opentext™

AppWorks™ Platform High Availability Deployment Guide

Release 16.6

Table of Contents

Chapter 1
Welcome to the High Availability Deployment Guide

This guide helps you to configure OpenText (AppWorks Platform) in a high availability (HA) environment. It serves as a guide to system administrators and other users who have to set up an HA environment for AppWorks Platform.

Understanding the goal

Performance, scalability, and high availability are sought qualities for enterprise business applications. These requirements have gained tremendous importance and prominence because of globalization, the Internet revolution, an increased community of online business users and a greater dependence on shared software services across geographical locations. The High Availability Deployment Guide assists you in achieving optimal performance for AppWorks Platform.

An enterprise software system is highly available (HA) when it is designed to be resilient to various disruptions that can happen in the system, and their dependent resources and infrastructure. These may include system crashes, application crashes, network problems, and so on. High availability ensures continuous availability of application services to the end-users and thereby ensures uninterrupted business. Lack of high availability not only makes the end-user unhappy about the disrupted service but also causes loss of business and damage to the reputation of the business provider. The high availability of systems has gained tremendous importance and prominence due to globalization, Internet revolution, increasing community of online users, and greater dependence on the shared software services across geographical locations.

On the other hand, business providers want their business to grow through continuous expansion of their customer base. Therefore, besides making the software highly available, it must also be flexible to meet the growing demand of resources. In other words, a system must be scalable, be capable of delivering services to the growing number of users, and support the volume of business, without affecting the actual performance.

High availability qualities and requirements

Performance, scalability, and high availability (HA) are critical with make-or-break qualities for any enterprise business application.

- **Performance** - Making the business transactions processed at the required speed and the transaction volume with the efficient resource utilization.
- **Scalability** - Business providers want to grow their business through continuous expansion of their customer base. Therefore, the business applications must be flexible to meet the growing demand for resources. In other words, a business system must be scalable and be capable of delivering services to the growing number of users and the growing volume of business, without affecting the actual performance.
- **High Availability** - High availability ensures continuous availability of the application services to the end users and thereby ensures uninterrupted business by making the business system resilient to the various disruptions that can take place in the system and their dependent resources and infrastructure. These can include system crashes, application crashes, network problems, and so on.

These requirements have gained tremendous importance and prominence through globalization, Internet revolution, increased community of online business users, and greater dependence on the shared software services across the geographical locations.

Before you get started

This guide helps you to configure OpenText (AppWorks Platform) in a high availability (HA) environment. It serves as a guide to system administrators and other users who have to set up a HA environment for AppWorks Platform.

Before setting up high availability for AppWorks Platform, ensure that you have:

- Administrative privileges on the computer where you want to configure AppWorks Platform
- Knowledge of networked computers
- Fundamental knowledge of HA configuration

AppWorks Platform is inherently built for high availability and scalability. The basic need for setting up HA is to have a cluster of computers (nodes) that work together to ensure that services running within the cluster never cease to operate.

Architectural principles

The AppWorks Platform architectural principles that enable enterprises to develop and deploy business solutions that make business operation processes more resilient and adaptive are as follows:

- State SyncUp
- SOA Grid
- Cluster configurations

- No single-point of failure
- No state or session maintenance
- Fail-fast and Self-Healing

Recommended reading

- OpenText CARS Installation and Administration Guide
- AppWorks Platform Installation Guide

Chapter 2

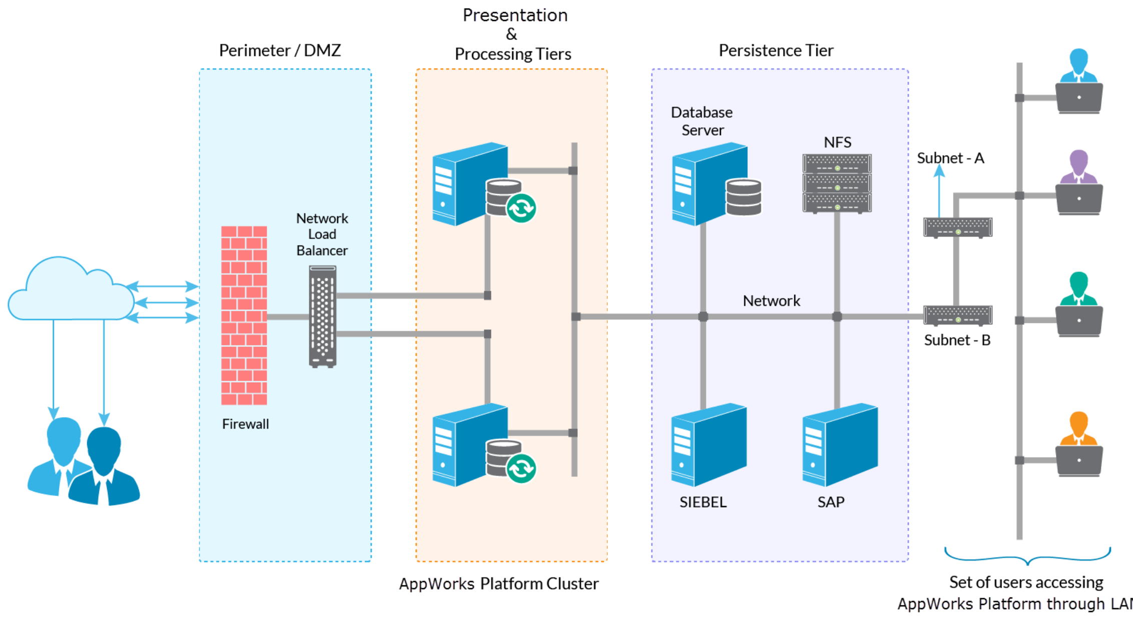
AppWorks Platform and the IT landscape

The IT infrastructure setup is one of the building blocks of any enterprise comprising elements such as a front-end to validate, send, and receive requests, a middleware to process the requests, and a back-end to store the information. The arrangement of these elements enables the flow and processing of information, resulting in fulfilling the requests.

IT architectural landscape at a broader level can be categorized into Presentation, Processing, and Persistence tiers. The elements can be associated as per their underlying functionality with these tiers.

OpenText™ AppWorks Platform aptly fits into the processing tier of this landscape, because it gives complete control over the internal business processes, and serves as a comprehensive enterprise application development platform.

The following diagram provides a visual representation of a typical IT landscape with real-time objects and the associated tiers.



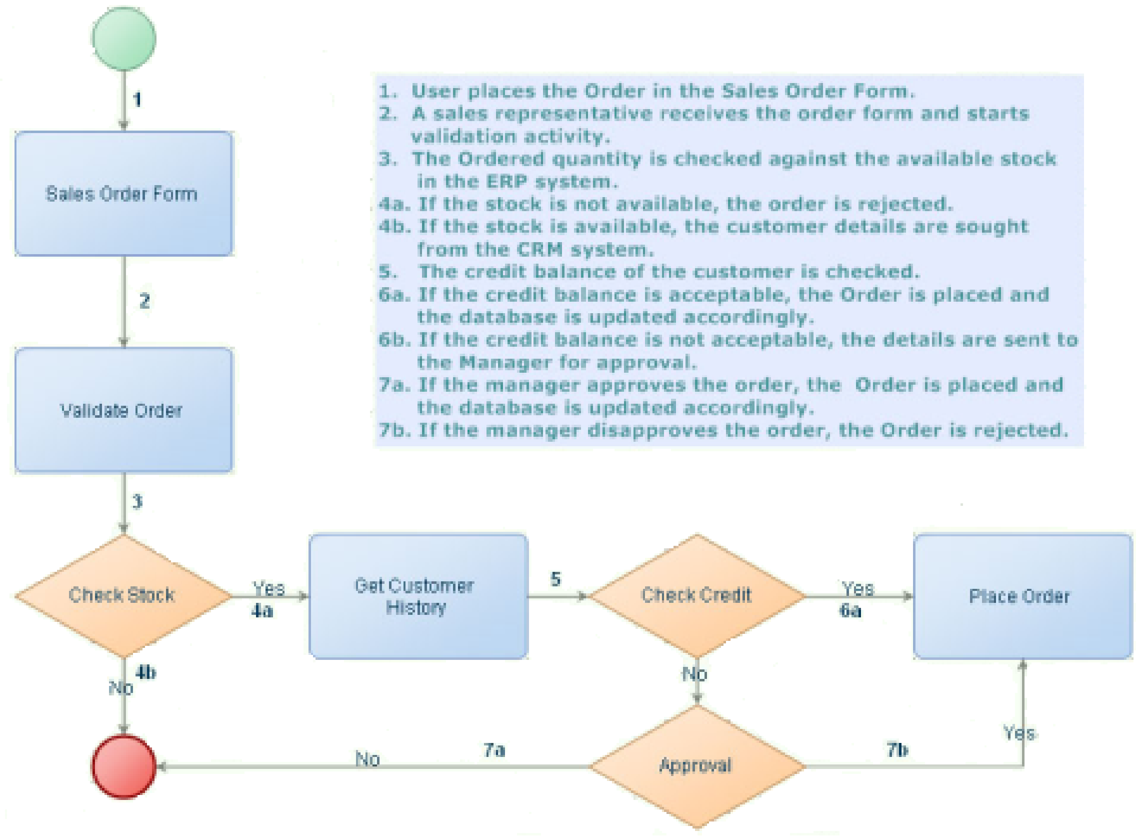
The following table lists the mapping of tiers and objects.

Tier	Objects / Components
Presentation	User Interface, Web Gateway
Processing	AppWorks Platform
Persistence	Database, File system, OpenText CARS

Business situation

To understand the role of AppWorks Platform and non-AppWorks Platform components, consider the business case of an Online Sales Order application. This sample business case outlines the flow at a high level, from the user placing an order via the Internet until its approval. In this process, the order request goes through a series of checks involving human interaction in a few instances and retrieves data from external systems to make the decisions.

The following flowchart depicts the flow of the request through various activities.



1. User places the Order in the Sales Order Form.
2. A sales representative receives the order form and starts validation activity.
3. The Ordered quantity is checked against the available stock in the ERP system.

4a. If the stock is not available, the order is rejected.
4b. If the stock is available, the customer details are sought from the CRM system.
5. The credit balance of the customer is checked.

6a. If the credit balance is acceptable, the Order is placed and the database is updated accordingly.
6b. If the credit balance is not acceptable, the details are sent to the Manager for approval.
7a. If the manager approves the order, the Order is placed and the database is updated accordingly.
7b. If the manager disapproves the order, the Order is rejected.

The following table presents the core components involved in each activity.

Activity	Component	Dependent on
Start	Web Gateway, Web Server,	XForm, SSO, and LDAP Service

Activity	Component	Dependent on
	XMLStore, or Filesystem	Containers
Sales Order Form	XForm, Flow in BPM	BPM, Notification, and email Service Containers
Validate Order	External database (SAP)	SAP Custom Connector or OLEDB/JDBC Connector
Get Customer History	External database (Siebel)	Siebel Custom Connector
Place Order	External database	Web service, SAP Custom Connector
Check Stock, Check Credit, Approve		Notification Service Container

Identifying high availability components

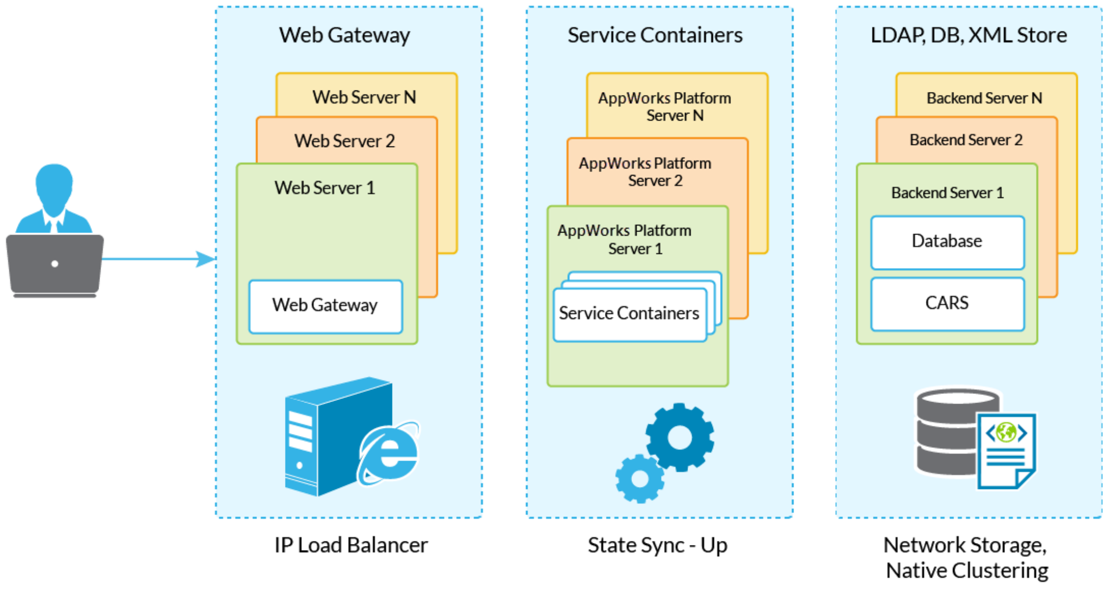
Ensuring seamless and uninterrupted communication between high availability components, which are inter-dependent, is vital for any enterprise. Hence, it is essential to identify the components of the business systems that are required to be in high availability. After you identify these components, the next step is to configure them for HA. In continuation of the business case, the following list of AppWorks Platform and non-AppWorks Platform components are involved in the setup.

