tcp” in the client’s axis2.xml conﬁguration ﬁle:

The activation of mail transport in Web Services Stack requires the open source SMTP and POP3 Apache Java Enterprise Mail Server (James) to transfer e-mail messages. After you have installed and conﬁgured your the Apache James server, you must create a mail account that represents the e-mail address of Web Services Stack. You can create additional accounts to correspond to diﬀerent clients. For more information on conﬁguring the Apache James mail server, see the Apache James documentation.

Web Services Stack distributes the Bouncy Castle JCE provider. It is required by the security module (Rampart) for retrieving cryptographic algorithms implementation used in encryption and/or signing of messages. The Bouncy Castle provider is added to the runtime list of Java security providers (when required for the ﬁrst time).

A Universal Messaging Realm is the name given to a single Universal Messaging server. Universal Messaging realms can support multiple network interfaces, each one supporting diﬀerent Universal Messaging protocols

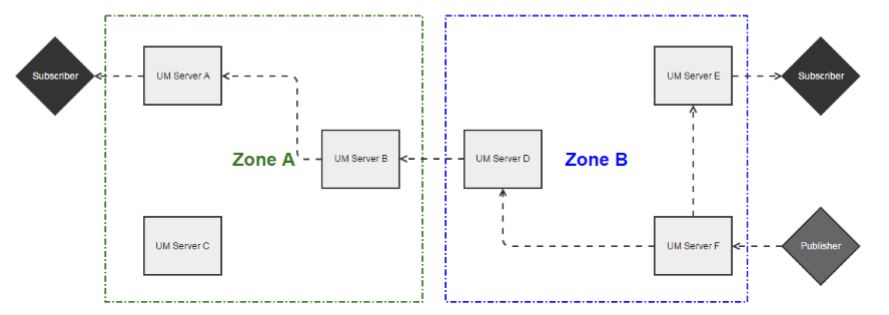
Currently, a realm cannot be a member of more than one zone, so you cannot deﬁne a realm that is common to two zones. Therefore, interest propagation between two zones using a common realm is not supported.

Realm D has a static join to Realm B. It will forward this message over the static join to Realm B.

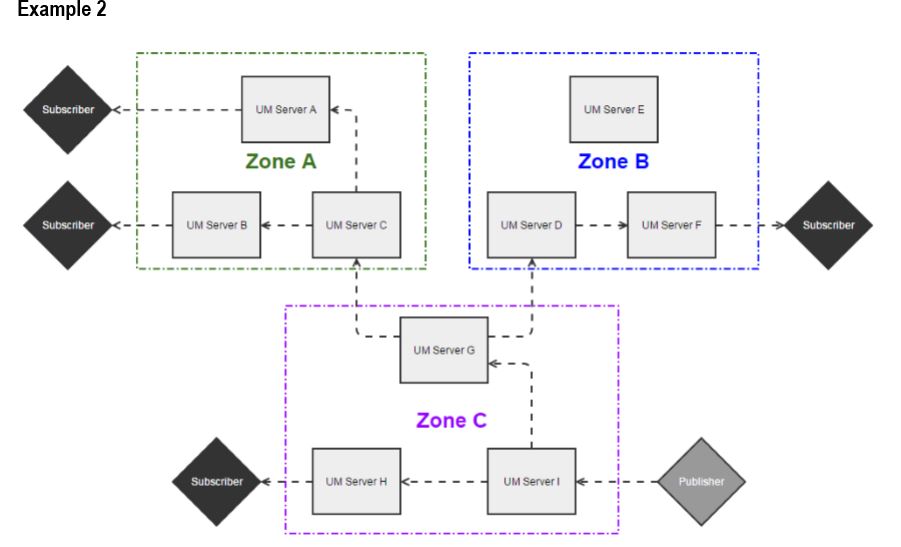
Static join is the same as Route to a diff topic on another Realm/server

Messages gets auto bridged/Forwarded to all members of a Zone.

