MTNZ IFS Technical Architecture Description

IFS Applications 10 Technical Documentation

Critical Infrastructure Resilience Company
5-2-2024

Table of Contents

I. ABOUT DOCUMENT

This document as been compiled on behalf of MTN Zambia by Critical infrastructure company (CIRC) for information purposes only.

In putting together this technical documentation heavy reliance has been placed on the readily available online IFS Applications 10 Technical documentation (see reference link below).

References:

1. https://docs.ifs.com/techdocs/

II. DOCUMENT CONTROL

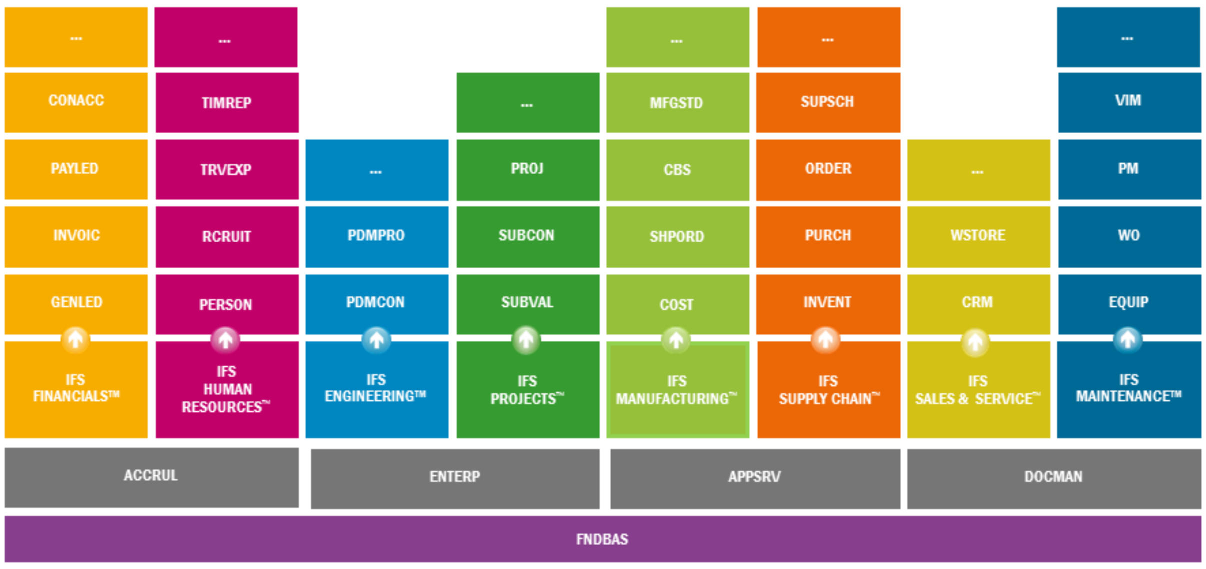
Ver	Documentary activity	Performed by	Date
1.0	Document created	Bassey Ademoyega	02/03/2024
1.0	Document reviewed	Bayo Adogbola	02/05/2024
1.0	Approved for distribution	Emmanuel Williams	03/05/2024

1.0. IFS APPLICATION HIGH-LEVEL ARCHITECTURE

1.1. Components

Component in IFS Applications is used to group entities, processes and client pages into installable units for a logical area of development.
Component enables for independent, parallel development activities as well as selective deployment of IFS Applications. A Component may be dependent on and require other Components to be present. The interfaces between them must be well defined, stable (not likely to change frequently), and limited in number. Components may be separately installed or upgraded at a customer site as long as the interfaces to other components are kept backwards compatible.
A key concept is that of 'Public' interfaces. A public method in one Component allows other Components to interact with that method. Much of the implementation details will be 'hidden' from other Components. This follows the Object-Oriented concept of encapsulation. This allows maximum flexibility in deciding how the internal details will be implemented. However, the public methods must be kept stable since others are now dependent on those interfaces. The general philosophy is to keep 'Public' definitions to a minimum, providing only those methods actually needed by others.
Many Components will detect the presence of other Components dynamically at run time and use functionality in the other Component if it is installed. Some basic support Components are required for IFS Applications, Components such as Enterprise, Accounting Rules, and Application Services.

A Component is the basic unit handled by IFS Deployment (Installation) tools.



