

1 概要・基本トポロジー	3
1.1 基本概念	3
1.1.1 パッケージング	4
1.1.2 WEBSphere APPLICATION SERVER	6
1.1.2.1 概念図	6
1.1.2.2 プロファイル	7
1.1.2.3 ノード	8
1.1.2.4 セル	8
1.1.2.5 Web サーバーと Web サーバー・プラグイン	8
1.1.2.6 Web コンテナ	8
1.1.2.7 EJB コンテナ	9
1.1.2.8 エージェント管理	9
1.1.2.9 管理サービス・管理ツール	9
1.1.3 WEBSphere APPLICATION SERVER NETWORK DEPLOYMENT	10
1.1.3.1 概念図	10
1.1.3.2 デプロイメント・マネージャーとノード・エージェント	11
1.1.3.3 クラスター	12
1.1.3.4 Edge Component	14
1.1.3.5 ジョブ・マネージャー	18
1.1.3.6 WebSphere プロキシ・サーバー	19
1.1.3.7 セキュア・プロキシ・サーバー	19
1.1.3.8 オンデマンド・ルーター (ODR)	19
1.2 トポロジー	20
1.2.1 WAS BASE と WAS ND の選択	20
1.2.2 WAS BASE トポロジー設計の考慮点	21
1.2.2.1 Web サーバーの配置	22
1.2.2.2 Web コンテナと EJB コンテナの分離配置	22
1.2.2.3 複数アプリケーション・サーバーの配置	23
1.2.2.4 複数 Web サーバーの起動	23
1.2.2.5 管理エージェントの配置	24
1.2.3 WAS BASE 構成例	25
1.2.3.1 単一マシン構成	25
1.2.3.2 単一アプリケーション・サーバー/リモート Web サーバー	26
1.2.3.3 複数アプリケーション・サーバー/ローカル Web サーバー	26
1.2.3.4 WAS Base シンプル・クラスター	27
1.2.3.5 セッション・データベース	29
1.2.4 WAS ND トポロジー設計の考慮点	29
1.2.4.1 デプロイメント・マネージャーの配置の考慮点	30
1.2.4.2 高可用性デプロイメント・マネージャー構成 (HA DM)	31
1.2.4.3 ジョブ・マネージャーの配置の考慮点	32
1.2.4.4 垂直クラスター構成と水平クラスター構成	32
1.2.4.5 静的クラスター構成と動的クラスター構成	35
1.2.4.6 セッション・パーシスタンス・トポロジー	39
1.2.4.7 Web サーバー配置の 1:N 構成の検討	42
1.2.4.8 Web サーバー配置の 1:1 構成の検討	43

1.2.5 WAS ND 構成例	45
1.2.5.1 負荷分散装置とクラスター (リモート Web サーバー)	45
1.2.5.2 負荷分散装置とクラスター (リモート Web サーバー 1:1 構成).....	46
1.2.5.3 Web コンテナと EJB コンテナの分離.....	47
1.2.5.4 動的クラスター	48

1 概要・基本トポロジー

この章では WebSphere Application Server (以下 WAS) アーキテクチャーを理解するために、これを構成するコンポーネントと、そのトポロジーについて説明します。

1.1 基本概念

WAS V8.5 は、Java EE アプリケーションの実行環境を提供する他、標準搭載した HA (高可用性) 機能、メッセージング・エンジンの搭載、OSGi や SCA などのオープン・スタンダードのサポートなどにより、SOA (サービス指向アーキテクチャー) に基づいた、オンデマンド・ビジネスの実現をサポートします。さらに、オンデマンドをサポートする製品群の基盤として、迅速な展開、スケーラビリティを提供します。

WAS には、様々なエディションがあり、システムの規模や要件に応じて、最適なエディションを選択することができます。この節では、シングル・サーバーでの環境を提供する WAS Express および WAS Base による構成と、マルチ・サーバー環境を提供する WAS Network Deployment (以下 WAS ND) による構成の基本コンポーネントについて説明します。

1.1.1 パッケージング



WAS ファミリー製品は、以下の各種 OS の上で稼動します。
(各種 Linux, AIX, Windows, Solaris, HP/UX, OS/400, i5/OS, および z/OS)

図 1-1 製品紹介

WAS Express は、エントリー・レベルに位置し、スモール・ビジネスでの利用やソリューション・パッケージのベースとして使用されることが期待される製品です。WAS Express には 480 プロセッサー・バリュー・ユニット (PVU) までという制限があり、また 64bit には対応していません。

WAS Base はその上位製品で、シングル・サーバーでのアプリケーション実行環境を提供しています。製品名は WAS となっていますが、ここでは WAS ND と明示的に区別するために、WAS Base と呼びます (一般にはシングル・サーバー構成の WAS と呼ばれる場合もあります)。

WAS ND は更にその上位製品として位置づけられており、マルチ・サーバー環境のためのさまざまな拡張機能を提供します。WAS Base と異なり、サーバーが複数ある場合でも管理を個別に行う必要がなく、セルと呼ばれる単位で統合的に管理することが可能です。更に、フレキシブル・マネジメントにより、複数のセル環境、Base/Express 環境をまとめて管理する機能も持っています。また、Web サーバーへの HTTP リクエストの負荷分散機能を持つ Edge Component や、その他のソフトウェアが同梱されています。