Members have to register/Subscribe through Zone membership



Realm D receives the message and forwards it to all members of Zone B that have interest. Therefore it sends the message to Realm F. Realm E has no interest and therefore it does not forward the message to this server



Universal Messaging supports three broad messaging paradigms –

1. Publish/Subscribe
   1. asynchronous messaging, Channels/Topics,
   2. supports multiple publishers and subscribers on a single channel
   3. Universal Messaging DataGroups provide an alternative to Channels/Topics for Publish/ Subscribe. Using DataGroups it is possible for remote processes to manage subscriptions on behalf of subscribers
2. Message Queues
   1. message queues decouple the publisher or sender of data from the consumer of data.
   2. only one consumer can read a message from a queue
   3. If more than one consumer is subscribed to a queue then the messages are distributed in a round-robin fashion
3. Peer to Peer
   1. Peer to Peer provides a direct communications path between an instance of a service and the client requiring access to the service

Universal Messaging clients can use a mixture of these paradigms from a single session

Universal Messaging oﬀers, in addition to its standard full-featured client-server API, an extremely lightweight client-client communication API known as the umTransport API.

Historically, messaging architecture has predominantly been based on a 'broker in the middle' approach. This is often referred to as 'hub and spoke'. The broker acts as the communications hub, routing messages between logically decoupled peers:



**umTransport Mode**

umTransport Model The umTransport model is a peer to peer model that allows peers to be aware of how to communicate directly with one another rather than through a broker. In eﬀect, each publisher peer acts like a server, and each consumer can communicate directly with the publishers:



While this model bypasses broker messaging functionality such as persistence or

transactional semantics, it results in a considerably lower latency delivery of information

from a publisher to a consumer. By halving of the number of "hops" between client and

publisher, latency too is effectively halved.

Universal Messaging communication technologies:

1. TCP Sockets: data is transmied directly over TCP Sockets
2. SSL: data is SSL encrypted then transmied over TCP Sockets
3. SHM: data is transmied via Shared Memory (for near-instant access by processes
4. on the same machine)
5. RDMA: data is transmied via Remote Direct Memory Access (for access by processes on a remote machine; requires network adapters that support RDMA)

The following table shows the Communication Protocols supported by each Universal

Messaging Client API:



An RNAME is used by Universal Messaging Clients to specify how a connection should

be made to a Universal Messaging Realm Server.

RNAME string looks like:

**<protocol> :// <hostname> :< port>,<protocol> :// <hostname> :< port>**

**or**

**<protocol> :// <hostname> :< port>;<protocol> :// <hostname> :< port>**

**e.g: nsp://GBADAMOSI.softwareag-gs.com:9000**

nsp://host1:9000,nsp://host2:9000,nsp://host3:9000

where:

<protocol> can be one of the 4 available native communications protocol identifiers

nsp (socket), nhp (HTTP), nsps (SSL) and nhps(HTTPS).

<hostname> is the hostname or IP address that the Universal Messaging Realm is

running on.

<port> is the TCP port on that hostname that the Universal Messaging Realm is

bound to using the same wire protocol.

If a list of RNAMEs is used and the Universal Messaging session becomes disconnected

and cannot reconnect, the API will traverse through the RNAME list until it manages to

reconnect. This functionality is particularly useful within the contexts of both clustering

In addition to the supported protocols shown above, Universal Messaging clients

implemented with APIs that support Native Communication Protocols have a number

of extensions available to them:

*nhpsc:* PROXY CONNECT command via this proxy to establish a connection

*nhpm and nhpsm:* The key difference is that any sessions established using a multiplexed RNAME only ever establish one connection to the Universal Messaging realm server. This is very useful for circumventing browser connection limits while supporting multiple sessions.

In addition to its communications APIs and features Universal Messaging provides

a sophisticated collection of management tools and APIs. These tools and APIs are

designed exclusively for:

"Collection of Statistical Data from Universal Messaging"

"Monitoring of Events"

"Creation of Universal Messaging Resources, ACLs and Clusters"

"Management of Configuration Parameters"

"Seamless Integration with Third Party Enterprise Systems Management Tools"

Universal Messaging's management client, the Enterprise Manager, is wrien using the

same management APIs thus demonstrating the powerful features of these features.

Most client and server induced actions in Universal Messaging result in a management

event being created. Asynchronous listeners can be created using the management API

that enables management clients to capture these **events**

Messaging realms. Universal Messaging's configuration management feature allow

clients to **snapshot configurations** and generate configuration XML files. New realms can

be very quickly configured with the XML files enabling the very fast bootstrapping of

new environments.