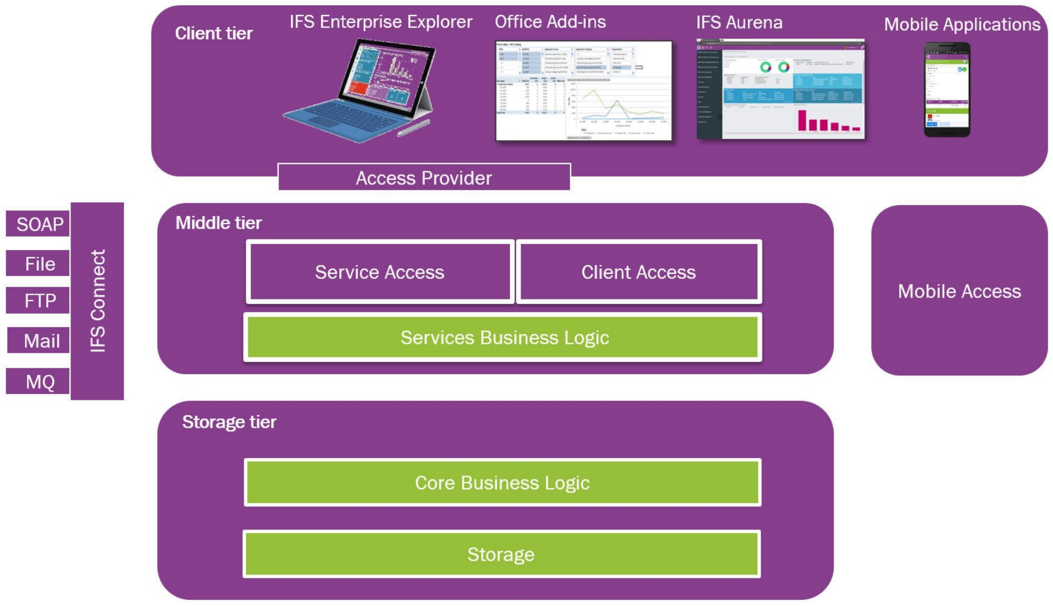
1.2. Multi-tier

IFS Applications Architecture is a multi-tier architecture and is designed using three main tiers with different purposes

- Client tier - Page interaction patterns, UI Shell, Look and feel
- Middle tier - API access, Authorisation, Business logic

- Database tier - Business logic and data storage

Communication between tiers are done using standard protocols such as HTTPS and JDBC.
Each tier has their own software objects representing data and functions. These software objects are all derived from a common model and designed in IFS development tools.

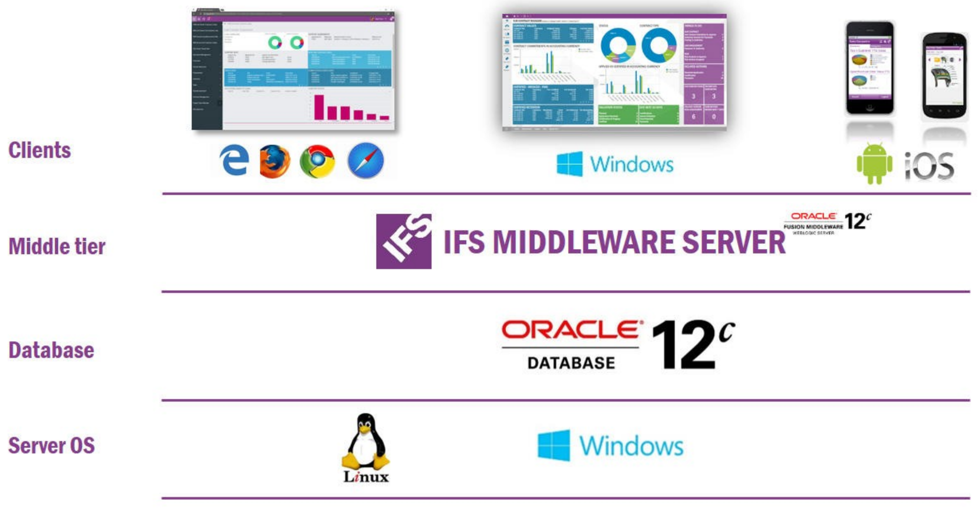


IFS Applications Architecture is divided in to three main tiers, with the business logic available through access providers to IFS user interfaces and custom interfaces.

The UI tier provides interaction with human users and client-side applications and devices. Clients can exists in many different scenarios, traditional desktop and web, mobile apps, add-ins to Office or other productivity software. Regardless which client technology, the same business logic is used.
The business logic tier is the heart of the application. It implements business knowledge, functionality, and processes. This tier is divided into two sub tiers. The core business logic sub tier is a high-performance, object-oriented implementation of business-object level and activity-level business logic. Above core business logic, an service access tier defines the business logic access, the API. This API then used for integrations, client access, and process level logic.
The fully normalized data storage tier is based on the relational database model. The database server is configured so that no table data can be accessed directly. All data modifications are done through the business logic, this guarantees data integrity and prevents "back-door" modification.

1.3. Deployment and platforms

IFS Applications is built using standard tools and technologies. IFS supports Microsoft Windows Server and Linux® as server platforms. Because each physical tier in the architecture is separated through standard protocols, it is possible to "mix and match" platforms in a deployment. IFS Applications can be deployed on anything from a single laptop running all components for demo purposes, to multi-server clustered high-availability installations - supporting tens of thousands of concurrent users.



The implementation at MTN Zambia is currently on Windows for the application server and Unix for the Database

The database runs both the storage tier and the application core business logic objects. The application server called IFS Middleware Server runs the services layer business logic objects and integration services. Both the database and the application server can be run in clustered configurations for extreme reliability and scalability.
Clients access the business logic using the https protocol. This allows easy passage through firewalls, proxies, and other network infrastructure. Ports can be configured. Integrations and customer interfaces access the business logic through the same access providers as used by IFS Applications clients, and thus use the same protocols.