AppWorks Platform components

The AppWorks Platform components involved in this flow that must be in high availability are:

- Service Containers and the associated connectors
- Web Gateway
- OpenText CARS

These can be categorized into the following stacks.



Non-AppWorks Platform components

Per the business requirement, a few non-AppWorks Platform components involved in the HA scenario are:

- Network
- Database Systems
- File System
- IP Load Balancer
- Other back-end servers such as email server, ERP Systems

Chapter 3

Configuring high availability

This chapter describes the procedure to configure the following components of AppWorks Platform to be highly available:

- OpenText CARS
- AppWorks Platform
- Service Containers
- Web Gateway

Note: You may need to refer to *OpenText CARS Installation and Administration Guide* and *AppWorks Platform Installation Guide* occasionally while configuring HA.

The following procedure assumes a failover or load balancing setup with only two nodes, which are referred as Master1 and Master2.

Ensure that the non-AppWorks Platform components are configured to be in HA. The procedure to configure HA of non-AppWorks Platform components is not discussed in this document.

Configuring for OpenText CARS

The names, configurations, access permissions, data source configurations of the artifacts such as SOAP Nodes, Service Containers, Users, Organizations, Method sets, Methods, and so on are stored in OpenText CARS or LDAP.

Note: To prevent OpenText CARS data loss due to outages or some foreseen and unforeseen issues, replicate OpenText CARS to make it highly available.

To configure for high availability:

1. Identify the nodes that are required for setting up AppWorks Platform high availability. You need at least two nodes for this setup, a Primary node (Master1) and a Secondary node (Master2).
2. Install OpenText CARS on Master1.

   Note: While installing OpenText CARS, provide a suffix that is the root of the OpenText CARS database. Provide the same suffix for both the nodes; else, the Distinguished Name for each of the LDAP entries will be different.

3. Install OpenText CARS on Master2.
4. Copy the OpenText CARS certificates (.cer) from the <*OpenText CARS_installdir*> of Master1 into the certificates folder of <*OpenText CARS_installdir*> in Master2.
5. Similarly, copy the OpenText CARS certificates (.cer) of Master2 into the certificates folder of OpenText CARS installation in Master1.
6. Set up replication between both the OpenText CARS installations by adding the syncrepl directive to `slapd.conf` file on both the instances.
   You can find this file at the following locations:

   Windows: <*OpenText CARS_installdir*>\bin

   Linux: <*OpenText CARS_installdir*>/sbin

   For more information on the procedure to add syncrepl directive, see Modifying `slapd.conf` file of the Master OpenText CARS section of the *OpenText CARS Installation and Administration Guide*.

7. After configuring the OpenText CARS Failover setup, evaluate the validity of the configuration to ensure successful LDAP synchronization through any third-party LDAP explorer, such as JXplorer tool.
   You can evaluate through any of the following scenarios:
   - Insert an LDAP entry into Master1 OpenText CARS. Check if this is replicated in Master2 OpenText CARS.
   - Insert an LDAP entry into Master2 OpenText CARS. Check if this is replicated in Master1 OpenText CARS.
   - Modify an attribute or delete an LDAP entry. Check if these are replicated.

Configuring AppWorks Platform for high availability

Install AppWorks Platform in the Primary-Distributed Installation mode. In this mode, two or more installations of AppWorks Platform share the same directory service. For the procedure to install AppWorks Platform, see Installing in a Primary-Distributed Setup section in the *AppWorks Platform Installation Guide* and the *AppWorks Installation Guide*.

To configure high availability for AppWorks Platform:

1. Install AppWorks Platform on Master1 and load all the application packages using the server URL.

   Caution: Do not use the load balancer URL while loading the application packages. Do not install AppWorks Platform on multiple nodes simultaneously. Do not install the application packages using the load balancer URL. Always install the application packages using the direct server URL.

2. Install AppWorks Platform on Master2 in Distributed mode and provide the AppWorks Platform instance name identical to the Primary instance name. For more information on distributed installation, see Distributed Installation section in the *AppWorks Platform Installation Guide*.

   Note: The AppWorks Platform installation on Master2 must point to the OpenText CARS of Master1 during installation; else, the monitor will fail to start due to an invalid license key error. Restart the OpenText AppWorks Platform (<instance name>) monitor service after deploying the application packages in each Primary and Distributed node.

3. Read the following instructions before proceeding with the installation:
   a. Ensure that the Primary AppWorks Platform installation is up and running.
   b. While installing AppWorks Platform on Master2, provide the OpenText CARS details of Master1 and point to the OpenText CARS certificates available in Master2.
   c. Copy the following from Master1 to a specific location in Master2. Provide this location while installing AppWorks Platform in Distributed mode.
      - OpenText CARS certificates (.cer) from the `<OpenText CARS_installdir>` of Master1.
      - AppWorks Platform certificates folder from `<AppWorks Platform_installdir>`
      - `wcp.properties` file from the `<AppWorks Platform_installdir>/config` folder
   d. Load all the application packages on Master2 using the server URL. Do not use the load balancer URL while loading the application packages.

Note: Ensure graceful shutdown of OpenText AppWorks Platform (<instance name>) while shutting down the node, as there is a possibility of request loss on a non-transactional transport. For more information on the procedure of graceful shutdown, see *Ensuring a Graceful Shutdown of OpenText AppWorks Platform (<instance name>) upon System Shutdown* section of the AppWorks Platform Configuration Guide.

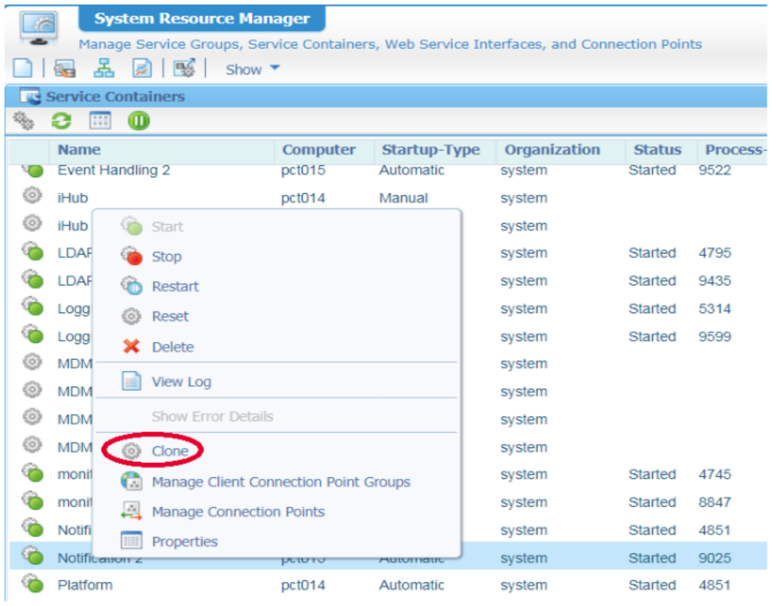
Configuring service containers

When AppWorks Platform is installed in the Distributed mode, by default all the Service Containers are configured on the Primary node (Master1). The Secondary node (Master2) contains the OpenText AppWorks Platform (<instance name>) along with CAP service container such as `CAP@machinename`, for instance, `CAP@srv-ind-svt1w`. Therefore, to ensure high availability, you must configure all the other Service Containers for Master2.

Tip: Based on your business requirements, to enable HA for the Service Containers, while configuring the Service Group properties, select the Failover or Load Balancing options in the Routing Algorithms field. For more information, see AppWorks Platform service containers load balancing and failover support matrix.

To clone the service container:

1. From the start page, open **System Resource Manager**, right-click the required Service Container, and select **Clone**.
   The Clone Service Container dialog box opens.



Name	Computer	Startup-Type	Organization	Status	Process-
Event Handling 2	pct015	Automatic	system	Started	9522
iHub	pct014	Manual	system		
iHub			system		
LDAF			system	Started	4795
LDAF			system	Started	9435
Logg			system	Started	5314
Logg			system	Started	9599
MDM			system		
MDM			system		
MDM			system		
MDM			system		
moni			system	Started	4745
moni			system	Started	8847
Notifi			system	Started	4851
Notification 2	pct015	Automatic	system	Started	9025
Platform	pct014	Automatic	system	Started	4851

2. Select **Master2** from the Computer Name list and click **OK**.
3. Continue with the configuration details of the original Service Container for other fields in the dialog box.

   Note: After cloning the LDAP Service Container, open Management Console > Platform Properties and enter the LDAP server name and port number of Master 2 in the `bus.ldap.processor.host` and `bus.ldap.processor.port` property values respectively.

   Important: Unless you anticipate heavy traffic on your LDAP Service Container, do not configure Simple Load Balancing as the Routing Algorithm.

4. After cloning or creating the LDAP Service Container on the new node, check if the property `ldap.soap.processor.dn` has been added to the `wcp.properties` through Management Console.
   The value of this property should be the Distinguished Name of the cloned/created LDAP Service Container. If the property is not available, you must add it manually.
5. In the corresponding Service Group window, set the required Routing Algorithm.
6. Repeat the procedure to clone all the required service containers.
7. Open the Management Console of Master 2 and provide details of CARS.
8. Click **OK**.
   The LDAP Server Certificate is Unknown dialog box opens.
9. To accept the certificates for Master 2 of AppWorks Platform, click **Accept**.

Configuring the web gateway (TomEE)

Configure your IP Load Balancer for making the Web Gateway highly available. These configurations depend upon the IP Load Balancer you use.

To configure the web gateway:

1. Add details of AppWorks Platform Web servers of all the cluster nodes in the Web servers list of the IP Load Balancer.
2. Perform a health check of the URLs to provide the context sensitive status of a given Web server.
   Though a successful ping on a Web server indicates the availability of a Web server, the required functions, such as AppWorks Platform services may not be available. In such cases, IP Load Balancer can redirect the requests to a fully functional Web server.
   For a properly functioning site you must always use the HEALTH_CHECK_ URL as follows:

   ```
   http://<<computername>>/cordys/wcp/library/util/eventservice/com.eibus.web.tools.he
   althCheck.HealthCheckURL.wcp
   ```

Sharing file systems

Some use cases require interaction with the file system. Therefore, various file system folders must be shared across all the cluster nodes to make the use cases workable in the clustered environment.

Note: Ensure that AppWorks Platform Web server and the OpenText AppWorks Platform (<instance name>) service is running in the context of the network domain user account, and you have read and write permissions on the given network shared folders.

To modify OpenText AppWorks Platform (<instance name>) service startup user account:

1. Navigate to **Windows** > **Services** > **OpenText AppWorks Platform** (<instance name>).
2. Right-click **Properties** and click **Log On**.
3. Modify the account to the domain user account (if the account is not a domain user).
4. Restart OpenText AppWorks Platform (<instance name>).

To run TomEE on the domain user:

1. Navigate to **Windows** > **Services** > **Apache TomEE**.
2. Right-click **Properties** and click **Log On**.
3. Modify the account to the domain user account.
4. Restart TomEE.

wcpdev folder

Use cases such as Exporting PIM report to Excel, GMF print preview, BPM Archive File Content, and so on requires a wcpdev folder.

To change the path of wcpdev for the TomEE application server:

Note: The file `<TomEE_installdir>/conf/Catalina/localhost/wcpdev.xml` contains the path to the `wcpdev` folder; however, this file is overwritten with each fix pack installation. Therefore, do not edit this file. Mount or link the shared folder directly as the `wcpdev` folder.

1. On windows, use the `mklink` command.
   For example:

   ```
   mklink /d  ${CORDYS_HOME}\webroot\wcpdev \\Server\ShareName\Folder
   ```

2. On Linux, use the `mount` command.
   For example:

   ```
   mount //server/sharename/folder ${CORDYS_HOME}\webroot\wcpdev
   ```

Note: Ensure that the TomEE user (the user context with which the AppWorks Platform Monitor is running) has full permission on the mounted folder.

Upload and download folders

If the business use cases involve uploading and downloading files to and from AppWorks Platform respectively, then perform the following steps.

To upload and download folders:

1. Ensure that AppWorks Platform UploadWritePath and DownloadReadPath are pointing to the appropriate shared file system.
2. Restart TomEE.
3. Perform the setting on all the nodes.

Collaborative workspace

For all HA scenarios, including testing and production, configure the Collaborative Workspace synchronization folder on a shared file system. Share this file system between the cluster nodes, allowing all the Collaborative Workspace service containers to access the same synchronization details. The recommended initial size of the shared file system is 1 GB.