**nserver.conf** or **nserverdaemon.conf**

**nserver.conf** is used if you start the realm server manually, e.g. from the

command line.

**nserverdaemon.conf** is used if you start the realm server as a

Windows service or as a UNIX daemon

**Universal Messaging servers also support JMX**

In order to connect to Universal Messaging over the network using JConsole or another

tool that supports JMX, the following JVM system properties need to be added to the

nserver.conf or nserverdaemon.conf

The Universal Messaging Realm Status JMX bean enables access to data

visible in the Enterprise Manager Realm Status view

**Persistence and Configuration**

Universal Messaging uses its own persistent stores that remain relative to its installation

location on your file system. Multiple realms can be configured from a single

installation, each with their own configuration files and persistent stores for event

storage

Universal Messaging can also act as a web server delivering static and server-generated content to clients. This resolves security sandbox problems and port use.

**Server Failover / High Availability**

**In order to provide your clients with a service that is highly available, clustering is**

**recommended**. Clusters enable transparency across your clients

When a client provides a list of RNAMES as a comma separated list, if each entry in the

list corresponds to realm that is a member of the cluster, then the client will reconnect to

the next realm in the cluster list.

**Data Routing**

Joining a channel to another channel or queue allows you to set up content routing so

that events published to the source channel will be passed on to the destination channel/

queue automatically. Joins also support the use of filters, thus enabling dynamic content

routing.

**Please note that while channels can be joined to both channels and queues, queues**

**cannot be used as the source of a join**.

Channels can be joined using the Universal Messaging Enterprise Manager GUI or

programmatically.

**Hop Count**

**The hop count is the number of *intermediate* stores between the source channel and the**

**final destination**

Joins have an associated hop-count, which can optionally be defined when the join is

created. The hop count allows a limit to be put on the number of subsequent joins an

event can pass through if published over this join. **If a hop count is not defined for a join,**

**it will default to 10.**

prevent channels receiving multiple copies of the same event, Universal Messaging implements **loop detection** on incoming events.

**Multiple Path Delivery**

Universal Messaging users can define multiple paths over **different network protocols**

**between the same places in Universal Messaging**. Universal Messaging guarantees that

the data always gets delivered once and once only

Universal Messaging supports the concept of a **federated namespace**, where realm

servers may be located in different geographical locations but form part of the same

logical namespace

**The entry into a Universal Messaging name space or server is via a custom URL called**

**an RNAME**

Universal Messaging's HTTP and HTTPS drivers **support straight proxy servers** as well as user authenticated proxy servers.

logical steps that need to be followed in order to establish a connection to a Universal Messaging sever (Realm). These involve:

1. Establishing a session,
2. Obtaining a reference to a channel or a transaction, or
3. Registering an object as a subscriber.

Please note that before making an HTTP/HTTPS connection to a Universal Messaging realm server you will first need to add a HTTP/HTTPS interface to the realm.

The channels that consume the most memory are those channels that keep the events in

memory and do not write events to persistent store. These channels are known as *Simple*

and *Reliable*

The Universal Messaging client API also supports a variety of different parameters that

can be specified in the command line of any Universal Messaging Client application.

E:g--- wrapper.java.additional.25=-DMaxFileSize=4000000000

wrapper.java.additional.26=-DLOGLEVEL=3

Universal Messaging supports, such as SSL enabled sockets

(nsps) and HTTPS (nhps). Please note that the example programs contained in the

Universal Messaging package will all work with **SSL** enabled on the realm server.

The certificate requirements differ depending on whether the realms require client

certificate authentication or not. Let

When using JSSE, only 1 keystore file/keystore password and 1 truststore file/truststore

password can be used

Universal Messaging supports the multiplexing of sessions to a specific host in Java, Flex

and C#. This allows the circumvention of connection limit issues by packing multiple

Universal Messaging sessions into one connection, and can be used to allow the same

client to set up multiple subscriptions to a given channel or queue if required.

UM ACL



When the server is configured for client certificate processing the subject is constructed

with the Common Name (CN) of the certificate and the remote host name. This allows

the ACLs to be configured such that not only is the certificate valid but it can only

access the Realm Server from a specific host. The



Universal Messaging comes with a default authenticator implementation that uses a

properties file to define users, groups and permissions (roles). In order to enable it on a

Universal Messaging plugin, the Authenticator parameter needs to be left empty (this

implies using the Default), the Authentication Realm set and one parameter needs to be

set in AuthParameters.