WAS V8.0 までは WebSphere Virtual Enterprise (WVE) という名称の追加製品にて提供されていた、流量制御/優先制御/無停止運用などの機能が、WAS V8.5 からは ND で追加ライセンスなしで利用できるようになりました。これらの WVE 相当の機能は、WAS ND V8.5 では Intelligent Management (図 1-1 製品紹介では「Intelligent Mgmt.」) という名称で提供されています。

WAS Express、Base、ND の各製品に含まれる機能と同梱されるコンポーネントをまとめると、下記の表のようになります。製品はすべて Java EE 6 を完全サポートしています。また、Web サーバーとして IBM HTTP Server (IHS) が同梱されています。

表 1-1 製品比較

	Express	Base	ND
Java EE6 / Java SE 7 サポート	○	○	○
HttpSession 共有	○ ・2 サーバーまで (※1) ・方法:DB	○ ・5 サーバーまで (※1) ・方法:DB	○ ・無制限 ・方法:DB またはメモリー
Web サーバー・プラグインが割り振れるアプリケーション・サーバー台数	2 台 (2JVM) (※1)	25 台 (25JVM) (※1)	無制限
ライセンス上の制限	480PVU まで	無制限	無制限
IBM HTTP Server	○	○	○
Edge Components	×	×	×
Tivoli Directory Server (LDAP)	×	×	○
Tivoli Access Manager Server	×	×	○
64bit 対応	×	○	○
DB2 Workgroup Server	○ (用途に制限有)	○ (用途に制限有)	○ (用途に制限有)
Liberty プロファイル	○	○	○
Intelligent Management	×	×	○
Deployment Tool	・IADT(※2)	・IADT(※2)	・IADT(※2)

(※1) WAS V8.5.5 では、HttpSession 共有/Web サーバー・プラグインからの割り振り対象の台数制限が撤廃されています。

(※2) IADT:IBM Assembly and Deploy Tools for WebSphere Administration

Intelligent Management で実現される機能

WAS ND V8.5 の Intelligent Management では、以下のような機能が提供されます。

1) インテリジェント・ルーティング

- ・ ビジネスの優先度に応じて、重要度の低いリクエストをキューイングし、重要度の高いリクエストを即座に処理するよう制御します。

2) 管理パフォーマンス

- ・ 負荷状況に応じて割り振りの重み付けを動的に計算し、システム全体としての最適化を図ります。
- ・ リソースの使用状況に応じてアプリケーションの配置を変更し、サービス提供を行う JVM の数を動的に制御する機能を提供します。

3) 管理ヘルス

- ・ 障害兆候を検知すると、事前に定義した問題解決のためのアクション・プランが実行されます。

4) アプリケーション・エディション管理

- ・ アプリケーション・バージョンの短時間でのリリースとロールバックを実現します。

- ・ アプリケーションの複数バージョンの並行稼動により、本番稼動と稼動確認が同一環境で実行可能となります。

1.1.2 WebSphere Application Server

この節では、WAS 製品全てに共通する基本概念と、WAS Base に搭載される基本機能について説明します。

Java EE においては、アプリケーションはサーブレットや JSP, EJB といったコンポーネントによって構成されます。アプリケーション・サーバーは、これらの実行環境を提供するものです。WAS のアプリケーション・サーバーは、Java EE における Web アプリケーション・ランタイム環境 (Web コンテナ)、EJB ランタイム環境 (EJB コンテナ) を実装しており、JVM プロセスとして実行されます。アプリケーション・サーバーは複数起動することも可能ですが、負荷分散を実現し、可用性を高めるクラスター構成にするためには WAS ND が必要となります。