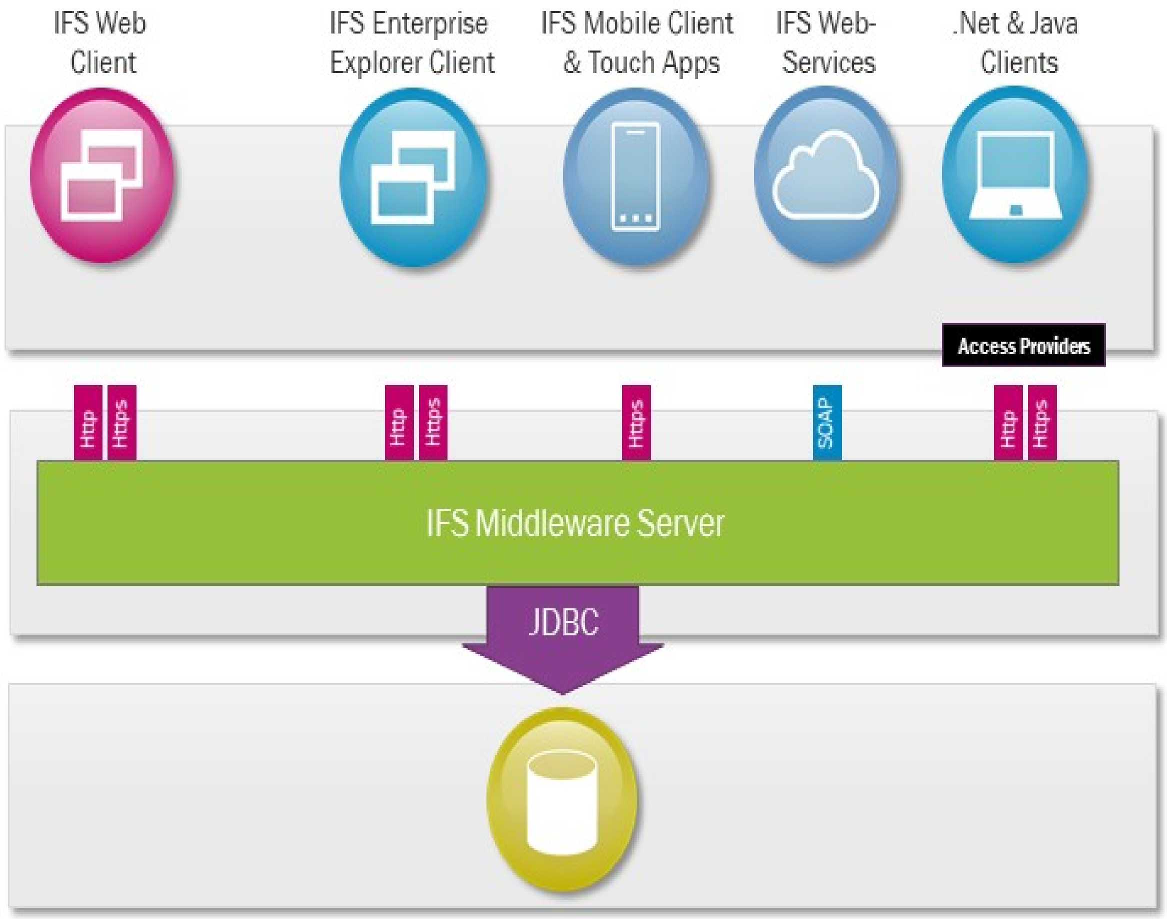
2.0. IFS APPLICATION ARCHITECTURE (LLD)

2.1. Core Server

IFS Service-Oriented Component Architecture is a layered, multi-tier architecture. Core Server includes the core runtime and development frameworks and libraries for IFS business logic.
Base Server contains the framework for for Oracle PL/SQL-based business logic and storage. Middle Tier contains the framework for services layer sub tier of the business logic tier. Access providers are small client-side high-level programming libraries that contain all the functionality and APIs needed for easy access to the business logic.

2.2. IFS and Access Providers

IFS has chosen to package the access APIs into separately installable access providers so that any application, whether it uses an IFS client framework or not, can have full access to business logic. Although all access providers provide the same functionality the implementation is different for each targeted platform (Java, .NET). The reason for this is to make them as natural as possible to use. For example the Java access provider is written in 100% Java. Similarly native mechanisms for exception handling, object collections etc. are used. If an error occurs in the business logic the Java access provider will throw a Java exception to the application. All of this makes it very easy for a developer to build an application (with or without UI) that uses the application business logic using any of the supported development environments. The developer is only working with objects and technologies that are native to the development environment he knows and understands.



Because all the functionality and APIs required to use the business logic have been separated into access providers, both IFS client applications and other applications can enjoy the same easy way to access the business logic.

2.2.1. How it works

Access providers use a simple request-response metaphor for invoking business logic. The basic principle works like this:

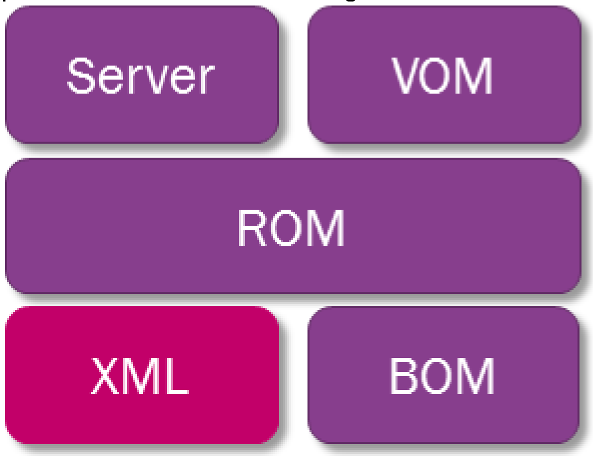
1. The client program creates the request document. This can be either an XML document or a document built using the BOM, ROM or VOM object models (see below)
2. The client program Invokes an Activity or Business API in the business logic, passing the request document along.
3. The access provider handles all required communication with the servers, where the request is processed and a response document is created. The response document will be of the same format as the request document. For example if an XML request document was passed, then the response will also be an XML document.