**The necessary parameter is called UserFile** and should point to the full path of a java

properties file, e.g. c:\users.txt. In order to get the Universal Messaging realm server

to encrypt your user passwords, you need to add a property called initialize as shown

below.

**An example of a UserFile defining 3 permissions (roles), 3 groups and 3 users is shown**

**below**:

#Request password initialisation

initialise=true

#Permissions (Roles) Definition

**Clustering** and Realm **Federation** are mutually exclusive. If a realm is a

member of a cluster, you cannot use the realm for federation. Similarly, if a

realm is part of a federation, the realm cannot be used for clustering.

*Each cluster has one realm which is elected as master, and all other realms are deemed slaves*. The master is the authoritative source of state for all resources within the cluster.



Should a realm or location become unavailable for any reason, the cluster's remaining

realms should be able to carry on servicing clients



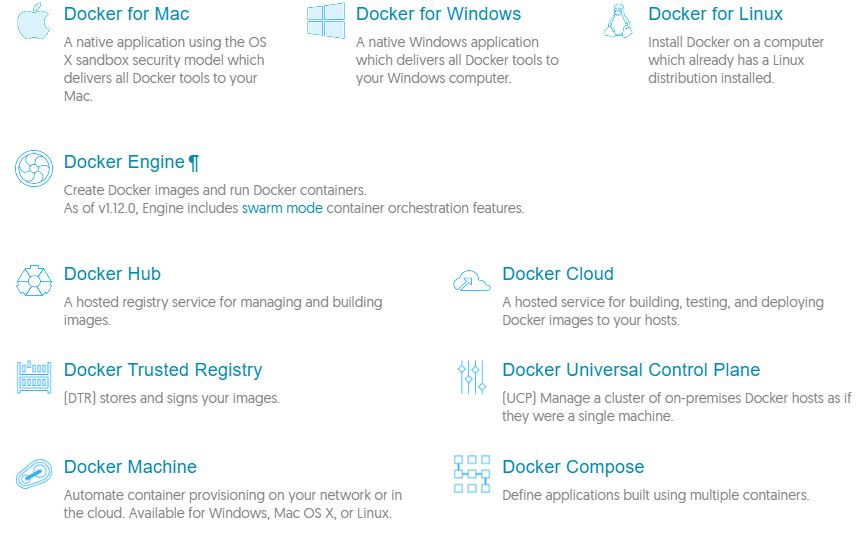
Docker is an infrastructure management solution. The point of docker is to bundle your disk image with your application image to remove traditional discrepencies between deployment, testing, and development environments.

This is all a headache and dockerization attempts to solve this problem by wrapping up your ENTIRE disk image, not just an app itself.

You are guaranteed that your app will run the same after its deployed because it is running on a mirror copy of your development disk that is simply being run on a more robust server that can handle all the traffic and can be managed.

And because containers are lightweight and run without the extra load of a hypervisor, you can run many applications that all rely on different libraries and environments on a single kernel, each one never interfering with the other.

1. Get your code and its dependencies into Docker [containers](https://docs.docker.com/engine/getstarted/step_two/):
   * [Write a Dockerfile](https://docs.docker.com/engine/getstarted/step_four/) that specifies the execution environment and pulls in your code.
   * If your app depends on external applications (such as Redis, or MySQL), simply [find them on a registry such as Docker Hub](https://docs.docker.com/docker-hub/repos/), and refer to them in [a Docker Compose file](https://docs.docker.com/compose/overview/), along with a reference to your application, so they’ll run simultaneously.
     + Software providers also distribute paid software via the [Docker Store](https://store.docker.com/).
   * Build, then run your containers on a virtual host via [Docker Machine](https://docs.docker.com/machine/overview/) as you develop.
2. Configure [networking](https://docs.docker.com/engine/tutorials/networkingcontainers/) and [storage](https://docs.docker.com/engine/tutorials/dockervolumes/) for your solution, if needed.
3. Upload builds to a registry ([ours](https://docs.docker.com/engine/tutorials/dockerrepos/), [yours](https://docs.docker.com/docker-trusted-registry/), or your cloud provider’s), to collaborate with your team.
4. If you’re gonna need to scale your solution across multiple hosts (VMs or physical machines), [plan for how you’ll set up your Swarm cluster](https://docs.docker.com/engine/swarm/key-concepts/) and [scale it to meet demand](https://docs.docker.com/engine/swarm/swarm-tutorial/).
   * Note: Use [Universal Control Plane](https://docs.docker.com/ucp/overview/) and you can manage your Swarm cluster using a friendly UI!
5. Finally, deploy to your preferred cloud provider (or, for redundancy, *multiple* cloud providers) with [Docker Cloud](https://docs.docker.com/docker-cloud/overview/). Or, use [Docker Datacenter](https://www.docker.com/products/docker-datacenter), and deploy to your own on-premise hardware.