Configuring the synchronization folder on a shared file system prevents certain operations that are performed by multiple threads simultaneously. The following is a list of a few scenarios in which such operations are triggered:

- Platform upgrade
- Accessing CARE data for the first time; for example, while working on User-Team assignments or reading Business Calendars
- Accessing staging area for the first time, while working with staging packages

Note: Development on cluster deployments such as HA is not supported. There is no need to configure the CWS sync folder on a shared location. It will not affect the required size of the shared file system.

Licensing

In a highly available scenario, store the license key and usage report on a shared file system. While updating the license, share the information stored on the file system across all nodes in the cluster.

Business activity monitoring

The Business Activity Monitoring (BAM) package can be deployed in multiple nodes of the cluster environment similar to other package deployments. However, it is adequate to deploy the AppWorks Platform BAM MDM Model package in any one of the nodes.

All the BAM service containers in a cluster or a AppWorks Platform instance must always point to the same repository.

Chapter 4

Scenarios for validating high availability

The following scenarios are for validating the high availability of AppWorks Platform components only.

Planned outage

In this scenario, the administrator intentionally brings down one or more AppWorks Platform components from the production environment. Few instances when a planned outage is performed can be while applying patches, re-configuring computer, archiving the data, upgrading the database, application, operating system, or network, and so on.

The following Planned actions can be performed on AppWorks Platform components.

Component	Action	Expectation
Service Containers	▪ Stop the Service Container ▪ Start the stopped Service Container	Despite the disruptions, the business case must continue its execution. Once the stopped component is online, it must start processing the requests. **Note:** This test case is not applicable for service containers configured to run inside TomEE.
WCP Monitor	▪ Stop the Monitor ▪ Start the stopped Monitor	Despite the disruptions, the business case must continue its execution. Once the stopped component is online, it must start processing the requests.
Active LDAP Server (OpenText CARS)	▪ Stop the Active LDAP Server	Despite the disruptions, the business case must continue

Component	Action	Expectation
	Note: Ensure that LDAP service is configured in the Fail-over mode. ▪ Start the stopped LDAP Server	its execution. Once the stopped component is online, it must start processing the requests.
Computer Restart	▪ Restart one of the AppWorks Platform computers **Note:** OpenText CARS must be in-sync with the changes made to the standby OpenText CARS.	Despite the disruptions, the business case must continue its execution. Once the stopped component is online, it must start processing the requests.

Unplanned outage

In this scenario, the administrator abruptly brings down one or more AppWorks Platform components, from the existing production environment. A few examples of unplanned outage are human error, software, hardware, and infrastructure failures, site disaster, natural calamities, and so on.

Note: Before killing a Service Container, ensure that the Automatic Startup option of the Service Container is selected.

Perform the following Unplanned actions on AppWorks Platform components.

Component	Action	Expectation
Service Container	▪ Kill the Service Containers	Requests received after the kill signal is sent must be processed by the corresponding Service Containers of the standby computer. **Note:** By default, the connection points of all the Service Containers are configured with the TCP/IP Socket transport. Hence, during the outage, the requests in the socket buffer are lost. You may have to resend the requests that were sent to the Service Containers before the stoppage. To handle this, you must configure transactional transport like JMS. For more information, see *Configuring JMS as the Messaging Service Using Client Connection Point Groups* topic in the product documentation. **Note:** This test case is not applicable for

Component	Action	Expectation
		service containers configured to run inside TomEE.
WCP Monitor	▪ Kill WCP Monitor on one of the AppWorks Platform computers with at least one active Service Container.	When the monitor is down, route all requests to the standby computer. After the monitor is up, the Service Container of the local computer must start processing the requests.
Active LDAP Server (OpenText CARS)	▪ Kill active LDAP Server (OpenText CARS)	The local LDAP Service Container must create the problem registry and stop accepting requests.
Network Cord	▪ Unplug the network cord on one of the AppWorks Platform computers, with at least one active Service Container.	Unplugging the network cord results in a response timeout. Requests sent during this period are not processed by the Service Containers, because they are not accessible. Hence, you have to resend the lost requests. The standby computer processes new requests sent in the meantime.
	▪ Plug-in the unplugged network cord.	When the network is up, the Service Containers must start sharing the requests.

Chapter 5
High availability best practices

This chapter provides best practices for achieving high availability.

- To eliminate the single point of failure while configuring a dependent Service Container, always point to a Service Group instead of a Service Container. For instance, while referring the Notification service in Business Process Management Service Container, point to the Notification Service Group. In the event of a Notification service container going down, the requests are routed to other containers associated with the Notification Service Group.
- Always use nodes with similar configurations as the speed of the slowest node in the HA cluster determines the efficiency of the clustering.
- Always validate HA setup before using it in the production environments.
- Always configure your HA setup to receive alerts. For instance, alerts can be raised in case of an LDAP server crash, a WCP monitor crash, and so on. This enables the administrators to take swift action and reduce the recovery time. For more information on configuring alerts, see References > Application Development > Working With Alert System > Generating Alerts topic in the product documentation.
- Avoid starting AppWorks Platform on cluster nodes simultaneously. Provide a gap of 30 seconds between each start.
- When members are hosted on VMWare nodes, set `com.cordys.cluster.timers.factor=2` property in the `wcp.properties` page.
- Ensure that all the nodes in a cluster carry the same system time.

Total business system and its components

1. Identify various components of the business system.
   a. AppWorks Platform components
   b. Non-AppWorks Platform components
2. Understand dependent back-end resources of each AppWorks Platform component for example, LDAP Service depends on CARS, BPM depends on database.
   a. Ensure that high availability is configured for all AppWorks Platform components and their dependent resources.

b. Ensure that HA is configured for all non-AppWorks Platform components. This list may include Network components, Database system(s), File Systems, Network load balancers, ERP systems, and CRM systems.

Production use cases

Thorough understanding of production use cases should indicate whether HA is supported for design time content (LDAP, XMLStore, Web folders) changes. A Yes or No answer to the question makes a big difference in HA infrastructure, administration cost, and setup complexity.

1. If No, then configuring high availability can be done using a simple infrastructure.
   - LDAP service can be made highly available through OpenLDAP multi-master setup.
   - XMLStore and Web folders HA can be achieved by copying file content across all nodes. In this case, the HA admin has to maintain consistent across all nodes during maintenance like upgrading AppWorks Platform and applying AppWorks Platform patches.
2. If Yes, then determine the following:
   - LDAP service HA on Windows and Linux can be achieved through OpenLDAP multi-master setup.
   - XMLStore and Web folders HA can be achieved by having a file system common for all cluster nodes (shared file system).

High availability testing

1. Define HA test scenarios for checking total business system HA.
2. Perform unit level testing as well as use case testing.
3. Document all use cases of the business system and conduct HA testing accordingly.
4. Automate testing because it is very important to test HA immediately after maintenance such as applying operating system patches and application patches.

High availability setup maintenance

1. Have predefined policies and strategies for maintenance activities.
2. Perform incremental (node-by-node) maintenance rather than modifying all cluster nodes at a time.
3. Always backup the total system before modifying it.
4. It should always be possible to roll back whatever changes happen to the business system.

Using existing clusters

1. Some AppWorks Platform components and dependent resources can be hosted in already existing clusters. Confirm if such clusters already exist.
2. Setup cost and maintenance overhead will increase with number of clusters you have in the total setup such as web server cluster, database cluster, and AppWorks Platform cluster. Try to reuse clusters when possible.

IP load balancer

1. Hardware-based IP load balancers are recommended for better performance and built-in HA features.
2. Software-based load balancers can be used for testing purpose.

Configuring alerts

1. Ensure that all HA resources have an alert mechanism.
2. Configure important alerts and define the corresponding course of actions.
3. AppWorks Platform provides alerts. Examples: when the LDAP Server crashes, when WCP Monitor crashes, when a service container crashes, when Database connecting lost.

Common file system

Ensure that the common file system you choose supports the following specifications:

- Abstracted file system
  - Simultaneous read/write operations with synchronization
  - Compliance with POSIX standards in Linux or UNIX environments
- Data backup and recovery strategies
- Scalable file system
- Hardware device that maintains data physically
- Hardware device with built-in high availability architecture
- High volume data storage

Use a reverse proxy in a cluster

AppWorks Platform services open some anonymous sockets internally for transmission of SOAP messages. These sockets are opened on any freely available port. However, in the case of a DMZ machine, the administrator makes only specified ports available. When as part of a cluster, a AppWorks Platform node is installed in the DMZ, there may be problems in the transmission of SOAP messages because the administrator may restrict the freely available ports.

Security implementation in all layers

You need to view security in a holistic approach, which requires securing all the layers with software and hardware, including the application platform. The types of tools you can use include the following:

- **Firewall** - A firewall is used to regulate the traffic between computer networks with different trust levels. Use an intermediate trust level in which the web server runs. This level is often called a DMZ (demilitarized zone).
- **Reverse proxy** - A reverse proxy server is installed in front of one or more web servers to regulate connections from and to these web servers. It adds another layer of security to the OpenText AppWorks Platform as it protects the AppWorks Platform Gateway.
- **Intrusion detection system (IDS)** - An IDS is used to detect unwanted manipulation of computer systems or applications. On each trust level, an IDS can be used to detect specific manipulation.

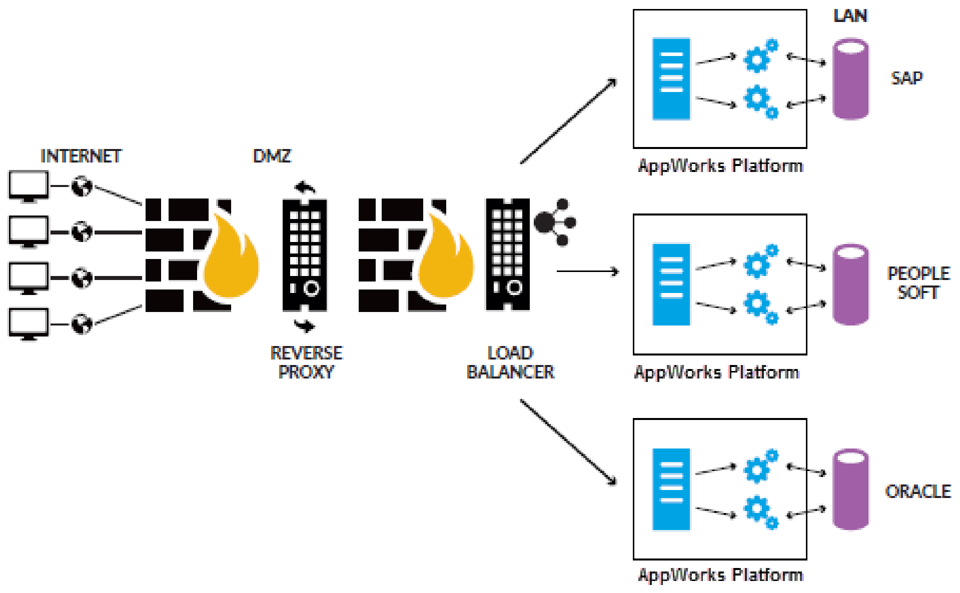
These tools can be combined in a DMZ setup:

- A firewall provides security at the network layer.
- Back-end services are connected to their physical local area network structure to prevent eavesdropping.
- IDS systems are installed to run at the network layer.
- Authentication is set up, and authorization ACLs are established.

Do not deploy the web server in the DMZ

In a typical three-tier setup of the web server, application server, and database server, the web server often is placed in the DMZ for security reasons.

For AppWorks Platform, the web server always is part of a complete installation and hence cannot be deployed separately. Nodes in a cluster must be able to communicate freely with each other. Therefore, there must not be a firewall between the different nodes. Additionally, do not place the web server node on a different network segment or DMZ.



Better approach - Reverse proxy

Use a reverse proxy in which case only a couple of ports are required to be opened. Depending on the requirements and usage of SSL offloading, for example, only the port(s) used by the AppWorks Platform web server need to be opened.

Benefits of reverse proxy server

There are several reasons for installing reverse proxy servers:

- **Encryption / SSL acceleration**: When secure websites are created, the SSL encryption is often not done by the web server, but by a reverse proxy that is equipped with SSL acceleration hardware. See Secure Sockets Layer. Furthermore, a host can provide a single "SSL proxy" to provide SSL encryption for an arbitrary number of hosts; removing the need for a separate SSL Server Certificate for each host, with the downside that all hosts behind the SSL proxy have to share a common DNS name or IP address for SSL connections. This problem can partly be overcome by using the SubjectAltName feature of X.509 certificates.
- **Load balancing:** The reverse proxy can distribute the load to several web servers, each web server serving its application area. In such a case, the reverse proxy may need to rewrite the URLs in each web page (translation from externally known URLs to the internal locations).
- **Serve/cache static content**: A reverse proxy can offload the web servers by caching static content like pictures and other static graphical content.
- **Compression**: The proxy server can optimize and compress the content to speed up the load time.