1.1.2.1 概念図

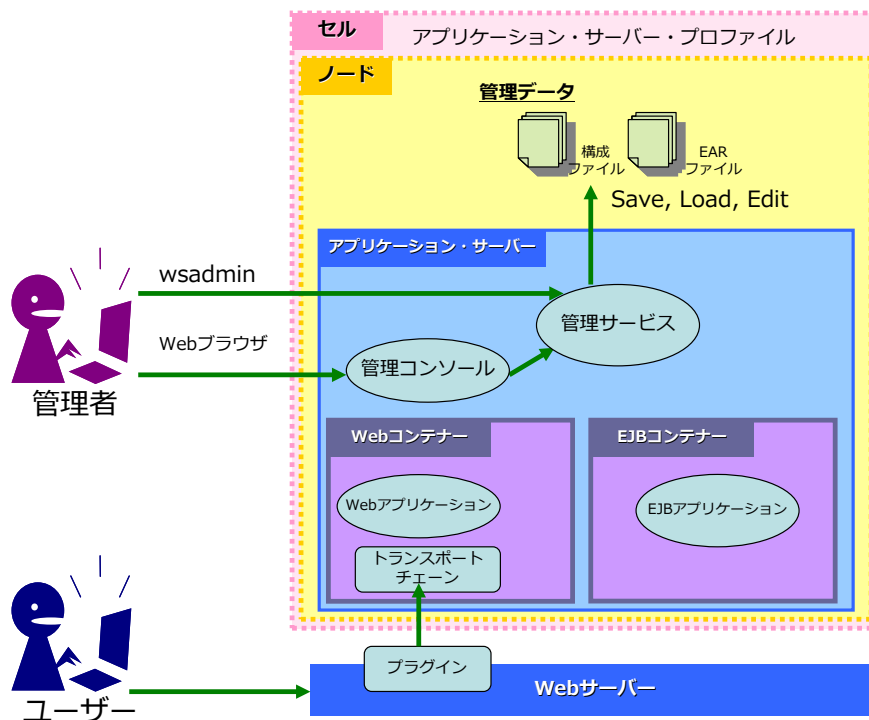


図 1-2 スタンド・アローン構成

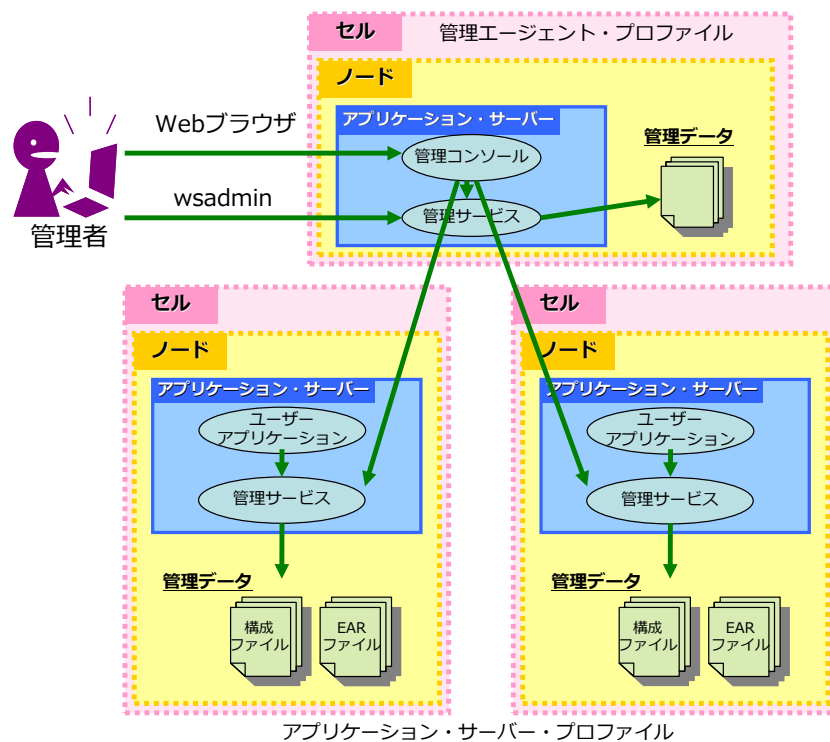


図 1-3 複数アプリケーション・サーバー構成

WAS Base を構成する要素として、Web サーバーとアプリケーション・サーバーがあります。これらは便宜的にそれぞれ独立したノードを構成し共通のセルに属します。アプリケーション・サーバーでは管理コンソールが動き、Web コンテナや EJB コンテナ上でユーザー・アプリケーションが実行されます。この構成を作成するためにはアプリケーション・サーバー・プロファイルを使用します。

Base 構成では管理エージェントを別にたてることができます。管理エージェントは独立したセルの中で稼働し、同じ筐体内で実行されている複数のアプリケーション・サーバーを管理することが可能です。管理エージェントをたてるには、管理エージェント・プロファイルを使用します。

図 1-2 スタンド・アローン構成は、アプリケーション・サーバーのみの構成です。一方、図 1-3 複数アプリケーション・サーバー構成は複数のアプリケーション・サーバーと管理エージェントからなる構成です。こちらの構成では、複数のアプリケーション・サーバーを統合管理することが可能です。

1.1.2.2 プロファイル

プロファイルとは、WAS のランタイム環境を定義するファイルの集まりです。WAS をインストールしたあと、プロファイル管理ツール (第 2 章を参照) によって作成します。一つの WAS インストールイメージに対して複数のプロファイルを作成することが出来ます。それぞれのプロファイルごとにディレクトリーが作成され、構成ファイル、データ・ファイル、ログ・ファイルなどは、すべてその中に配置されます。

プロファイル管理ツールで作成できるプロファイルの種類としては、以下の 7 種類があります。

- セル・プロファイル (ND でのみ作成可能) (WAS V6.1 より新規追加)
： デプロイメント・マネージャー・プロファイルとカスタム・プロファイルが同時に作成されます。
- デプロイメント・マネージャー・プロファイル (ND でのみ作成可能)
- アプリケーション・サーバー・プロファイル
- 管理エージェント・プロファイル (WAS V7.0 より新規追加)

- ジョブ・マネージャー・プロファイル (ND でのみ作成可能) (WAS V7.0 より新規追加)
- セキュア・プロキシ・プロファイル (ND でのみ作成可能) (WAS V7.0 より新規追加)
- カスタム・プロファイル (ND でのみ作成可能)

WAS V5.1 以前は、WAS のインストール後に各コンポーネントを個別にインストールする必要があり、また修正プログラムも個別に適用する必要がありました。WAS V6 以降では、各コンポーネントは WAS 上でプロファイルとして作成することができ、製品に対する修正プログラムは、インストールした 1 つの製品ファイルに適用するだけになりました。この利点としては、導入作業の軽減、プロファイルを削除するだけで初期状態に戻すことができること、またディスク・スペースの削減などが挙げられます。

1.1.2.3 ノード

ノードは1つまたは複数のコンポーネントから構成される、論理的なグループです。WAS Base の場合、1 つのアプリケーション・サーバーや管理エージェントがそれぞれノードに相当します。WAS ND の場合、アプリケーション・サーバーとノード・エージェントはセットで 1 つのノードを構成します。また、他に、デプロイメント・マネージャー・ノードやジョブ・マネージャー・ノードがあります。