The **is\_instance.bat/sh script** creates, updates, and deletes an Integration Server instance.

It also **installs**, **updates**, or **deletes** packages and language packs on an Integration Server

instance.

The is\_instance.bat/sh script is located in the following directory:

***Integration Server\_directory*\instances**

At the command prompt, use the following syntax:

*Integration Server\_directory*\instances\is\_instance.bat *command parameters*

If the command syntax is not correct, the script writes error information to the console

**Configuration seings** for the Integration Server are stored in

the server configuration file (server.cnf). This file resides in the

***Integration Server\_directory*\instances*\instance\_name\* config** directory and contains

parameters that determine how the server operates.

Typically, you will use the Integration Server Administrator to set parameters in the

**server.cnf file**,

**Software AG Command Central**

**Software AG Command Central** is a tool that release managers, infrastructure engineers,

system administrators, and operators can use to perform administrative tasks from

a single location. Command Central can assist with the following configuration,

management, and monitoring tasks:

1. Infrastructure engineers can see at a glance which products and fixes are installed,

and where. Engineers can also easily compare installations to find discrepancies.

1. System administrators can configure environments using a single web UI, commandline

tool, or API so maintenance can be performed with a minimum of effort and risk.

1. Release managers can prepare and deploy changes to multiple servers using

command-line scripting for simpler, safer lifecycle management.

1. Operators can monitor server status and health, as well as start and stop servers

from a single location. They can also configure alerts to be sent to them in case of

unplanned outages.

**To set the session timeout limit**

1. Open the Integration Server Administrator if it is not already open.

In the **Settings** menu of the Navigation panel, click **Resources**.

2. Click **Edit Resource Settings**.

3. Under **Session** in the **Session Timeout** field, enter maximum number of minutes an

idle session can remain active (in other words, how long you want the server to wait

before terminating an idle session).

To set the **Session Timeout** parameter appropriately, you must be familiar with the

clients that use your server. If your clients are all Java programs, you can usually

**Setting the Stateful Session Limit**

Integration Server starts a new session for every remote client that connects to it. This

can be a problem if the server receives multiple requests simultaneously and does not

have the resources to handle them.

The number of concurrent sessions allowed is specified by your license. However, you

can tune performance by seing the stateful session limit using the **Resources** screen in

Integration Server Administrator. When you set a stateful session limit and the number

of concurrent stateful sessions exceeds that limit, the server rejects new requests and

returns an error message to the user.

**Clustering IS**

**Install and configure theTerracotta Server Array.** You must install and configure the

Terracoa Server Array *before* you enable clustering on the Integration Servers

It is recommended that you maintain the same environment for all servers in a cluster,

for example, the same packages of services and ACLs that protect services

Although not required, it is recommended that the servers in a cluster each have the

same server environment. For server clustering to work effectively, you should keep the

following the same on all servers in the cluster:

**Each server should have the same set of licensed capabilities.**

**All servers should have the same packages of services.** For a service to execute on any

server in the cluster, that service must exist in the same package on every server in

the cluster.

Software AG recommends that you **use webMethods Deployer**, or the package replication functionality in the Integration Server Administrator to copy packages to other servers in the cluster, instead of using Designer to copy them.

**Each server should have the same user accounts.** The server uses user account

information to authenticate clients. When a server redirects a request to an alternate

server, the alternate server re-authenticates the user using the credentials supplied to

the original server. If authentication fails, the request fails.