- **Spoon feeding**: Spoon feeding reduces resource usage caused by slow clients on the web servers by caching the content the web server sent and slowly "spoon feeding" it to the client. The benefit is dynamically generated pages.
- **Security**: The proxy server is an additional layer of defense and can protect against some operating system and web server specific attacks. However, it does not provide any protection to attacks against the web application or service itself, which is considered the larger threat.
- **Extranet publishing**: A reverse proxy server facing the Internet can be used to communicate to a firewalled server internal to an organization, providing extranet access to some functions while keeping the servers behind the firewalls. If used in this way, consider security measures to protect the rest of your infrastructure if this server is compromised, as its web application is exposed to attack from the Internet.

Chapter 6

Maintaining high availability

This chapter summarizes some of the tasks required to maintain a highly available setup, which can involve upgrade or rollback of AppWorks Platform or non-AppWorks Platform (operating system, system, and hardware). These tasks revolve around changes in the cluster setup to meet the scaled-up or scaled-down capabilities.

Remember to:

- Define the policies and strategies for maintenance activities in advance.
- Maintain the nodes in an incremental mode.
- Backup the complete node before modifying it.
- Roll back at any stage, regardless of the changes in the business system.

Bringing AppWorks Platform system online

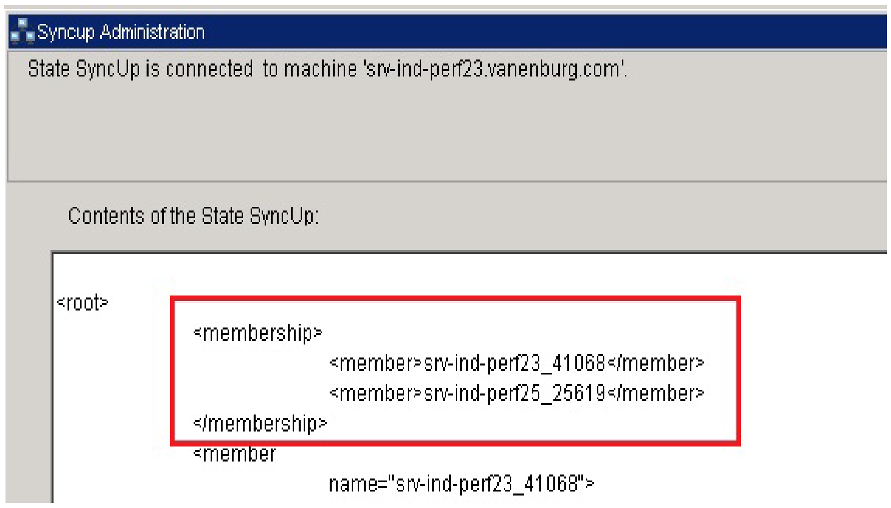
To bring the AppWorks Platform system online:

1. Start OpenText CARS.
2. Start OpenText AppWorks Platform (<instance name>).
3. Start TomEE.
   The new AppWorks Platform computer joins the cluster group.

To confirm if the new AppWorks Platform computer has successfully joined the cluster:

1. Click **Start** > **Programs** > ***<AppWorks Platform Instance>*** **Tools** > **Management Console**.
   The Management Console window opens.
2. In Linux, access the Management Console in the following way:
   - Launch Terminal.
   - `CD <AppWorks Platform_installdir>/bin`
   - Set your environment variables using `../wcpenv.sh`
   - Execute `./cmc.sh`
3. Click **State SyncUp** utility in the Administrative Utilities section.

4. The names of all the members in the cluster must appear within the membership tag of the Syncup Administration window as in the following image.



Syncup Administration

State SyncUp is connected to machine 'srv-ind-perf23.vanenburg.com'.

Contents of the State SyncUp:

```
<root>
        <membership>
                <member>srv-ind-perf23_41068</member>
                <member>srv-ind-perf25_25619</member>
        </membership>
        <member
                name="srv-ind-perf23_41068">
```

5. In System Resource Manager, the status of the monitor Service Container must be shown as Started.

Taking a AppWorks Platform system offline

To take a AppWorks Platform system offline:

1. Stop TomEE.
2. Stop OpenText AppWorks Platform (<instance name>).
3. Stop OpenText CARS.
4. Repeat the step of Bringing AppWorks Platform System Online as specified in the section above and verify that the stopped AppWorks Platform instance does not appear in the membership list.

Note: AppWorks Platform Distributed nodes are dependent on the Primary node. Therefore, the Distributed nodes must be stopped before stopping the Primary nodes.

Adding a new node to the AppWorks Platform cluster or scaling a AppWorks Platform cluster

To add a new node to the cluster or scale a cluster:

1. Install AppWorks Platform in Distributed mode.
   For detailed process, see Configuring AppWorks Platform for high availability.
2. Clone the required Service Containers and configure them to run on the newly added node.

3. Install OpenText CARS, if required, and configure replication with the existing OpenText CARS.
4. To make Distributed node as a Primary node, setup OpenText CARS on this node, clone all the Service Containers, and configure them to run on this node.

Removing a node from AppWorks Platform cluster

Uninstalling AppWorks Platform from the required node removes that node from the existing AppWorks Platform cluster.

When removing a node that is the primary node in the cluster:

1. Set another node in the cluster as the primary node:
   a. Edit the `installvariables.properties` file in *`<AppWorks Platform_installdir>`*`/_uninst`
   b. Change **IS_DISTRIBUTED_INSTALLATION = true** to **IS_DISTRIBUTED_INSTALLATION = false**
2. Change the existing primary node to a distributed node:
   a. Edit the `installvariables.properties` file in *`<AppWorks Platform_installdir>`*`/_uninst`
   b. Change **IS_DISTRIBUTED_INSTALLATION = false** to **IS_DISTRIBUTED_INSTALLATION = true**

Before uninstalling a node from a AppWorks Platform cluster:

1. Stop and delete all Collaborative Workspace service containers on the node to prevent the Collaborative Workspace from unregistering CWS document types.
2. For more information, see Uninstalling in a Primary-Distributed Setup in GUI Mode section in the *AppWorks Platform Installation Guide for Windows*.

Note: Uninstaller automatically stops all the active service containers and OpenText AppWorks Platform (<instance name>) in a graceful manner.

Applying hotfixes or upgrade

Always apply hot fixes or upgrade in an incremental manner, starting with the master node.

To apply hotfixes or upgrade:

1. Take a backup of the OpenText CARS content.
2. Change the LDAP Server details of all the secondary LDAP Service Containers to Primary OpenText CARS.
3. On all the nodes, except the master node, stop all the Collaborative Workspace service containers, if available, and set the startup type to Manual.

Note: Stop all AppWorks Platform nodes one after the other, except one master node. Stop all the distributed nodes and then the master nodes. However, ensure that one master node is up and running all the time.

4. Apply the fix on Master1 and then on Master2, and so on.

   Note: Do not start OpenText CARS on Master2 and other nodes until the entire process is updated.

5. Apply the fix or upgrade in the sequence of Secondary setup1, Secondary setup2, and so on.
6. Start all the secondary OpenText CARS instances one after the other and wait until the replication is completed.
7. On all the nodes, reset the startup type of the Collaborative Workspace service containers to their original value, **Automatic**.

Managing applications on cluster nodes

You can deploy, undeploy, upgrade, rollback, and revert the applications on the selected nodes from a single node.

Note: Ensure that you start the OpenText AppWorks Platform (<instance name>) on all the cluster nodes before you start this task.

To deploy the applications:

1. From the start page, open the **Application Deployer**.
2. Right-click the required applications in the **Application List** and select **Deploy**.

   Note: Upload the required application through Deployment Overview if the application is not available in the list.

3. Select the nodes on which you wish to deploy the application.

The applications are deployed on the selected nodes in the cluster. Similarly, you can undeploy, upgrade, rollback, or revert the applications.

Starting and stopping AppWorks Platform components

This section discusses procedures to start and stop various AppWorks Platform components.

OpenText CARS

Purpose	Operating System	Procedure
Start	Windows	Provide the following in the command prompt: `net start OpenLDAP-slapd<instance>` Or Go to **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services**. Within the services, locate the OpenText CARS instance, right-click and select **START**.
	Linux	Provide the following in the terminal prompt: `service OpenText CARS-slapd<instance> start`
Stop	Windows	Provide the following in the command prompt: `net stop OpenLDAP- slapd<instance>` Or Go to **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services**. Within the services, locate the OpenText CARS instance, right-click it and select **STOP**.
	Linux	Provide the following in the terminal prompt: `service OpenText CARS-slapd<instance> stop`

OpenText AppWorks Platform (<instance name>)

Purpose	Operating System	Procedure

Start	Windows	▪ Set the following environment variables:
		`SET AppWorks Platform_install_dir=`*`<AppWorks Platform_installdir>`*
		`SET CLASSPATH=%AppWorks Platform_install_dir%\cordyscp.jar;%CLASSPATH%`
		`SET PATH=%AppWorks Platform_install_dir%\lib;%PATH%`
		▪ Provide the following in the command prompt:
		`java com.eibus.applicationconnector.monitor.Launcher`
		Or
		Go to **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services** . Within services, locate the OpenText AppWorks Platform (<instance name>), right-click and select

Purpose	**Operating System**	**Procedure**
		START.

	Linux	Set the following environment variables:
		`export AppWorks Platform_install_dir=<AppWorks Platform_installdir>`
		`export CLASSPATH= $AppWorks Platform_install_dir/cordyscp.jar:$CLASSPATH`
		`export LD_LIBRARY_PATH= $AppWorks Platform_install_dir/lib:$LD_LIBRARY_PATH`
		Provide the following command in the terminal:
		`java com.eibus.applicationconnector.monitor.Launcher`
		Or
		`service wcpd start`

Stop	Windows	If the OpenText AppWorks Platform (<instance name>) is started from the command prompt, press **ENTER** in the command prompt window. Or Go to **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services**. Within services, locate the Monitor instance, right-click it and select **STOP**.
	Linux	Press **ENTER** in the terminal. Or Provide the following command in the terminal: `service wcpd stop`

Web server

Purpose	**Operating System**	**Procedure**
Start	Windows	Go to **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services**. Within services, locate the Web server, right-click it, and select **START**.
	Linux	Provide the following in the terminal: `<TomEE_installdir>/bin/./tomee start`
Stop	Windows	Go to **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Services**. Within services, locate the Web server, right-click it, and select **STOP**.

Purpose	**Operating System**	**Procedure**
	Linux	Provide the following in the terminal: `<TomEE_installdir>/bin/./tomee stop`

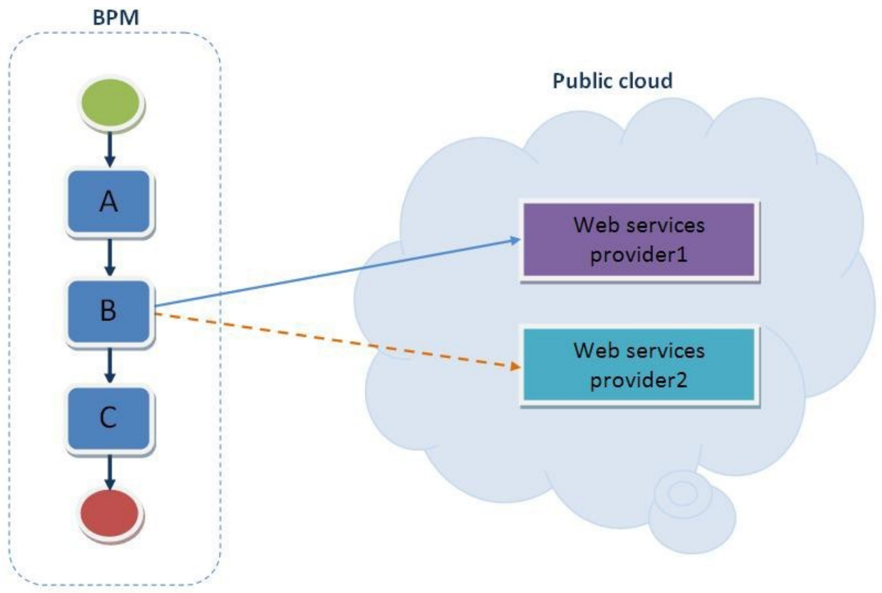
Chapter 7

Achieving high availability for consuming web services in public cloud

This chapter discusses various possibilities for achieving high availability for consuming web services in a public cloud through AppWorks Platform BPMs.

It is possible to achieve high availability for consuming web services available in the public cloud. One can adopt the one of the discussed solutions by weighing pros-and-cons. Achieving high availability for BPM service container is already known. However, it could be challenging if one or more activities in a BPM involve communicating with (external) web services in the public cloud, and more than one provider is responsible for the required web services.