1.1.2.4 セル

WAS において、各ノードはセルというグループに属します。セル内の複数のノード、アプリケーション・サーバーそしてアプリケーションは共通の管理データで管理されます。WAS ND 環境においては、セル内のノードの構成情報およびアプリケーションはセル内に1つだけ存在するデプロイメント・マネージャー (1.1.3.2デプロイメント・マネージャーとノード・エージェントを参照下さい) によって各ノードのノード・エージェントを経由して管理されます。セル内のすべてのノードのアプリケーションと構成情報はデプロイメント・マネージャーによってマスター構成情報として管理されます。

1.1.2.5 Web サーバーと Web サーバー・プラグイン

Web サーバーは、ブラウザからのリクエストを最初に受け付けます。Web サーバー・プラグインは、アプリケーション・サーバーとの通信を行うためのものです。アプリケーション・サーバーで実行を必要とするリクエストが Web サーバーにきた場合、Web サーバー・プラグインを経由してリクエストがアプリケーション・サーバーに転送されます。

アプリケーション・サーバーではリクエストの内容に応じて、Web コンテナ上でのサーブレットや JSP の実行、EJB コンテナでの EJB の実行を行い、結果は再び Web サーバーを経由してユーザーに戻されます。Web サーバー・プラグインは、クラスター構成時のセッション管理で必要とされるセッション・アフィニティの機能も持っています。Web サーバー・プラグインの構成情報は、XML 形式でプラグイン構成ファイルに保管されます。

1.1.2.6 Web コンテナ

Web コンテナは、サーブレットや JSP ファイルなどの処理を行います。Web コンテナは下記の機能を提供します。

- Web コンテナ・トランスポート・チェーン
 - : Web コンテナの内部トランスポート・チェーンを使用して、ユーザー・リクエストは Web コンテナに渡されます。このトランスポート・チェーンは以下のチャンネルで構成されます。
 - ▶ TCP インバウンド・チャンネル: ネットワーク接続を提供
 - ▶ HTTP インバウンド・チャンネル: HTTP1.0 と 1.1 のリクエストの送受信
 - ▶ Web コンテナ・チャンネル: サーブレットや JSP へのリクエストを Web コンテナに送信

- サブレット処理
： サブレット要求を受信したときに、Web コンテナは、リクエスト・オブジェクトとレスポンス・オブジェクトを生成し、サブレットのサービス・メソッドを実行します。
- HTML などの静的コンテンツの処理
： サブレットや JSP などの動的コンテンツだけでなく、HTML などの静的コンテンツのリクエストも Web コンテナで処理できます。
- セッション管理
： サブレットの仕様に準拠し、同一ユーザーからのリクエストを識別するセッション管理の機能を提供します。
- Web サービス・エンジン
： Java EE 仕様の一部として Web サービスの API が規定されています。Web サービス・エンジンは SOAP (Simple Object Access Protocol) のサポートを提供します。

1.1.2.7 EJB コンテナ

EJB コンテナは、EJB のインスタンス管理、メソッド実行、トランザクション管理、ネーミング・サービスの提供等、EJB を利用するために必要なサービスを提供します。EJB には以下の 3 種類があります。

- セッション Bean
： サブレットや他の EJB からの処理の依頼を受け付け、処理を実行し、結果を返します。
- エンティティ Bean
： データベースのデータをオブジェクト化し、他のセッション Bean やメッセージ・ドリブン Bean から利用できるオブジェクトを提供します。EJB3.0 以降では利用されず、代わりに JPA という形で新たに仕様が定義されています。
- メッセージ・ドリブン Bean (MDB)
： メッセージ・キューに到着したデータを受け付け、処理を実行します。

1.1.2.8 エージェント管理

管理エージェントは、WAS V7.0 で新しく加わった、WAS Base 構成、WAS Express 構成を集中管理するためのコンポーネントです。管理対象のアプリケーション・サーバーから構成管理/アプリケーション管理/コマンド・マネージャー/ファイル転送/管理コンソールの機能を引き継ぎ、アプリケーション・サーバーのフットプリントを縮小させ起動時間を短縮させることができます。また、**エラー! 参照元が見つかりません。** エラー! 参照元が見つかりません。で述べますが、管理エージェントはジョブ・マネージャーからシングル・サーバー・ノードの代理としてジョブを取得し、各シングル・サーバー・ノードに対してジョブを実行させることが可能なため、ジョブを非同期に実行させることを可能にします。管理エージェントを利用せずに、ノードを管理対象外にすることも可能です。管理エージェントで管理するためには、ノードに対して登録を行う必要があります。

1.1.2.9 管理サービス・管理ツール

WAS におけるアプリケーション・サーバーや各コンポーネントの構成管理は、管理ツールで行います。管理ツールには、以下の 4 種類があります。

- 管理コンソール
- コマンド行ツール
- wsadmin

- JMX プログラム

これらの管理ツールについては、第 2 部第 1 章を参照下さい。

WAS の構成情報は XML ファイルの形式でファイル・システムに格納されます。ですが、WAS の構成変更を行う場合は、直接 XML ファイルを編集するのではなく、管理コンソールや wsadmin といったインターフェースを介して編集作業を行います。編集作業を保存することで変更内容が XML ファイルに反映されます。