4. The client application reads the response document and continues execution.

Important to know is that the transaction control happens in it's entirely on the server. Each invocation becomes it's own transaction.

2.2.2. Access Provider Contents

As much as the technical implementation differ between the access providers, they all have a common set of functionality which they provide to the applications. Each access provider includes the following:



- **XML parser**
  The XML parser is not developed by IFS, but it is an important part of an access provider. The XML parser is used to create XML documents to be passed to a server request. Each access provider uses the standard XML parser for the target platform.
- **Buffer Object Model (BOM)**
  The buffer object model is a low-level object model to construct request and response documents. It serializes to a binary format rather than a text format like the XML documents. Because of the binary format used, the BOM gives significantly better parsing times and smaller documents than is possible with XML.
- **Record Object Model (ROM)**
  This is a high level record object model. The basic constructs are records (representing one data entity), record collections (a set of records, or many entities in one lump), and attributes (the individual information attributes of a record). Depending on the target platform the record object model may be built on top of existing data models and api:s on that platform. The ROM uses the same binary serialization format as the BOM.
- **View Object Model (VOM)**
  The view object model consists of real classes representing the various entities and views used in the business logic. Access providers that support VOM contain functionality to generate class representations of any entity or view in the business logic. This means that the developer will get compile-time rather than run-time checking of record types, attribute names etc. The generated VOM classes internally use the ROM to construct the request and response documents.
- **Server**
  The server object is the object that handles the communication with the business logic. It encapsulates the underlying protocol and provides additional services (see below). Client applications use the server object to invoke the business logic, passing a request document along. Client can use request documents built using either of the document object models XML, BOM, ROM, or VOM.

In addition to the core components listed above, the access providers also provide other services useful to client applications. These services are:

- **Data compression**
  Compression of the request/response documents sent and received.
- **Server debug trace**
  All application servers optionally output a debug trace form the execution. The access provider let's the client enable different categories of debug trace in the server, and to retrieve the complete trace text after a call to the server has been made.
- **Statistics**
  Keeping track of statistics such as the amount of sent and received data, time used for server invocations, and number of server invocations made.

From a developer's point of view an access provider is basically a Software Development Kit (SDK) which allows him/her to build an application which interacts with the business logic. Not all component and services are implemented by all access providers.

2.3. Base Server

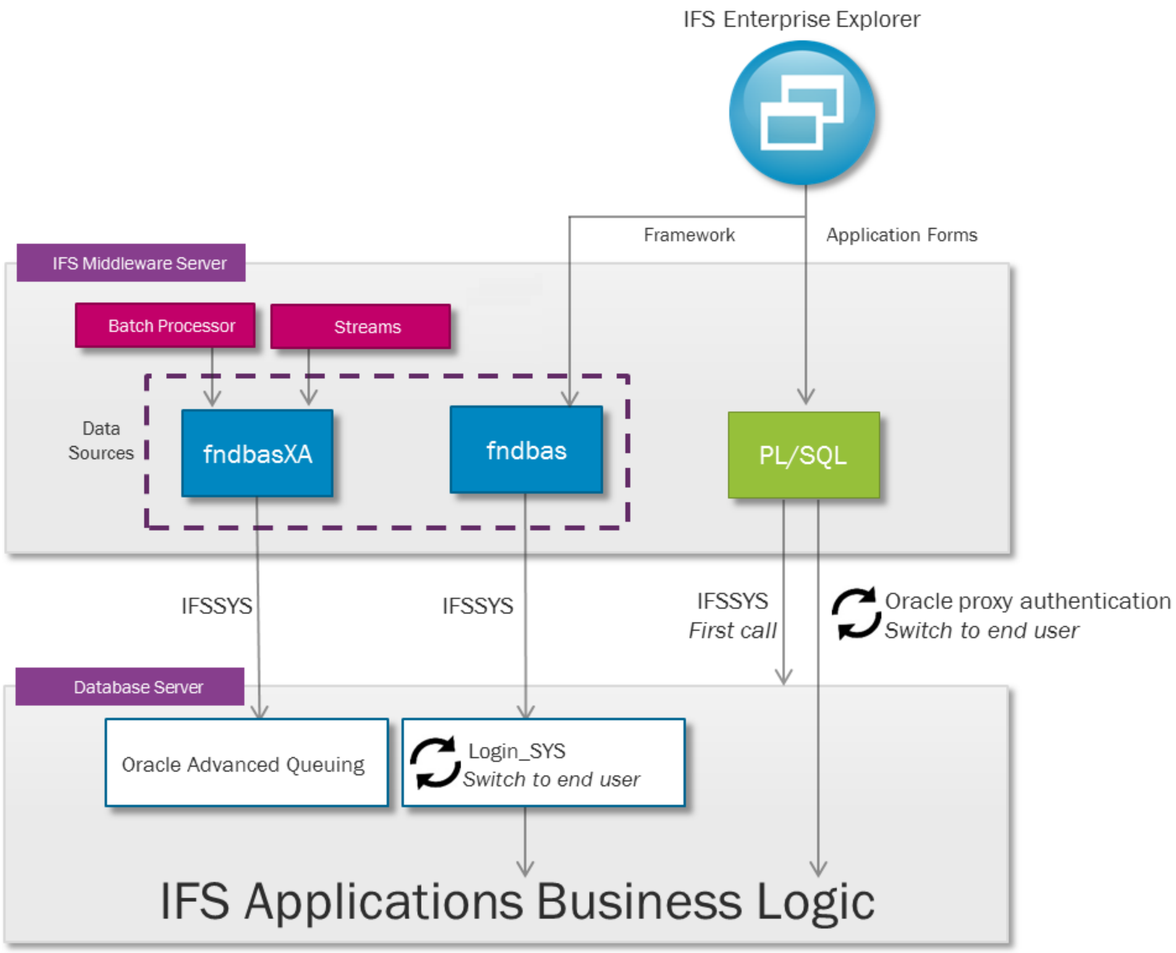
Base Server includes the framework and APIs for IFS Applications business logic and storage in the database tier. This business logic is based on database packages and views, storage is based on tables and indexes.