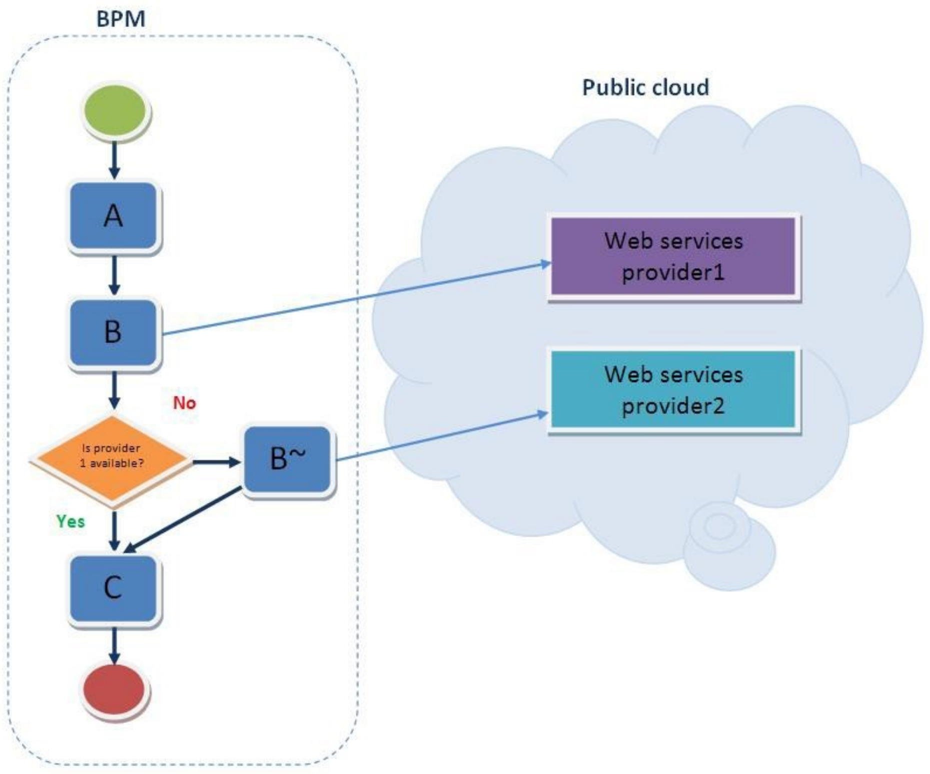
If a given web service provider is not available, then BPM should be intelligent enough to contact alternative web service providers, so that execution of BPM can continue without problems. The following image depicts this requirement where Activity B depends on Web services Provider 1, and if Web services Provider 1 is down then Activity B should contact Web service Provider 2.



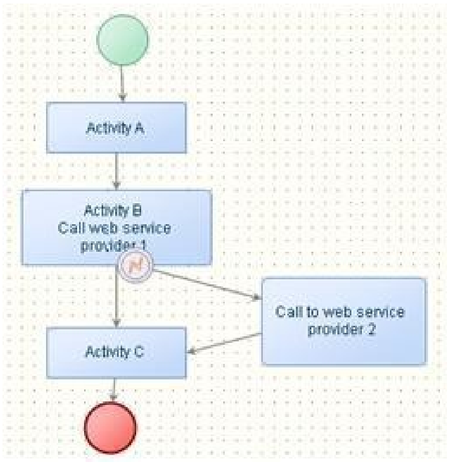
The following sections explain various approaches to achieve this.

Solution 1: Exception handling in BPM

You can leverage the Exception handling feature in BPM to meet the requirement. Exception handling logic must be modeled in such a way that BPM will contact an alternative web service provider if a given web service provider is not available. Review the following diagram for more details.



The equivalent AppWorks Platform BPM diagram is as follows.


Activity A
Activity B Call web service provider 1
Call to web service provider 2
Activity C

Advantages

- No programming is required, so it is very easy to achieve.

Disadvantages

- If Provider 1 is continuously down for a prolonged period, then every time Provider 1 is contacted first (even though it is not available) and then Provider 2 is contacted. This also generates many error messages.

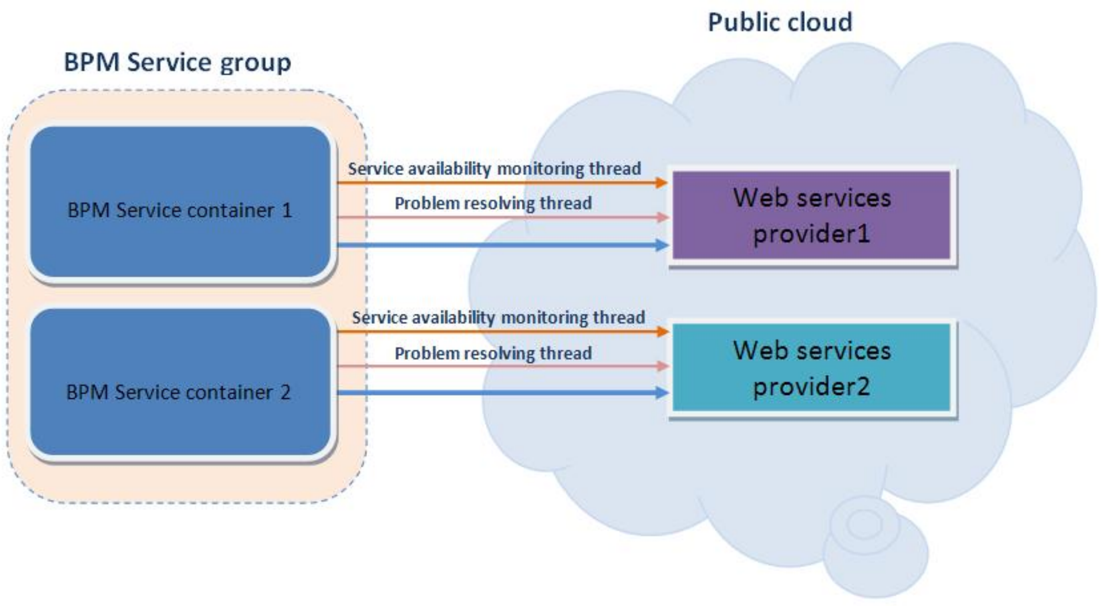
Solution 2: Using AppWorks Platform problem registry framework

The concept is to divide BPM service containers for each web service provider and using problem registry framework extend BPM service container logic to achieve the following:

- Continuously monitor the availability of web service provider and exclude the service container from receiving further requests if the service provider is functional.
- Enable service container for receiving requests when web service provider becomes available.

Note: The problem registry provides by default a monitoring thread and resolving thread. You must implement an interface to leverage this.

The following diagram explains the overall concept.



Advantages

- Disadvantage mentioned in the Solution 1 can be overcome here.
- Service container automatically becomes non-functional if the corresponding web service provider is down.

Disadvantages

- Implementation of logic involves developing Java code using problem registry.
- Traffic to the web service provider increases due to monitoring and problem-resolving threads. This leads to degradation of performance. It would helpful if the web service provider supports status-health-check (similar to a ping) functionality so that service availability can be monitored efficiently.
- Both web service providers are always contacted. Avoid this by configuring BPM Service group in failover mode.

Chapter 8

Configuring cluster nodes and service containers

This chapter provides guidelines for creating cluster nodes and service containers. The guidelines assist in making appropriate choices and avoiding problems with cluster configuration.

Perhaps you struggle with questions such as:

- Do I need all platform service containers on all cluster nodes?
- Do I need all application service containers on all cluster nodes?
- Do all cluster nodes require LDAP Servers (CARS)?
- What is the recommended number of high availability nodes for a cluster?
- What is the recommended number of load balancing nodes for a cluster?
- When should I clone service containers? How many should I clone?

Suggested guidelines for creating cluster nodes and service containers

OpenText suggests the following guidelines for creating cluster nodes and service containers.

AppWorks Platform cluster

AnAppWorks Platform cluster (PSP cluster) should satisfy two objectives:

- High availability
- Load balancing

A PSP cluster should comprise some high availability nodes and some load balancing nodes.

High availability nodes

High availability (HA) nodes ensure business continuity for PSP clusters:

- All high availability nodes should have all components of PSP clusters (platform service containers, platform Webgateway, and LDAP server). **Note**: If the business does not require some of the platform services, such as BAM and MDM, do not set them as highly available.
- All high availability nodes require application service containers that are responsible for handling application services.
- Set the startup type to manual for service containers that are not required. For example, some businesses may not require BAM and MDM services.

Note: These guidelines do not apply to application service backend systems such as SAP, databases, and CRM systems.

Load balancing nodes

Load balancing (LB) nodes satisfy capacity requirements (production peak load requirements) in addition to capacity available on the HA nodes.

- In major cases, all LB nodes will have application services.
- Depending upon business use cases, some of the platform services such as XForms, Repository, and Notification services may require additional capacity.

Business criticality

High availability and load balancing decisions for Application Services should be driven by business criticality. For example, make all end user facing services highly available. Administrative services like archiving may not be business critical and may not require HA configuration.

Capacity requirements

Calculate the capacity requirements of a service as follows.

Consider a case of a five-node cluster with 2 HA nodes and 3 LB nodes. Assume that a service (banking service) expects to receive 150 concurrent requests.

- By default, each service container (including application service containers) has 10 worker threads.
- Then increase worker threads from 10 to 50 and create 3 additional services by cloning: two on 2 HA nodes and two on 2 LB nodes.
- For the service, there are 4 service containers each with 50 worker threads. Total parallel processing capacity is 200. If one of the nodes is down any time then you still have 150 parallel processing capacity.

Number of service containers

OpenText recommends a maximum total number of service containers of 200 (including application services) in a PSP cluster. The current theoretical limit is 280. Often, there are more configured service containers than are required. OpenText recommends that you configure service containers based on your capacity requirements.

Example: A 5-node cluster, with 3 HA nodes and 2 LB nodes is simultaneously configured to handle failure of two nodes at a time. However, this is a rare situation. Therefore, for the majority of cases, 2 HA nodes are sufficient.

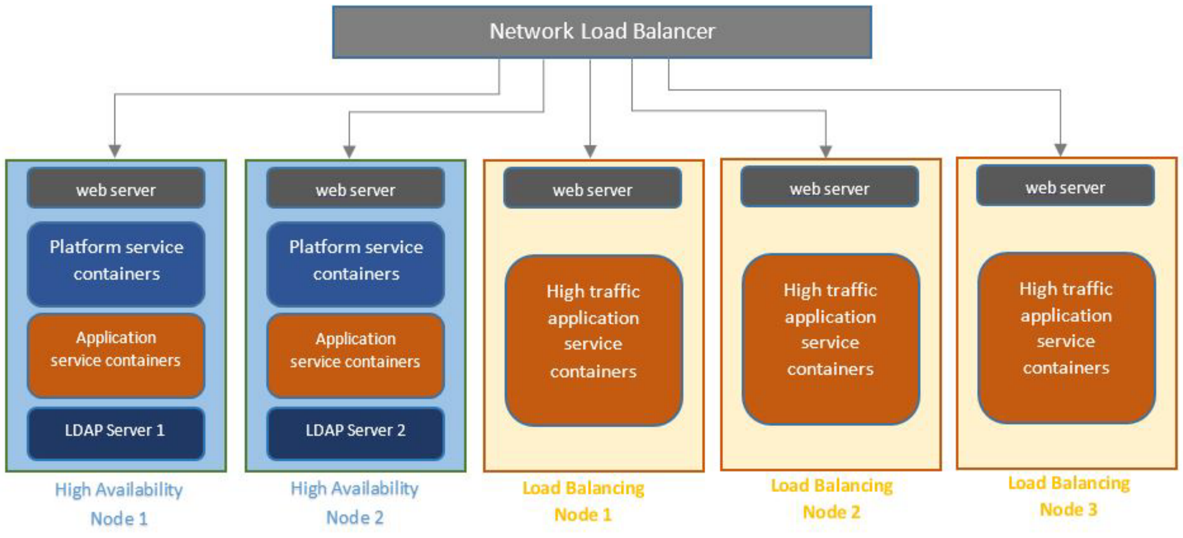
Note: If you expect more than one node to go down simultaneously, improve the infrastructure layer outside the PSP cluster configuration. For example, if two nodes are connected to a single switch, then switch failure can lead to simultaneous failure of both nodes.

Making all nodes as HA nodes multiplies the number of service containers and increases resource consumption and maintenance overhead. This in turn creates stress on PSP cluster communication, which can lead to cluster instability.

Typical PSP cluster

The following diagram shows a typical 5 node AppWorks Platform cluster. Out of 5 nodes, the first two represent high availability (HA) nodes; these nodes contain all AppWorks Platform components, to include AppWorks Platform Web gateway, all platform service containers, LDAP server (CARS), and all application service containers. The HA nodes ensure business continuity in the event of failure of AppWorks Platform components and application service containers. If one of the HA nodes is down (for example a server crash), business continuity is assured.

The last 3 nodes represent load-balancing nodes. The 3 nodes are configured for handling additional capacity required for the business application. These load-balancing nodes contain only high traffic application service containers. The number of instances of application service containers should be configured based on capacity requirements. Avoid putting other application service containers with low traffic on these nodes. AppWorks Platform service containers are not required on load-balancing nodes. Note: The CAP service container is the exception; it is required for processing AppWorks Platform deployment related services. An example is deploying a package or upgrading a package. All load-balancing nodes should have one instance of a CAP service container.