管理コンソールでは WAS だけでなく、IHS などの Web サーバーも管理することができます。管理できる内容として、Web サーバーの始動・停止、状況のモニター、Web サーバー構成ファイルの表示・構成、Web サーバー・プラグイン構成ファイルの表示・編集、Web サーバー・プラグイン構成ファイルの生成・伝播などがあります。

1.1.3 WebSphere Application Server Network Deployment

WAS ND は、複数のサーバーでのアプリケーション実行環境を提供するエディションです。デプロイメント・マネージャーや Edge Components など、WAS ND にはマルチ・サーバー環境で欠かすことのできない機能が含まれています。本章および次章で述べるマルチ・サーバー環境についての解説は、この WAS ND で提供される機能に基づいて記述しています。

1.1.3.1 概念図

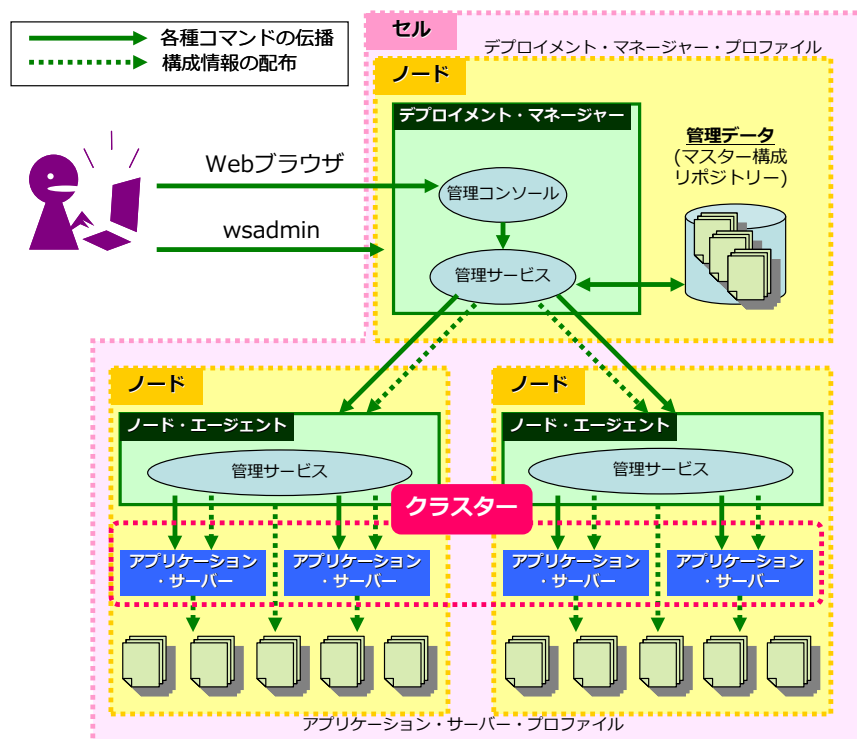


図 1-4 分散サーバー構成

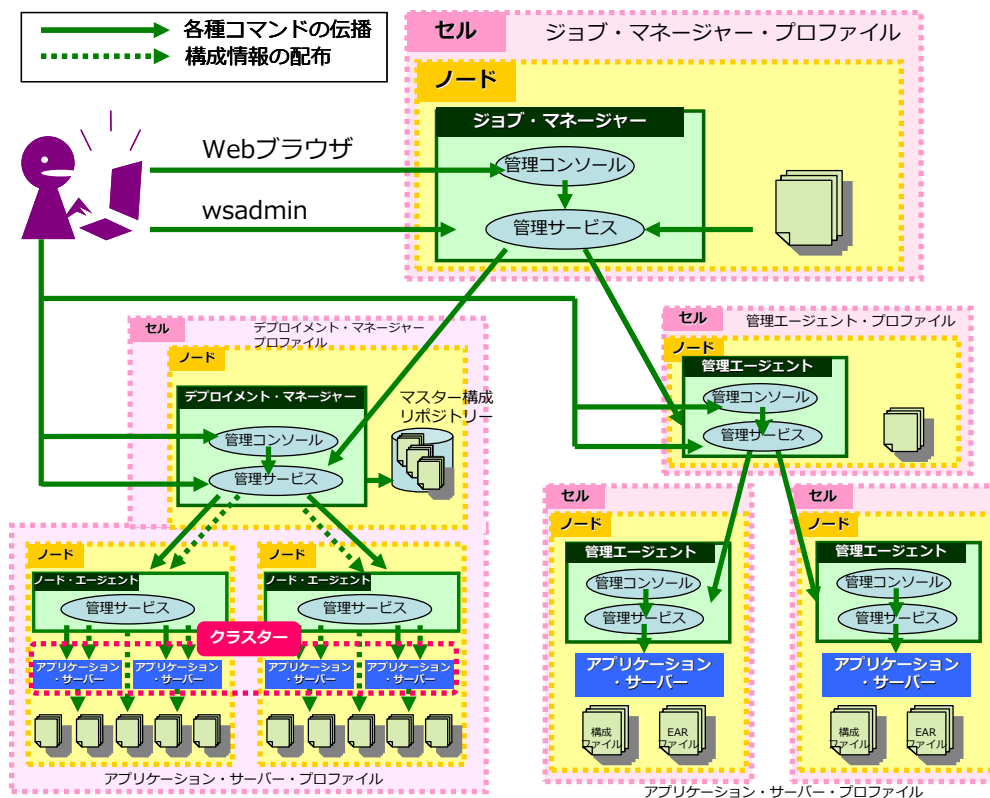


図 1-5 フレキシブル・マネジメント構成

上図は WAS ND 実行環境で取り得る構成を表したものです。分散サーバー構成では、デプロイメント・マネージャーを利用することで、セル内を統合管理することができます。また、フレキシブル・マネジメント構成では、WAS V7.0 で登場したジョブ・マネージャーを利用することで、複数の ND セルと Base/Express 環境を統合管理することができます。