**Access to services should be the same on all servers.** Integration Server uses group

information and ACLs to determine whether a client has access to a service. All

Integration Servers should have the same groups with the same group membership.

Services should be protected by the same ACLs. The ACLs should identify the same

Allow Groups and Deny Groups.

If a request is redirected to a server that denies access to the requested service, the

request will fail.

**Each server should have the same public caches.** One of the purposes of clustering is to

allow a session to be directed to any server within a cluster and for the session to

be processed the same way. Some of that processing may involve caching, in which

case the same caches, with identical configurations, need to be on each server in the

cluster.

**Note:** Software AG recommends that you use webMethods Deployer to copy

cache managers and caches to other servers in the cluster. For information

about webMethods Deployer, see the *webMethods Deployer User’s Guide*.

**Each server should have the same event subscriptions.**Integration Server saves

information for event types and event subscriptions in the eventcfg.bin file. This

file is generated the first time you start an Integration Server and is located in the

following directory: *Integration Server\_directory*/instances/*instance\_name* /config. Copy

this file from one server to another to duplicate event subscriptions on all servers in

the cluster.

**Clock time must be the same on all servers.** The clocks on all machines in the Integration

Server cluster must be synchronized for scheduled jobs to work properly in a cluster.

**Each Integration Server should connect to the same messaging providers.** All servers should use the same messaging connection aliases to connect to the same messaging

providers (Broker and/or Universal Messaging). Each messaging connection alias

with the same name must use the same client prefix. Furthermore, each Broker

connection alias must use the same client group.

**Each server should connect to the same set of databases.** For example, if you are using

the audit database to store audit data, all servers must connect to the same audit

database. If you are using a back-end database for application-related data, all

servers must connect to the same back-end database.

**M**Even Header

**Each server should have its own diagnostic port.** If you plan to troubleshoot using the

diagnostic port, you must configure a diagnostic port for each server in the cluster.

The diagnostic port can access only the Integration Server on which it is defined.

**Each server should have its own Tspace.** If you want the Integration Servers in a cluster

to temporarily store large documents in a hard disk drive space rather than keep

them in memory, you must define a different

**Each server must have the same cluster name.**

**Every server must point to the sameTerracotta Server Array.** The Integration Servers must

use the same Terracoa Server Array URLs and the same cluster name. Additionally,

the Terracoa Server Array that you want to use to store the system caches must

already be configured. Session timeout and time-to-live should be the same on

all servers. If it is not the same, session lifetime will be unpredictable.

For Integration Servers to participate in a cluster, they must share database components

in an external RDBMS. If you installed an Integration Server with the embedded

database, but now want to add it to a cluster, you must switch the Integration Server to

use the shared external RDBMS

**webMethods Mediator** is a service intermediary that enforces the runtime policies you create in CentraSite

Mediator also virtualizes shared services, making it easy to change services.

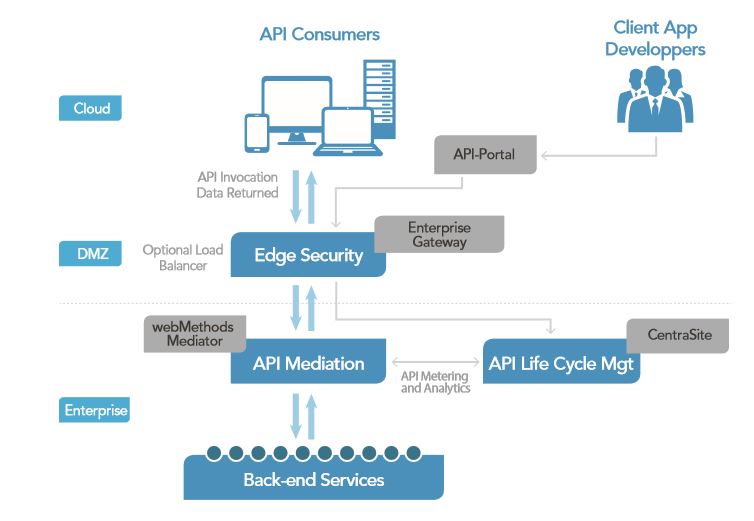
Combined with CentraSite, Mediator provides end-to-end governance of all your services and APIs from development to runtime.