Network Load Balancer
web server
Platform service containers
Application service containers
LDAP Server 1
High Availability Node 1
web server
Platform service containers
Application service containers
LDAP Server 2
High Availability Node 2
web server
High traffic application service containers
Load Balancing Node 1
web server
High traffic application service containers
Load Balancing Node 2
web server
High traffic application service containers
Load Balancing Node 3

Chapter 9
Concepts and architectural insight

From high availability point of view, AppWorks Platform has no single point of failure in a clustered environment. Because AppWorks Platform uses SOA everywhere, services can be hosted multiple machines in the cluster. Failures can range from application crash to system crash to network outage to power failures, and so on. High availability refers to a system or component that is continuously operational for a desirably long period. Availability refers to the ability of the user community to access the system. If a user cannot access the system, it is unavailable. High availability clusters are computer clusters that are implemented primarily for improving the availability of services. They operate by having redundant computers or nodes, which are used to provide service when system components fail. When an application crashes on a server, high available clustering remedies this situation by detecting failures and immediately initiating the service availability on a different system without requiring administrative intervention. This process is known as failover.

Architectural insight

Because no session maintenance is required in AppWorks Platform, clustering and high availability do not introduce any overhead. This helps to achieve near linear scalability. The setup and administration are transparent and straightforward.

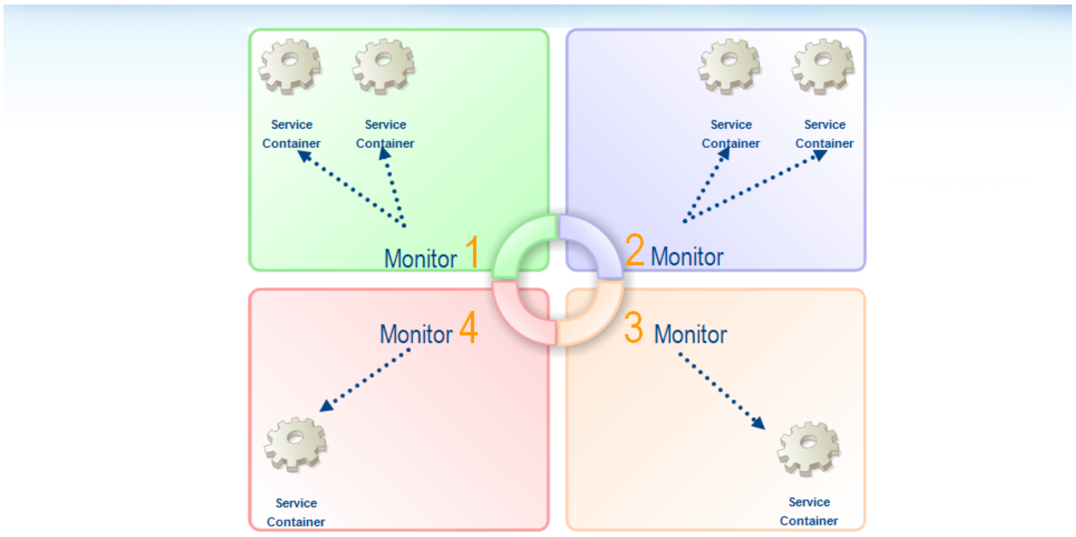
AppWorks Platform implements the following concepts to achieve this:

1. **Self-healing**
   - Detect problems and initiate corrective actions without disrupting services.
   - Corrective actions can include changing its state to 'unavailable' to enable other services to take over.
   - Increase availability by discovering and fixing the problems.
2. **Fail fast**
   - A non-intuitive technique for 'fast failover' due to fast error detection. n When a problem occurs, fail the service immediately and visibly.
   - Makes problems easier to find.

To achieve fail-fast and faster failure detection, AppWorks Platform has built a technology called State SyncUp (SSU).

AppWorks Platform State SyncUp maintains and keeps all members up-to-date about the state of each service container running in the cluster. The state information is known before firing the request. This ensures that, on SOA Grid, the message is always routed to the healthy service containers.

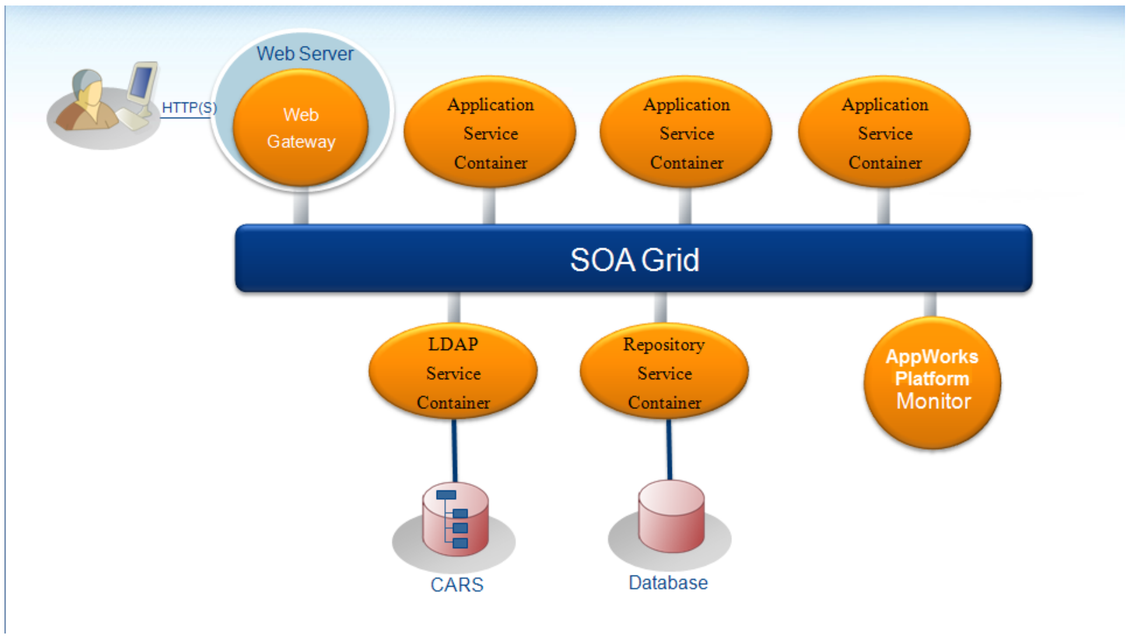
All AppWorks Platform monitors form a network to be part of State SyncUp. They keep exchanging the state information of service containers running on their respective nodes.



The routing mechanism can be configured for load balancing and failover. For load balancing, all service containers are treated as active. Messages are distributed across different service containers. For failover, only one service container is treated as active; the others are in passive standby. When the active service container crashes, one of the passives becomes active and starts accepting requests.

AppWorks Platform on a single node

All the components can run on a single node or spread across a cluster. The following diagram provides an idea when all of the components are running on a single node.

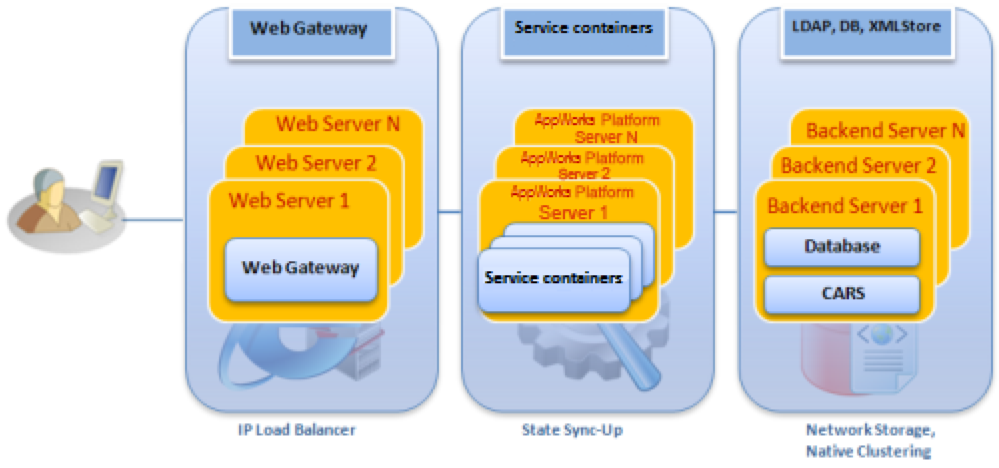

Web Server
HTTP(S)
Web Gateway
Application Service Container
Application Service Container
Application Service Container
SOA Grid
LDAP Service Container
Repository Service Container
AppWorks Platform Monitor
CARS
Database

All service containers, Web Gateway and AppWorks Platform Monitor running on a single node.

The repositories (CARS, Database) and any back-end systems may be running on the same node or different nodes in the same LAN.

High-level deployment architecture

Running everything on a single node does not address failures. The following diagram shows a logical view of a clustered environment.



Different logical clusters for web servers, AppWorks Platform servers, and back-end systems might exist.

Firewalls may exist between the following clusters:

- **Web server cluster**

  Two (or more) machines dedicated to the web server. These servers are configured in load balancing mode to achieve high availability and scalability. Each web server has AppWorks Platform Web Gateway configured. Single point of failures of web servers is tackled in this cluster.

- **AppWorks Platform server cluster**

  Two (or more) computers are running AppWorks Platform. Each AppWorks Platform server can host multiple service containers. The HA solution to be adopted for individual service containers depends on the backend server capabilities of the systems to which they are connected. Example: LDAP service container to run in Fail Over mode while the Repository service container can be configured to run in load balancing mode.

  In the case of a failure of one of the service containers or servers, the heartbeat is lost, and the State SyncUp switches to standby services.

  Failures of AppWorks Platform service containers are tackled in this cluster.

- **Backend server cluster**

  Two (or more) machines dedicated to the database engine/LDAP engine/network attached storage or any other back-end systems.

  Failures of storage, LDAP, databases and back-end systems are tackled in this cluster.

Hence, the clustering of:

- Web servers is achieved through IP load balancers or similar technologies.
- AppWorks Platform servers is achieved through State SyncUp.
- Back-end systems may use native clustering technologies such as SAN, high available database.

Clustering AppWorks Platform servers include all service containers. Redundant service containers are configured on multiple nodes to server failover and load balancing. All such redundant services must be functionally equal.

Minimum requirements for high availability

To setup clustering or high availability, AppWorks Platform needs to be installed on multiple computers. The first installation is called primary, and the rest are called distributed installations. On distributed installations, AppWorks Platform Monitor must be running to make sure that the node joins the cluster.

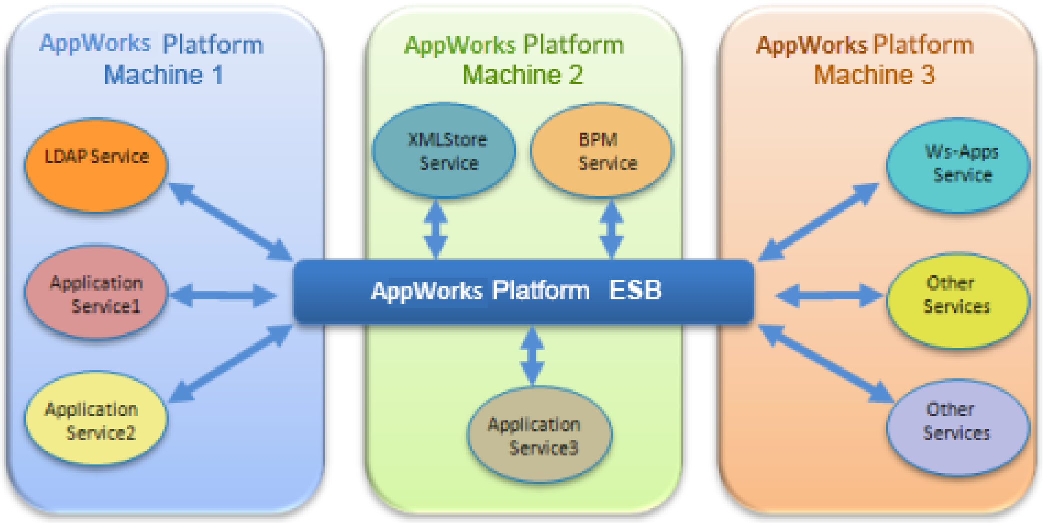
Service containers, both platform and application specific, can be configured on any of the nodes.

AppWorks Platform state SyncUp

The State SyncUp (SSU) updates all the AppWorks Platform cluster nodes about the state of each Service Container running on the cluster. The message is always routed to the healthy Service Container on the SOA Grid, because of this pre-informed state before firing a request. All the OpenText AppWorks Platform (<instance name>)s form a network to be part of State SyncUp. They keep exchanging the state information of service containers running on their respective nodes.

AppWorks Platform SOA grid

AppWorks Platform is built using Service Oriented Architecture (SOA) and Enterprise Service Bus topology (ESB). It is a collection of distributed service providers (Service Containers) connected to a robust ESB. A service provider can be available on one or more nodes in a distributed fashion.