1.1.3.2 デプロイメント・マネージャーとノード・エージェント

デプロイメント・マネージャーは、セル内のすべての構成要素を集中管理する機能を提供します。デプロイメント・マネージャーはそれ自体がアプリケーション・サーバーとして実行されていて、Web ベースの管理インターフェースを提供しています。セル内の各ノードで実行するアプリケーション・サーバーの管理は、後述するノード・エージェントと通信することにより行います。マスター構成情報はデプロイメント・マネージャーが保持し、各ノードにはマスター構成情報の一部のコピーが保存されます。マスター構成情報と各ノードの構成情報とは自動または手動によって同期がとられ、各ノードはローカルにコピーされた構成情報を参照して動作します。

ノード・エージェントは管理用のプロセスで、WAS ND 環境では各ノード上でデプロイメント・マネージャーと通信を行います。ノード・エージェントは以下のような機能を提供します。

- ファイル転送サービス
- 構成情報の同期サービス
- パフォーマンス・モニタリング

ノード・エージェントは、デプロイメント・マネージャーから送られた構成情報のコピーをローカルに保存し、マスター構成情報と同期を取ります。

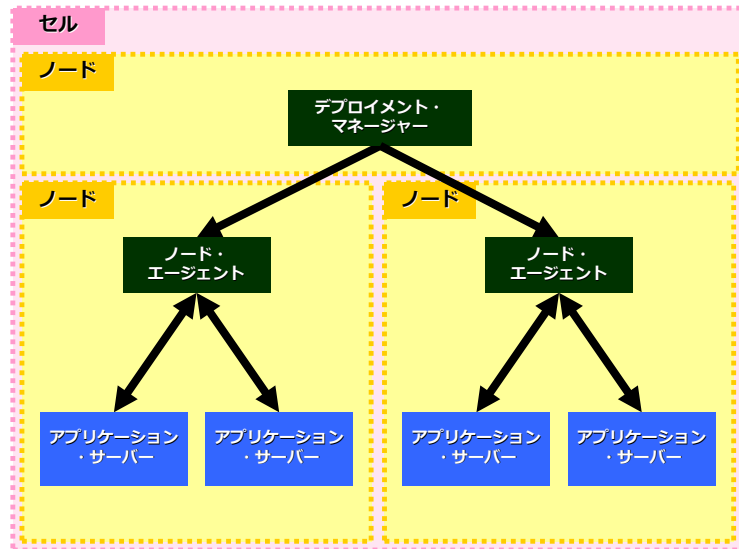


図 1-6 デプロイメント・マネージャーとノード・エージェントの関係

1.1.3.3 クラスター

クラスターとは、アプリケーション・サーバーの集まりのことを指し、クラスターに属する各アプリケーション・サーバーをメンバーといいます。クラスターを構成すると、アプリケーションのインストールはクラスターに対して行うだけで済み、それぞれのアプリケーション・サーバーに個別に行う必要がなくなります。

アプリケーション・サーバーは同一ノード上であっても、異なるノード上であっても、同一セル内にあれば、メンバーとしてクラスターに定義可能です (図 1-17 垂直クラスター、図 1-18 水平クラスター参照)。また、セルには複数のクラスターを定義できます。セル内の特定のノードを、クラスターのメンバーから除外することも可能です。

同じクラスター内のサーブレットや JSP ファイルは同一の URL を持ち、クラスターに対して行われるリクエストは、あらかじめ設定されたポリシーにしたがって、Web サーバー・プラグインによりクラスター・メンバーのいずれかひとつに割り振られます。また、障害時には Web サーバー・プラグインがそれを検知し、フェイル・オーバーされます。

クラスター・メンバーとなったアプリケーション・サーバーはクローン ID と呼ばれるユニークな ID を持ちます。Web サーバー・プラグインはリクエストの Cookie 情報などに含まれるクローン ID を参照することによって、セッション・アフィニティーを実現しています。

WAS V6.1 までは、クラスター構成を取るには WAS ND 以上の環境が必要で、Base ではサポートされませんでした。

WAS V7.0 からは、WAS Express/Base 環境でもクラスター構成を取ることがサポートされるようになりました。ただし手動による構成が必要でサーバー台数などに制約があります (詳細は「[1.1.1 パッケージング](#)」パッケージング を参照のこと)。

WAS ND V8.5 では、クラスターに関するさらなる機能追加が行われ、「クラスター」を作成する時には、以下の 2 種類のうちいずれかを選択できるようになりました。

1) 静的クラスター

ここまで説明してきた従来の「クラスター」を WAS ND V8.5 で作成すると、Intelligent Management 機能を利用可能な「静的クラスター」として作成されます。

2) 動的クラスター (WAS ND V8.5 New)

静的クラスターよりも更に柔軟なシステム運用管理機能を利用することができます。

動的クラスターで実現できることは後に解説します。

なお、WAS 管理コンソールでは、静的クラスターは WAS V8.0 以前と同様「クラスター」と表記されます。当ガイドで WAS ND V8.5 の静的クラスターを意図する場合には、「静的クラスター」と明記します。