Combined with webMethods Enterprise Gateway, Mediator provides comprehensive API Gateway functionality

Cache the responses to improve performance.

webMethods Mediator creates virtual services from actual services. Virtual services provide loose coupling between consumers and service providers, which shields consumers from changes to underlying services.

Mediator supports seamless HTTP/Java® Message Service (JMS) switching between service consumers and providers.



**CentraSite** is an enterprise API catalog and SOA registry/repository. It can help increase asset visibility and re-use while guiding the development of new APIs and services, and improving collaboration between business and IT

Unified asset catalog CentraSite gives you a central, platform-independent application for defining, describing and storing assets. You can easily catalog all services, APIs, business processes and related assets, such as XML schemas and business rules

CentraSite’s flexible metadata store captures relevant asset information, such as asset capabilities, owner, location, version, classifications and configuration.

**Command Central**

Central repositories are used by template-based provisioning

•  Allows to configure access to SAG SDC and Empower in one central place

•  Supports product and fix repository types

•  Product repos support multiple releases and locations

•  Adds ability to manage and transfer Installer and Update Manager images to the target nodes

•  Provides Web UI and CLI/API

**Installing and Updating Packages on a Server Instance**

When you install or update packages using the installer, you have the option to install packages on the initial instance created by the installer. If you select this option, packages are copied to that instance automatically during installation; therefore, you do not need to run the update command on that instance.

**Note:** Application Platform and ActiveTransfer can be used with the instance of Integration server created by the installer only.

is\_instance.bat **update** -Dinstance.name=*instance\_name* -Dpackage.list=*package\_list*

is\_instance.bat **deletePackages** -Dinstance.name=*instance\_name* - Dpackage.list=*package\_list*

*instance\_name* The name of the instance to which you want to add

packages.

*package\_list* A comma separated list of packages to add to the server

instance. For example, packageA, packageB.

**webMethods Application Platform complements** the webMethods product line

by allowing you to create custom business applications. You can use Application

Platform together with other Software AG products to create entire business solutions.

Application Platform provides a user friendly way for building custom business logic.

When you first install Application Platform, you must add a server runtime for

Application Platform, so that your projects can reference their runtime container. For

the runtime configuration, use an absolute path to the product installation. Runtime

containers are Designer configuration elements that define a set of product libraries that

are included in project classpaths.

1. In Designer, go to **Window** menu, select **Open Perspective**, and then click **Other…**.

2. Click **App Platform** and then click **OK**.

If you have not created a runtime environment for Application Platform, a warning

message will be displayed after opening the App Platform perspective.

After you open the App Platform perspective for the first time, it is cached in the

upper right corner of Designer for quick access.

3. Optionally, if a warning message is displayed after you execute step 2, click **Yes** and

configure a runtime environment.

Designer redirects you to the App Platform Runtime configuration view.

**Adding a Server Runtime Environment**

When you first install Application Platform, you must add a server runtime for

Application Platform, so that your projects can reference their runtime container. For

the runtime configuration, use an absolute path to the product installation. Runtime

containers are Designer configuration elements that define a set of product libraries that

are included in project classpaths.

**To add a server runtime environment for Application Platform**

1. In Designer, go to **Window** menu and click **Preferences**.

2. In the Preferences dialog box, click **Server**, and then click **Runtime Environments**.

3. In the Server Runtime Environments dialog box click **Add**.

4. In the New Server Runtime Environment dialog box, select an Application Platform

server, and then click **Next**.

eader

Currently, the available servers are Integration Server and My webMethods Server,

so you can select **Application Platform Integration Server** or **My webMethods Server**.

5. In the **Designer installation root directory** field, enter the path to the Software AG

installation folder.

Depending on the type of server you are using, keep in mind the following:

If you are configuring Integration Server runtime, the installation folder must

reside in a *Software AG\_directory*, which contains a profiles directory.

If you are configuring My webMethods Server runtime, the installation folder

must reside in a *Software AG\_directory*, which contains a My webMethods Server

directory.

**Important:** The installation root directory of Designer is stored in the Eclipse

workspace metadata area. If you install another instance of Designer

on the same machine, you must not use the same workspace directory.

Using the same workspace directory for more than one instance of

Designer can lead to errors, since both instances will share the same

runtime configuration and will communicate to the same server.

6. Click **Finish**.