This enables service providers to form a cluster and separate when it is not required.

Cluster configuration

The cluster configurations that are possible with AppWorks Platform are:

- Fail-over cluster
- Load-balancing cluster

Failover cluster

The most common solution for the failure of a system performing critical business operations is to have another system waiting to take over the failed system workload, and continue the business operations. When a server with a particular application crashes, failover clustering remedies this situation without requiring administrative intervention by:

- Detecting failures
- Immediately initiating the service availability on another node available within the same cluster

Based on the preference or priority set for the services within the cluster, the failover activity is performed. One of the nodes in this cluster has services with high priority or first preference. Such a node is called an Active node. All other nodes are called Passive or Standby nodes.

This is also known as Active-Passive cluster configuration. A disadvantage with this configuration is that most of the time, the Standby nodes remain inactive. The automatic detection of failure in AppWorks Platform components and initiation of failover process is powered by AppWorks Platform State SyncUp (SSU).

Load-balancing cluster

This is an advanced type of cluster configuration, in which all the nodes perform the tasks with equal preferences and share the load among them by simultaneously processing requests. If a node fails, the services are still available to end-users because another node in the same cluster is already available and continues to process requests. This is called Active-Active configuration. This capability of configuring a load balancing cluster is powered by the scalable architecture of the underlying AppWorks Platform SOA Grid.

AppWorks Platform service containers load balancing and failover support matrix

While designing high availability of business systems, be aware of failover and load balancing support of various AppWorks Platform Service Containers.

Service Container	**Failover**	**Load balancing**	**Dependent resources**	**Remarks**
Audit	✓	✓	Database system	
BAM	✓	✓	Database system	
BPM	✓	✓	Database system	

Service Container	Failover	Load balancing	Dependent resources	Remarks
CWS	✓	✓	Database system	
COBOC	✓	✓	Database system	
Data Transformation	✓	✓	Database system	
Email	✓	✓	Email server	
Event Service	✓	✓		
Java Call	✓	✓		
LDAP	✓	✓	LDAP Server (OpenText CARS)	Failover is recommended
Notification	✓	✓	Database system	
Repository	✓	✓	Database system	
Rule Management	✓	✓	Database system	
Scheduling	✓	✓	Database system	
Security Administration	✓	✓		
Single Sign-On	✓	✓		Uses custom routing algorithm
UDDI	✓	✓		
WS-AppServer	✓	✓	Database system	
XForms	✓	✓		
XMLStore	✓	✓	Database system	
FTP	✓	✓	FTP server	

Routing optimization for requests inside an operating system process or TomEE

With Service Containers running in TomEE, routing logic optimization can keep requests within TomEE. Primarily, the SOA grid maintains the SOAP message and cascading requests within the same operating system Process or TomEE (also an operating system Process). When the Web Gateway or Service Containers that run in TomEE receive a SOAP message, the routing logic identifies the Service Groups and Service Containers within the same TomEE instance to route the SOAP message. If the routing logic cannot identify a Service Group and Service Containers within the same TomEE instance, the request is sent to the matching Service Groups and Service Containers on another Platform node of the cluster.

There is an exception to this rule. If a matching Service Group and Service Container is identified in the same TomEE instance that can handle the SOAP message, but the Service Container is stopped, an error is displayed, and the SOAP message is not routed to a running Service Container on another Platform node.

No single point of failure

In AppWorks Platform, multiple Service Containers operate under a single Service Group, and these are spread across multiple nodes. This clearly reduces the dependency on a specific Service Container residing on a given node to process a particular request. Also, builds a redundancy to eliminate the single point of failure. In other words, a stand-by mechanism is implemented immediately when one or more service containers fail.

No state or session maintenance

AppWorks Platform evades the need of maintaining the state information of any service, consequently reducing the consumption of IT resources. AppWorks Platform adapts the principle of stateful objects on state-less connections, which in turn is based on the simple stateless HTTP protocol. This choice architecture enables various services to run with very low memory and CPU footprint. In addition, it enables and simplifies the scale-out possibilities for businesses.

Because of this statelessness, an aborted transaction need not bank upon a single node and can fall back on any available node. This immediate switch enables a continued high availability of resources.

Fail-fast and self-healing

When a Service Container fails to operate, depending on the cluster configuration, the cloned container takes over and avoids any gap in the operation. Meanwhile, the failed Service Container withdraws its activities, proactively registers its status in the Problem

Registry and initiates a corrective action, which can include changing its state to unavailable to enable other services to take over.

The Problem Registry is an SSU based framework, which stores the registered problems and retrieves them programmatically. In this framework, the component that registers a problem is the one that resolves it. This is based on the concept of self-healing. Self-healing is also known as autonomous computing.

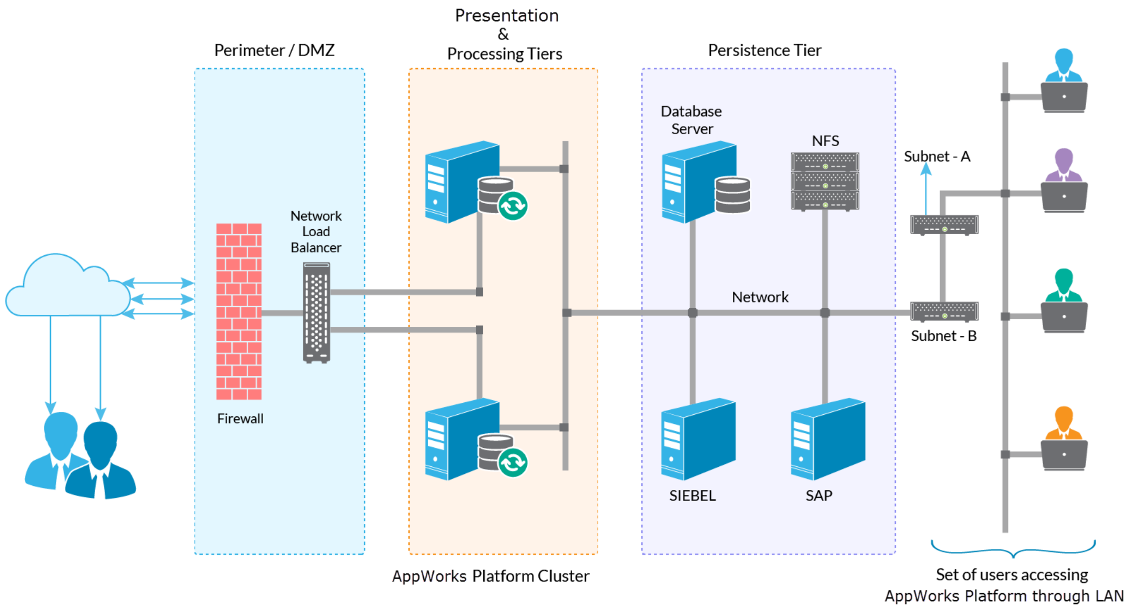
AppWorks Platform SSU maintains and keeps all the members up-to-date about the state of each Service Container running in the cluster. The state information is known before firing the request. This ensures that, on SOA Grid, the message is always routed to the healthy service containers.

This implies, it detects the problems and initiates corrective actions automatically without disrupting services. Thus, enabling increased availability by discovering and fixing the problems.

Chapter 10

High availability deployment types and best practices

An enterprise software system is highly available when it is designed to be resilient to various disruptions that can take place in the system and their dependent resources and infrastructure. These can include system crashes, application crashes, network problems, and so on. The HA ensures continuous availability of the application services to the end users and thereby ensures uninterrupted business.



Configuring HA includes three major steps:

1. Configuring CARS HA
2. Configuring AppWorks Platform SOA Grid HA
3. Configuring AppWorks Platform Web Gateway HA

Significant functions and features

1. **Failover cluster**

   When a server application crashes, the failover clustering option remedies this situation without administrative intervention. The failover activity is performed based on the preference or the priority set for the services within the cluster. One of the nodes in this cluster will have services with high priority or first preference. Such a node is called the *Active* node. All the other nodes are called *passive* or *standby* nodes. This is also known as *Active-Passive* cluster configuration.
2. **Load balancing cluster**

   All nodes in the cluster will perform the tasks with equal preferences and share the load among them by simultaneously processing the requests. If a node fails, the services are still available to end users as another node in the same cluster is already available and continues to process the requests. This is called *Active-Active* configuration.

AppWorks Platform high availability deployment types

1. HA Deployment Type 1: Active-Active cluster without shared file system. See Deployment 1.
2. HA Deployment Type 2: Active-Active cluster with shared file system. See Deployment 2.

Best practices

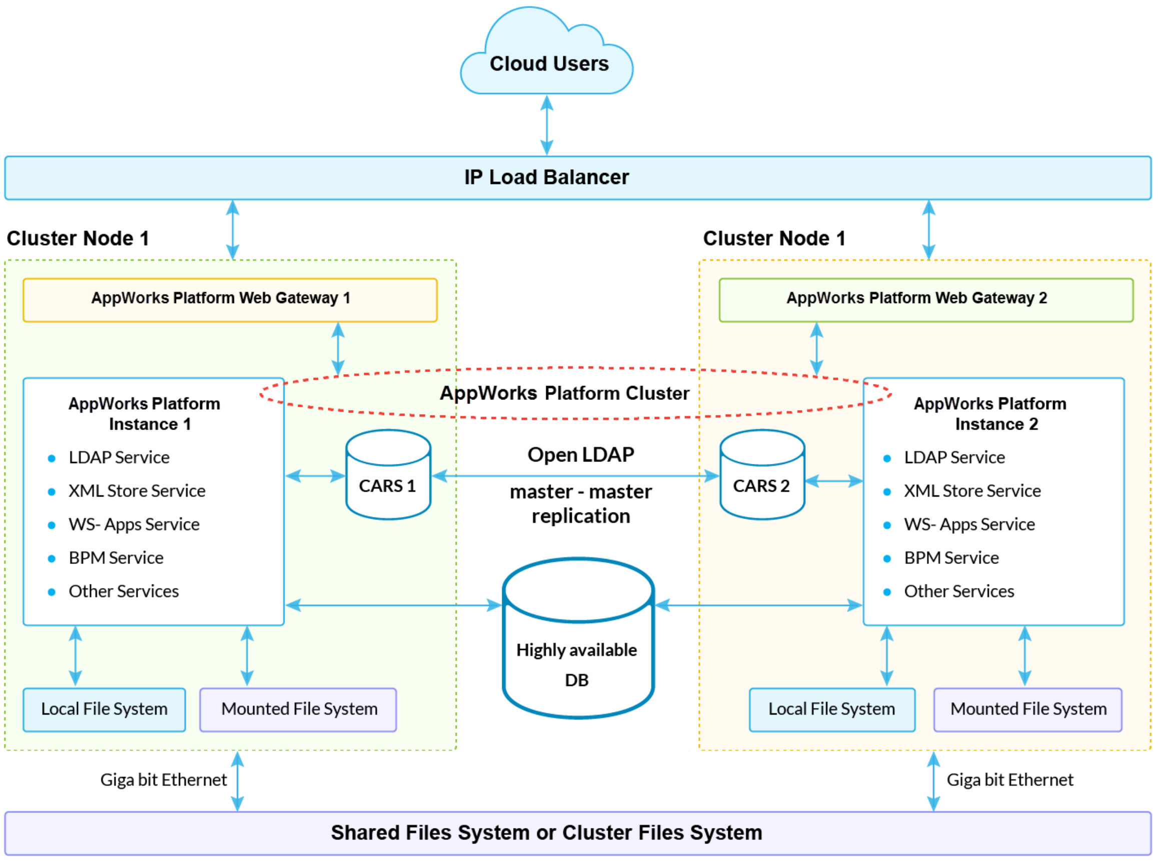
1. Understand HA goals thoroughly and select your infrastructure accordingly.
2. Note: The *Active-Passive* cluster with the described deployment types can be achieved by choosing failover routing at all AppWorks Platform Service Groups and AppWorks Platform Web Gateway.
3. Validate the HA environment with appropriate test cases.

Deployment scenarios

This section illustrates how AppWorks Platform components are made Highly Available in different scenarios with varying infrastructure support.

Deployment 1

When the application needs multiple users to access the file system simultaneously, the file system is shared across the nodes through common or shared file system, such as NFS, GFS. The following deployment scenario shows the shared file system mounted on both the nodes.