The database tier is using an Oracle database to run business logic, store and retrieve data in an acceptable time. IFS Developer Studio is the tool that is used to develop business logic with the Base Server framework.
The Base Server framework has many central concepts and areas including:

- Security
- Events
- Connectivity
- Background Processing
- Monitoring
- Data Archiving
- Application Search
- Subscriptions
- Custom Objects

2.4. Middle-tier database connection pooling

The IFS middle-tier makes use of three separate pools of database connections. Such connections are shared between multiple users. The three connection pools serves different use cases and have very different behavior.



Overview of IFS Applications three separate pools of database connections

2.4.1. Data Sources

Two of the database connection pools are managed by the application server as Data Sources:

- **fndbas**
  This connection pool is used for middle-tier Activities and Services, such as bizAPIs, web services, some IFS Enterprise Explorer framework services, IFS Solution Manager, etc.
  All connections are created as user *IFSSYS*, but when a connection is to be used by a client call the current Foundation1 user identity is temporarily switched with a call to *Login_SYS.Init_Fnd_Session* (the Oracle database user is still *IFSSYS).* The connections are shared between all interactive clients and service consumers.
  When monitoring database sessions, connections from this pool are identified as category *Activity/Service*.
  The minimum and maximum size of the connection pool (number of database connections) can be configured using the Installer. The size of the connection pool will vary, depending on the current load, between these configured sizes.

- **fndbasXA**
  This connection pool is used only by the Application Server integration with Oracle Advanced Queuing.
  Both the Batch Processor and Streams use connections in this pool in

order to subscribe to messages sent from the database to the application server. Such connections are never used to invoke business logic and are never used by interactive clients.
All connections are created as user *IFSSYS*.
When monitoring database sessions, connections from this pool are identified as category *JMS-AQ integration*.

Both connection pools can be monitored using IFS System Monitoring and configured using the Installer.

2.4.2. PL/SQL Access

This type of connection pool is used by interactive clients when accessing database business logic directly. The behavior of this connection pool is very different from the two application server managed connection pools. All connections are created as user *IFSSYS*, but rather than temporarily switching only the Foundation1 user identity when a connection is in use the Oracle database user is also switched using *Oracle proxy authentication*. Connections are shared between clients, but the pool will attempt to reuse an already switched connection (where the database user identity matches that of the current user) as a way to minimize the cost and impact of database user switching. If no such idle connection is available the database identity will be switched to that of the current user.
There are two scenarios where connections are *reserved* for a particular client session - when the client keeps an open cursor (this typically happens when scrolling in a table window with a large result set) or when the client keeps a database transaction active between client-server calls. In both these scenarios is absolutely vital that the *same* database connection is used for all database calls.
The *state* of the connection indicates whether the connection is currently:

- USED - Ongoing database call
- INITIALIZED - User identity switched to some database user
- ANONYMOUS - Not yet initialized connection, will have to be initialized before use
- RESERVED - The connection is dedicated to a particular client session, either because of an open cursor or an active transaction. The connection will go back to state INITIALIZED automatically after any of the two configurable timeouts explained below.

(Other transient connection states exists.)
There are three important configuration parameters which can be changed using the Installer:

- Maximum Number of Connections
  The connection pool will grow up to this many connections as needed, depending on the current load. Since connections can be shared, this value can be significantly less than the number of concurrent users.
- Cursor Timeout
  The time until an idle open cursor will be closed automatically.
- Transaction Wait Timeout
  The time until an active transaction, waiting for subsequent client-server call, will be rolled back automatically.
  This is basically the maximum user "think" time between server calls in the context of an ongoing transaction started by the client.

The behavior of this pool can be monitored using IFS System Monitoring and configured using the Installer.
When monitoring database sessions, connections from this pool are identified as category *PL/SQL Access*.

3.0. IFS Middleware Server

3.1. Overview

The Application Server, which is used by IFS Applications, is distributed in IFS component "IFS Middleware Server" and is based on Oracle WebLogic Enterprise Edition. Since it is embedded into an IFS component all installation, administration and configuration is performed using IFS supplied tools.
IFS Middleware Server introduces some new keywords and concepts that are important to know in order to understand how the Application Server functions. This page will describe central concepts as Node Manager, Admin Server and Managed Server. It also describes how clustering works with and without an external load balancer and how servers are controlled.
Configuration changes of parameters and cluster of the IFS Middleware Server is made using IFS Admin Console after a fresh install is performed.

3.2. Concepts

An IFS Applications instance requires a number of resources such as data sources, messaging queues and enterprise applications in order to be operational. The umbrella under which these resources are managed is called a *domain*.