動的クラスターとは

動的クラスターは、静的クラスターと同様、アプリケーションをインストール (デプロイ) するグループです。静的クラスターと異なる点は、動的に拡張可能な仮想化されたクラスターであること、また負荷状況や重要度に基づいてクラスター・メンバーのアプリケーション・サーバーを開始、停止することが可能な点です。

動的クラスターを作成する際、クラスター・メンバーを決定しますが、このクラスター・メンバーはアプリケーション・サーバーとして作成されます。

1.1.3.4 Edge Component

Edge Component は、負荷分散を行うための Load Balancer コンポーネントと、キャッシュや Proxy を行うための Caching Proxy コンポーネントを提供します。また、これ以外にも様々な機能を提供しており、WAS と組み合わせることで、可用性や、拡張性、パフォーマンスを向上させることが可能になります。しかしながら、WAS ND V6.1 から、Edge Components の次の機能が「非推奨 (Deprecated)」となっております。

2011 年 6 月 追記

V8.0 よりEdge ComponentsのLoad Balancer for IPv4 、Caching Proxyが非推奨（廃止予定）から安定化に変更されました。

WAS V8 Information Center - Stabilized features

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rmig_stabfeat.html

V7.0 まで廃止予定であったNAT・KCBR・Caching Proxy+CBR転送等の機能は今後も提供される予定ですが、システム設計に関しては今までと同様、Load Balancer for IPv4 and IPv6 のMAC転送方式を使用する構成が推奨されます。

非推奨機能

※MAC 転送は非推奨の対象外です。

- Load Balancer
- Dispatcher
- NAT/NAPT 転送
- KCBR 転送
- CBR
- Site Selector
- Cisco CSS Controller
- Nortel Alteon Controller
- ルール機能を使用したルーティング
- ※ルール機能は、Load Balancer 以外の代替策では対応できないものが多いため、現在機能改善要望を挙げています。いくつかのルール機能もしくはそれに代替する機能が今後残る予定です。
- 相互ハイ・アベイラビリティ機能
- GRE サポート
- 管理リモート
- 広域ロード・バランシング
- SNMP サブエージェント サポート
- UDP サポート
- Caching Proxy
-

Edge Component V8.5 が提供するコンポーネント

非推奨機能も含め、Edge Components V8.5 が提供しているコンポーネントは以下の通りです。

- Load Balancer IPv4
- Load Balancer IPv4 and IPv6
- Caching Proxy

※非推奨リストに記載された機能は、将来の製品リリースにおいて製品から除去する予定であることを示しています。IBM の基本的な方針としては、非推奨になったリリースから数えて、次の 2 メジャー・リリースまたは満 5 年 (いずれか長い方) は、その機能を製品から除去しません。例えば、WAS ND V5.0, V5.0.1, V5.0.2 で非推奨となった機能は、WAS ND V6.0 までは除去されません (WAS V5.0 から数えると、WAS V5.1、WAS V6.0 が 2 つのメジャー・リリース更新となります)。

Load Balancer とは

Load Balancer は Web サーバーへの HTTP リクエストの負荷分散機能を提供します。Load Balancer IPv4 では主に Dispatcher と CBR の 2 つのコンポーネントのいずれか、または両方を組み合わせて使用します。CBR は Edge Components のもうひとつの機能である Caching Proxy のプラグインとして動作します。Dispatcher と CBR を比べると、転送方式およびその実装方式から、Dispatcher の方が CBR よりもはるかに高い性能を示すため、通常は Dispatcher を使用します。ただし、この節で紹介するトポロジは特に断りがないかぎり、Dispatcher と CBR のいずれでも構成することが可能です。

表 1-2 サポートされる転送方式

	Load Balancer IPv4	Load Balancer IPv4 and IPv6
MAC 転送	○	○
NAT 転送	○	×
NAPT 転送	○	×
KCBR 転送	○	×

MAC 転送は、受信したリクエストの MAC アドレスを、割り振り先のサーバーの MAC アドレスへと変換します。この時、リクエストのヘッダーや宛先 IP などの変更しません。またレスポンスは割り振り先サーバーから直接クライアントに返ります。最もパフォーマンスが良い転送方式です。

NAT・NAPT 転送は、受信したリクエストの MAC アドレスおよび宛先 IP を割り振り先サーバーのものへと変換します (NAPT ではポートも変換します)。Load Balancer と割り振り先サーバーの間にルーター等がある場合等に使用します。

KCBR 転送は、CBR と同様にリクエストの URI を基に割り振りを行います。Cookie や URL などによる割り振り制御が可能になります。CBR と異なる点は、SSL を復号化できない点です。CBR は Caching Proxy のプラグインとして動作するので、Caching Proxy による SSL の復号化が可能となり、SSL 通信の場合でもリクエストの URI による割り振りが可能です。

Load Balancer IPv4 と Load Balancer IPv4 and IPv6

WAS ND では Load Balancer として Load Balancer IPv4 と Load Balancer IPv4 and IPv6 の 2 種類を提供しております。これらの違いを以下にまとめます。

表 1-3 Load Balancer IPv4 と Load Balancer IPv4 and IPv6 の相違

	Load Balancer IPv4	Load Balancer IPv4 and IPv6
Executor の実装レベル	カーネル・レベル	Java プロセス
非推奨機能の実装	有り	無し
IPv6 サポート	×	○