Key aspects of this deployment:

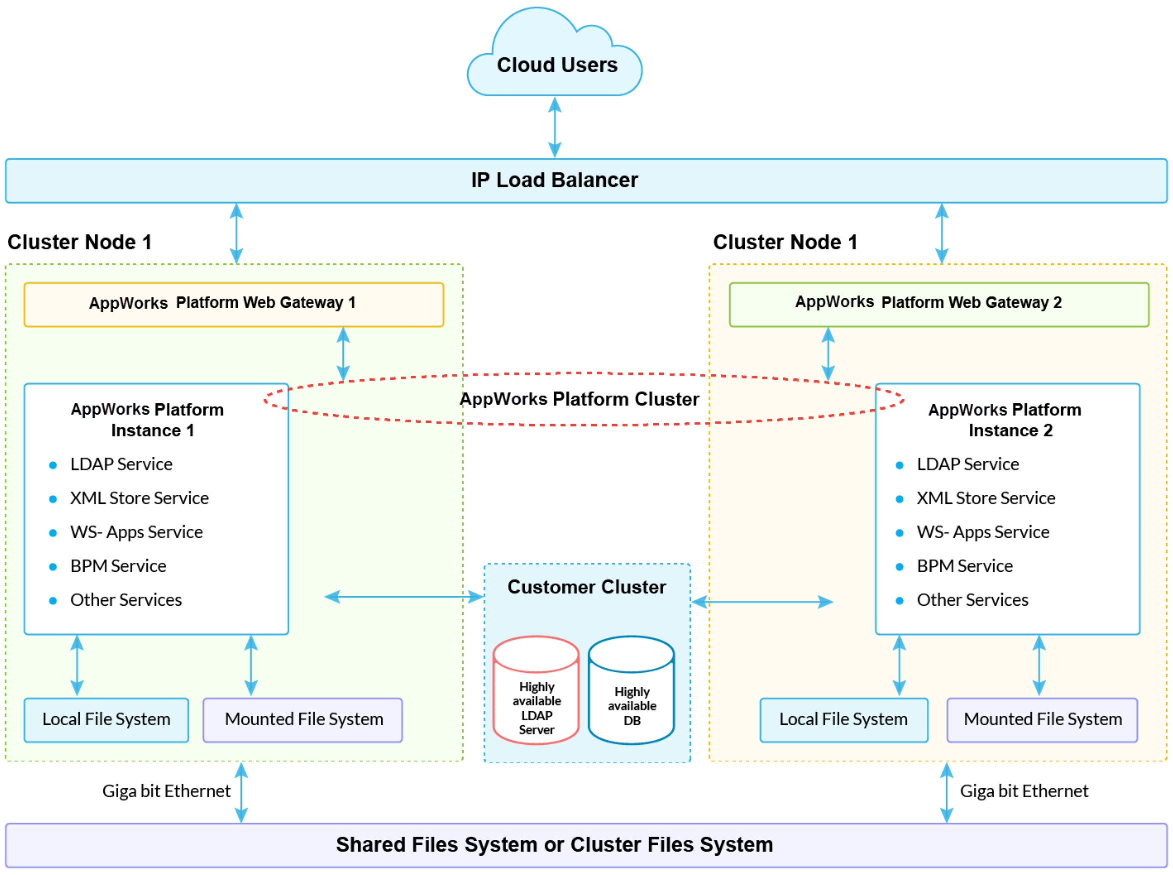
- AppWorks Platform is installed on both the active nodes and the Service Containers are synchronized through the SSU framework.
- OpenText CARS is configured in Multi-Master mode.
- Web servers are load balanced through an IP load balancer.
- File system is shared across all nodes.
  - If the business logic involves uploading and downloading files, then you must ensure that AppWorks Platform `UploadWritePath` and `DownloadReadPath` are pointing to appropriate shared file system. Achieve this by setting the following properties in the `wcp.properties` files:

    ```
    com.eibus.web.tools.upload.UploadWritePath=<shared file system folder>
    com.eibus.web.tools.download.DownloadReadPath=<shared file system folder>
    ```
  - Restart the Web service after setting the properties.
  - Configure the setting on all the nodes.
  - The shared file system must be POSIX compliant.
  - Ensure the uninterrupted availability of the shared file system to the application.

Note: Ensure that AppWorks Platform user has the appropriate rights while accessing the file system.

Deployment 2

You can configure OpenText CARS to run within an existing cluster. In this deployment scenario, OpenText CARS need not be replicated. Only one instance of OpenText CARS is installed and added to the existing cluster. This reduces the overhead of maintaining multiple OpenText CARS instances.



Key aspects of this deployment:

- AppWorks Platform is installed on both the active nodes and the Service Containers are synchronized through SSU framework.
- Web servers are load balanced through an IP load balancer.
- OpenText CARS is published in an external Cluster.
- The file system is shared across all the nodes. Address all the points mentioned in the Deployment 1 section related to the shared file system.

Chapter 11

Troubleshooting

This chapter describes the common problems that you may encounter while configuring or maintaining a HA setup.

Problem Definition	Issues while bringing a AppWorks Platform system online
Analysis	Bringing a AppWorks Platform system online blocks the existing AppWorks Platform cluster communication for a while. This is applicable when the AppWorks Platform system has multiple Service Containers and a primary LDAP Service to be started. The default batch start size for Service Containers is 100000. When the OpenText AppWorks Platform (<instance name>) tries to start all of them simultaneously, the LDAP Service Container is occupied in providing bootstrap information to all the Service Containers being started. During the same phase, when the client tries to retrieve connection point details of the recently added Service Containers, it is delayed as the LDAP is busy with numerous Service Containers, and hence, the client is not able to send requests on time.
Solution	Set the Batch Size to one (1) in the corresponding Monitor configuration. This enables the Monitor to start the Service Containers one after the other, and consequently the client is not blocked. **Note:** This is applicable in the following situations: ▪ AppWorks Platform running on slow nodes ▪ AppWorks Platform running on VMWare instances ▪ AppWorks Platform instances with many Service Containers, for example 20, 30

Problem Definition	A SOAP fault states "Wrong credentials entered" is displayed when entering correct credentials.
Analysis	This may occur when the nodes (computers) in a cluster are not synchronized to display the same time and time zone.
Solution	Set the same time for all the nodes in a cluster.

Problem Definition	Content is not replicated properly after replicating the OpenText CARS instances.
Solution	**Check the following:** 1. Set the same time for all nodes in a cluster. 2. If the following LDAP errors are displayed while performing an LDAP insert or update, add mirrormode in the `slapd.conf` of the LDAP servers. Unwilling To Perform (53) Unwilling To Perform- Server Message: shadow context; no update referral. 3. Validate the OpenText CARS setup using a third-party LDAP Explorer tool. **If none of the previous steps rectify the problem, do the following:** 1. **Verify the Configuration File** - Verify the following elements in the `slapd.conf` • `serverI` - This must be an integer with a minimum value = 0 and maximum value = 999. The serverID must be unique in both the machines. If it is 1 in master1, it must be any integer other than 1 in master2. • `#access to * by * auth` - Ensure that this command is commented. • `overlay syncprov` - Add this directive to the file. • `provider=ldaps://<master2 server>:<master2 port>` - In this command, the Master2 server name is the same computer name as Issued to: value of the Master2 certificate and likewise for Master1 OpenText CARS. • `tls_cacert, tls_cert` - This must contain the absolute path to the other master certificate file that is copied to the local machine. For example: In Master1, `tls_cacert="<Installation Directory of master1 OpenText CARS>/certificates/<master2 certificate>.cer"  tls_cert="<Installation Directory of master1 OpenText CARS>/certificates/<master2 certificate>.cer"` • `mirrormode on` - Add this directive to the file. 2. **Set the Debug Levels of the Log file** - If the evaluation fails even after verifying the configuration file, you must set the debug levels of the log file. Include `-d -1`arguments while starting the OpenText CARS service. Do the following: • **Linux** : • *`<OpenText CARS_installdir>`*`sbin/slapd -f slapd.conf -h ldaps://<machinename>:<port> -d -1` from terminal.

	• **Windows** : • Open **Services**. • Double-click **OpenText CARS**. • Type `-d -1` as parameters and click **Start** without clicking OK. • Now open the *<OpenText CARS_ installdir>*`/bin//openldap.log` to view the debug logs.

Problem Definition	A SOAP fault stating "File not found or class not found" is displayed.
Solution	Ensure that the AppWorks Platform user has all the required permissions on the folders that are shared across the clustered nodes.

Problem Definition	An error message stating "HTTP 404 File not found" is displayed while accessing AppWorks Platform through IP Load balancer.
Solution	In the IP Load balancer configuration, ensure that "/" character is not used after the port number in the Web server URLs. Example: Wrong configuration is <Proxy balancer://mycluster> BalancerMember http://vclient4:80 / BalancerMember http://vclient5:80 / status=+H </Proxy> Correct configuration is <Proxy balancer://mycluster> BalancerMember http://vclient4:80 BalancerMember http://vclient5:80 status=+H </Proxy>

Problem Definition	OpenText AppWorks Platform (<instance name>) does not start after configuring OpenText CARS HA.
Analysis	This occurs when you configure the same host name for both primary and secondary LDAP service containers. In such a case, both the LDAP Processors are configured to run on the same node, which is not supported.
Solution	The LDAP service container name in wcp.properties file will be the secondary LDAP service container name. ▪ Change this to primary LDAP processor name and restart OpenText AppWorks Platform (<instance name>).

Problem Definition	Deploying a CAP application in one of the cluster nodes fails with error: "ApplicationTransaction 'com.cordys.cap.applicationconnector. Transaction' threw an exception while handling Web service operation 'DeployCAP'."

Analysis	This occurs because the CAP service container is not able to find the classes that are added to classpath after it is started.
Solution	Restart the CAP service container in all nodes of a cluster and try deploying application CAPs.

Chapter 12
Abbreviations and acronyms

The following table provides descriptions of acronyms used in this chapter.

Acronym	**Description**
CoBOC	Collaborative Business Object Cache
CAP	Cordys Application Package
LDAP	Lightweight Directory Access Protocol
SAN	Storage Area Network
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language

Chapter 13
Glossary

Term	Definition
Active Node	The node that currently owns the resources of the cluster and responds to SOAP requests.
Active-Active Cluster	A setup is defined as active-active, if all the nodes in the cluster are actively involved in servicing requests.
Active-Passive Cluster	A setup is defined as active-passive, if some nodes of cluster are servicing the requests, while others are inactive (Passive Nodes).
BPM	A Business Process Model (BPM) is a document in AppWorks Platform that captures a graphical view of a company's business processes in detail. These models can be used for either analysis, documentation, or training/knowledge transfer. It also serves as a platform for process execution in combination with various other artifacts such as XForms, Web services, and messages.
OpenText CARS	OpenText CARS is the format of Lightweight Directory Access Protocol (LDAP) of the product. It serves as a central repository of information on organizations, roles, application packages, middleware, and URI.
Cluster	A group of connected computers, which serve the client request and provides services as a single system. Each computer in this group is referred as Node.
ESB	Enterprise Service Bus (ESB) is an implementation of Service Oriented Architecture, with an ability to integrate systems or applications and is based on open standards.
Heart Beat Communication	A periodic message that shows that a node is in active state.
Application Package	An application package is a bundle of objects or software, which is used to work on AppWorks Platform. It can contain components such as Web service operations, user interfaces, XML content, tasks, roles, or any .jar files generated that are built in AppWorks Platform.

Term	**Definition**
LDAP	Lightweight Directory Access Protocol (LDAP) is a networking protocol that helps in querying and modifying a directory service based on TCP/IP. An LDAP directory enables the user to locate organizations, individuals, and other resources, such as files and devices on the Internet or on a corporate Intranet. In AppWorks Platform, LDAP services are available through OpenText CARS.
Primary - Distributed Installation	To setup clustering or HA, AppWorks Platform must be installed on multiple computers. In this setup, two or more installations of AppWorks Platform share the same directory service (OpenText CARS) and distribute the load of service requests among themselves. The first installation is called Primary and rest is called Distributed installations.
Passive Node	The node that is dormant and is not activated until one or more of the active nodes fail.
Primary Node	The most essential node in a cluster, on which most of the infrastructure installations are carried out.
Problem Registry	In AppWorks Platform, Problem Registry is a framework that helps application developers deal with Service Container failures in handling SOAP requests. Based on the autonomous computing principle, the framework allows the application connectors handling the requests to register the problem to prevent any unexpected behavior such as a crash. When the activity is completed, the connectors remove the registered problem from the framework. The framework is based on State SyncUp, which stores the registered problems. These problems can also be retrieved programmatically.
SAN	A storage area network (SAN) is an architecture to attach remote computer storage devices (such as disk arrays, tape libraries, and optical jukeboxes) to servers in such a way that the devices appear as locally attached to the operating system.
Secondary Node	The standby node that takes over when the Primary node fails.
SOA	Service-Oriented Architecture (SOA) is a collection of services that communicate with each other. The services are self-contained and do not depend on the context or state of the other service. They work within distributed systems architecture.
Service Containers	A Service Container refers to an instance of a Java Virtual Machine (JVM), which processes a request.
Service Group	A Service Group is a logical entity to which client applications can send SOAP messages and receive answers.

Term	Definition
Unicasting	The transmission of messages from a single sender to a single receiver over a network.
State SyncUp	State SyncUp maintains the system state (state of Service Containers) up to date so that the clients for intelligent routing of requests can use this information.
Web Gateway	The SOAP requests and responses are exchanged between two Web applications through this channel. For the first time, the selection of the gateway is made during the AppWorks Platform installation. However, it can be switched later using the customization options available in AppWorks Platform.