Node Manager

Node Manager is a utility that enables starting, shutting down and restarting Admin Server, Managed Servers and HTTP Server on one node either from the host itself or from another node. The Node Manager has low memory footprint and should always run as a service.

Admin Server

The Admin Server is a server that controls the domain configuration. It hosts the IFS Middleware Server Admin Console and IFS System Monitoring Console applications, but no business functionality.

Managed Server

A Managed Server hosts one or more applications. There are two types of Managed Servers - "Main" servers and "Integration" servers.
"Main" servers expose business functionality consumed by interactive clients such as IFS Aurena and IFS Enterprise Explorer.
"Integration" servers implements and exposes functionality with an integration intent (Web Services and IFS Connect). It also implements reporting functionality.

Host

Host is the term used for a physical- or virtual machine, i.e. a computer.

Machine

Machine is the term for the logical representation of a host within the domain.

HTTP Server

The HTTP Server is the entry-point of all applications. The HTTP Server is sometimes referred to as Web Server.

Cluster

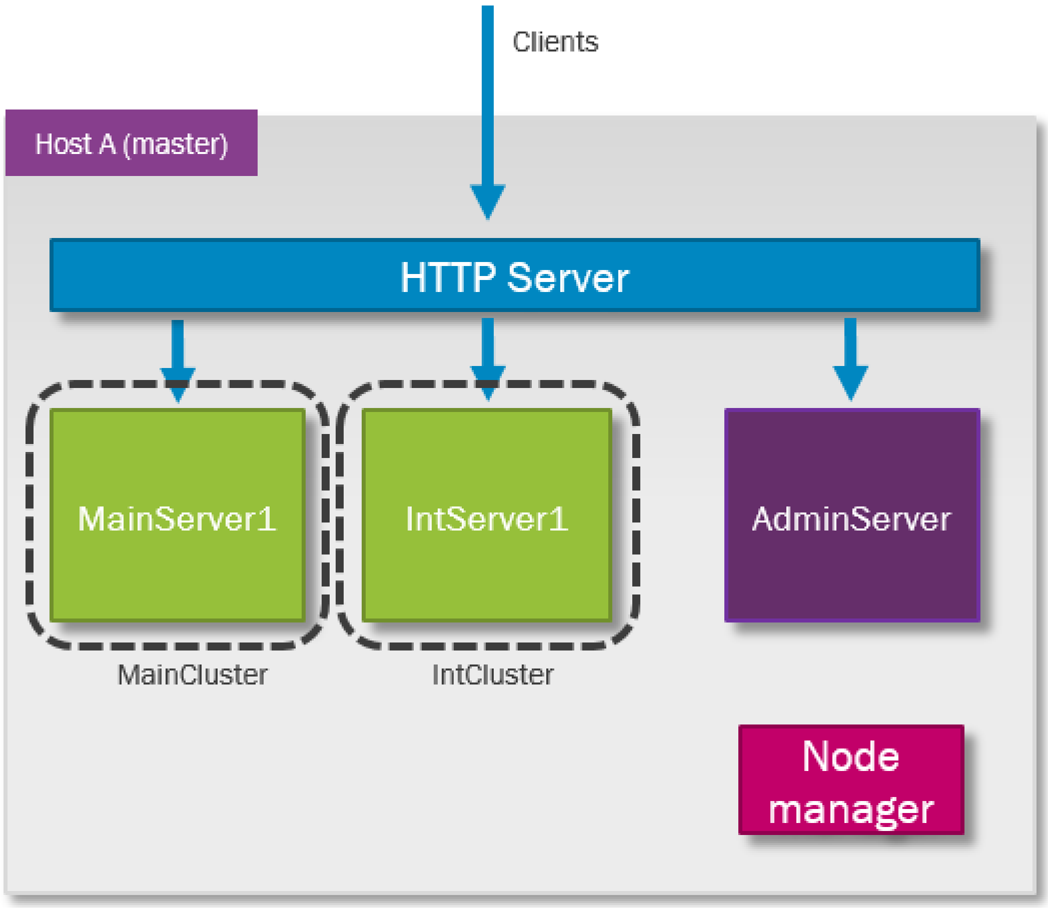
There are two types of clustering; vertical and horizontal. Vertical clustering means that there are two or more servers running on the *same* machine and horizontal clustering means there are two or more servers running on *different* machines. In IFS Middleware Server it is possible to configure how many servers to run on any machine, thus giving the option to run **both** a horizontal and a vertical cluster at the same time.
When forming the cluster, one of the hosts in the cluster will act as the master, which will always be the host where you run the IFS Installer in order to apply patches and new functionality. The master will run the Admin Server which is used as the center point for the cluster. The Admin Server holds the master configuration and the other machine will periodically check for any changes and update their local configuration accordingly. Managed Servers can also be started, stopped and monitored using the Admin Server. No other machines will run an Admin Server.

To add host machines to a cluster the script cluster.cmd is run on the Master node. It will create a zip file with all neccesary configuration and software. The zip file is then copied over to all hosts to be added. The zipfile is unzipped and the cluster.cmd is run on all new hosts. The hosts will now register themselfs to the AdminServer. In the Admin Console the new host will be available and new managed servers can be deployed into the MainCluster and the IntCluster on any of the new hosts.

The number of managed servers each machine will host is configurable, it might not contain any Managed Servers at all, a scenario which would only make sense in case you wish to decrease the load of the master node. Each host will also contain everything necessary to set up an HTTP Server. However, by default it will only run on the master after installation. Additional HTTP Servers can be set up using the IFS Middleware Server Admin Console, but this is really only required in case an external load balancer is used.

3.3. Default configuration

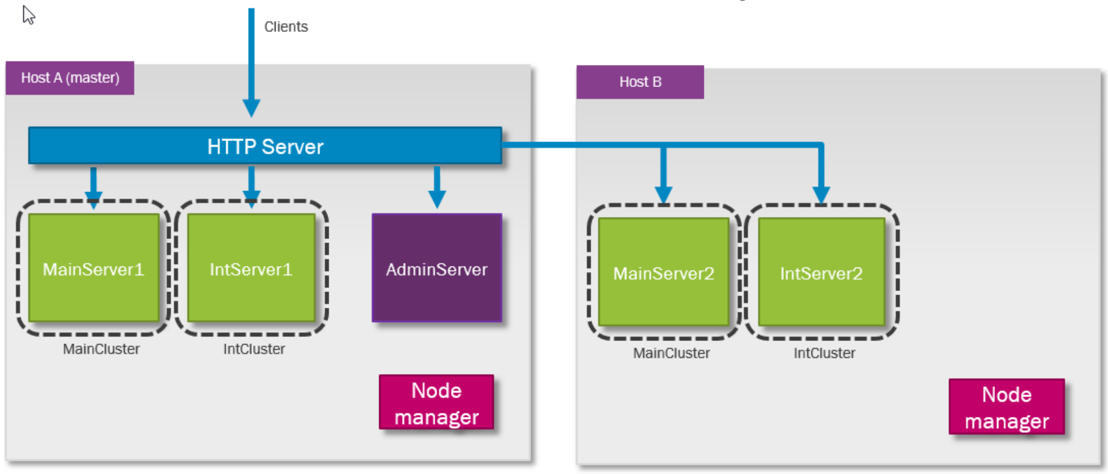
The default configuration for IFS Middleware Server is built up using a single machine hosting two managed servers as illustrated below. Note that there are two separate clusters - *MainCluster* exposing services used by interactive clients and *IntCluster* exposing integration services. It's this default configuration that is always set up in a fresh install. To extend a default configuration with horizontal or vertical cluster the Admin Console will be used.



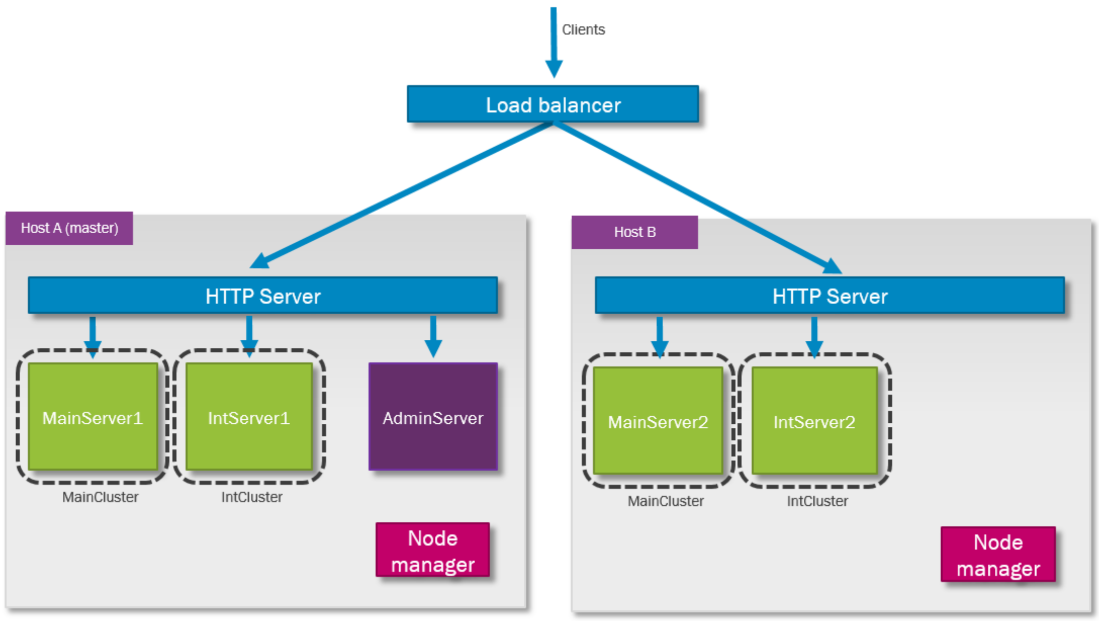
Note: This is the current set-up for MTN Zambia IFS Applications 10.

3.4. Extended configuration

In the example configuration below an additional host has been added hosting two additional managed servers. Note that the two clusters now stretches across two machines. Also note that only the master host runs a HTTP Server, which in this configuration becomes a single point of failure. A real highly available installation would also use an external load balancer as shown in the second image below.



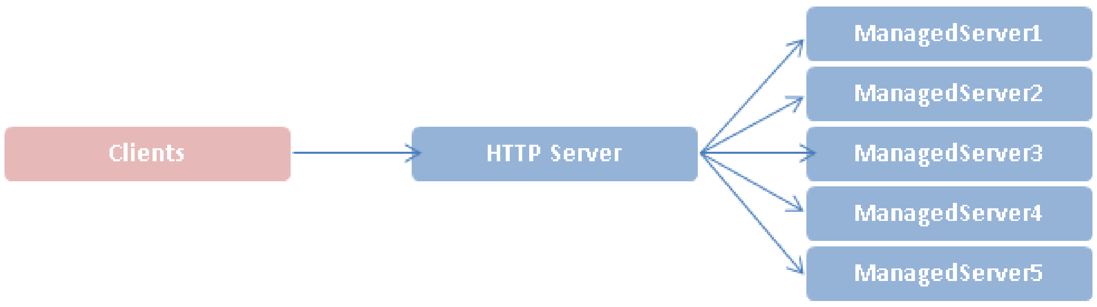
Horizontal cluster with a single HTTP Server.



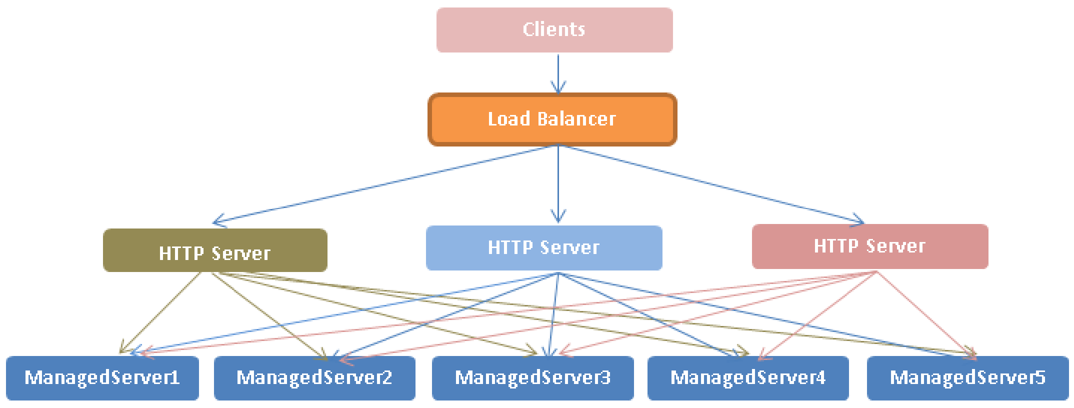
Horizontal cluster with two HTTP Servers and an external load balancer.

3.5. Load Balancing

The load is always balanced between the application servers by the HTTP Server no matter if there is an external load balancer in front or not. This typically eliminates the need of having more than one HTTP Server running when there is no external load balancer.



Load balancing example for one HTTP Server and five managed servers in a cluster.
While the HTTP Server distributes the load between the application servers, an external load balancer can be used to distribute the load between HTTP Servers, thus increasing the throughput for HTTP calls if this is needed. This means that although the external load balancer forwards your request to Host B you might still end up communicating with a managed server on Host A. The below image describes the load balancing where an external load balancer is used and the HTTP Servers on all hosts are serving requests.



Load balancing example with an external load balancer, three HTTP Servers and five managed servers.

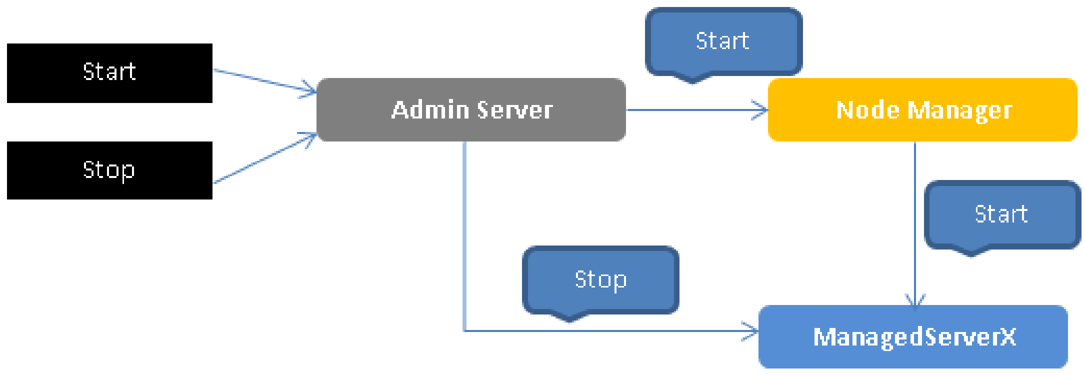
3.6. The Node Manager's Role

The Node Manager is acting as an entry point on each machine. Therefore it is important that it is up and running at all times. It also handles server crash recovery if the system goes down or a running server crashes. This is why it is important that the Node Manager is started automatically when the host starts, only then can it start up the servers that terminated unexpectedly. It is also important that managed servers are stopped correctly using the stop scripts if it needs to be stopped for any reason, otherwise they will bounce right up again.

3.7. The Admin Server's Role

The Admin Server plays an important role in the cluster. As previously mentioned, it is the center point of the entire cluster. It maintains the master configuration and propagates any changes to the application servers; it is responsible for managing new or updated applications and it also starts and stops the managed servers. Should the Admin Server become unavailable, the applications servers will continue to run independently but try to reconnect on a regular basis in order to receive configuration changes.

When starting an application server, the Admin Server is contacted and asked to start the specific server. The Admin Server then contacts the Node Manager on the machine where the application server resides which in turn tells the managed server to start and reports back to the Admin Server. The Admin Server can now communicate directly with the application server and can tell it to stop.



Example of starting or stopping a managed server.

More information about how to control the cluster can be found here.