Load Balancer は負荷分散機能を提供するもので、以下のコンポーネントに分かれています。通常は Dispatcher と Content Based Routing (CBR) を使用します。また、Load Balancer IPv4 と Load Balancer IPv4 and IPv6 では含まれるコンポーネントが異なります。

表 1-4 実装コンポーネント

	Load Balancer IPv4	Load Balancer IPv4 and IPv6
Dispatcher	○	○
Content Based Routing	○	×
Site Selector	○	×
Cisco CCS Controller	○	×
Nortel Alteon Controller	○	×

Dispatcher と CBR は、クライアントからのリクエストを直接受信し、あらかじめ決められた負荷分散の設定に基づいて複数のサーバーから1つを選びだして転送するという方法で、負荷分散を行います。

Site Selector は DNS と連動して、クライアントから DNS 要求がきたときに、最も負荷の軽いサーバーの IP アドレスを DNS レスポンスとして返す、という方式で負荷分散を行います。

Cisco CSS Controller または Nortel Alteon Controller コンポーネントは、サーバー加重を生成するために使用できます。サーバー加重は、それぞれ、最適なサーバー選択、ロード最適化、および耐障害性のために、それぞれ、Cisco CSS Switch または Nortel Alteon Web Switch に送信されます。

このような負荷分散システムとサーバーの冗長化によって、負荷分散、可用性、スケーラビリティを実現することができます。Web サイトの設計を行うにあたり、障害設計や、パフォーマンス設計、運用管理などは避けて通れない課題であり、ほとんどのシステムでは、Edge Components の Load Balancer のような負荷分散装置を使用してシステムを構築する必要があります。

この節では Dispatcher と CBR のような負荷分散システムを使用した WAS システムを構築する Load Balancer コンポーネントについての概要を説明します。

クラスター (Edge Components)

Load Balancer の基本的な概念の1つにクラスターというものがあります。WAS ND にもクラスターと呼ばれる概念がありますので、混乱しないようにして下さい。Load Balancer の世界でのクラスターとは、複数のサーバーで負荷分散構成をとった時のそのサーバーのグループをクラスターと呼び、Load Balancer がその代表としてクラスター・アドレスを持ちます。クライアントで入力される URL のホスト名の部分に対応する IP アドレスがクラスター・アドレスになります。

アフィニティー機能

Web サーバーやアプリケーションサーバーは、HttpSession や SSL セッション ID などのステート情報を保持しているため、なるべく同一のクライアントからのリクエストは同じサーバーで処理した方がパフォーマンスがよくなります。このような要求に対応するために、Load Balancer には以下のアフィニティー機能を提供しています。

表 1-5 Affinity の方式

	Load Balancer IPv4	Load Balancer IPv4 and IPv6
IP Sticky	○	○
Passive Cookie	○	×
Active Cookie	○	×
SSL Session ID	○	×

IP Sticky は、クライアントの IP アドレスを使用して、クライアントを識別し、同じクライアントからのリクエストを同じサーバーに割り振ります。ただし、この方法ではプロキシなどがクライアントとの間にあるような環境では割り振りが偏る危険性があります。プロキシを通過してきたクライアントの IP アドレスは常にプロキシのアドレスと変換されてしまうためです。またモバイル端末が普及した現在では、同一のクライアントから時間をおいて異なった IP アドレスでリクエストが送信されることもしばしばあります。

Passive Cookie は、割り振り先サーバーで発行される Cookie を使用するアフィニティーです。リクエストに含まれる Cookie を解読するため、Dispatcher の KCBR 転送もしくは CBR を使用する必要があります。

Active Cookie は、CBR が発行する Cookie を使用するアフィニティーです。CBR のみ使用できません。

SSL Session ID は、SSL 通信を行う時に割り振られる SSL の Session ID を使用するアフィニティーです。ただし、クライアントが Internet Explorer を使用している場合、定期的にセッションを張りなおすため正しく動作しない可能性があります。

以上のように、Load Balancer で使用できるアフィニティー機能は完全では無かったり非推奨のコンポーネントでしか使用できなかったりと制限があります。アプリケーションの機能として完全なセッション維持が必要な場合には、Web サーバー・プラグインによるアフィニティー機能や、アプリケーション・サーバー間での HttpSession 共有機能を利用するようにします。

障害検知

Load Balancer は定期的にサーバーの状態をモニタリングすることによりサーバーの障害を検知します。サーバーへのモニタリングを行うコンポーネントをアドバイザーといいます。アドバイザーはデフォルトでは 7 秒間隔で、予め指定されたリクエストをサーバーに送信し、否定応答を受信したり、タイムアウトしたりすると、サーバー障害と判断して、そのサーバーへの割り振りを停止します。サーバーへのポーリングは定期的に行われますので、サーバーが再起動すると、アドバイザーはその応答を受信して、ふたたびそのサーバーへの割り振りを開始します。アドバイザーがサーバーに送るリクエストは指定可能なので、WAS のサーブレットを指定できます。サーブレットを指定すると、Web サーバーだけでなく、アプリケーション・サーバーの障害まで検知できます。さらに、指定されたサーブレットでデータベースのチェックまで行い、データベースが使用できない場合は否定応答を返すことにより、データベースの障害まで Load Balancer で検知可能です。ただし、アプリケーション・サーバーをクラスター構成していて、Web サーバー・プラグインで複数のアプリケーション・サーバーに割り振りを行っている場合には、Load Balancer でアプリケーション・サーバーまでの障害検知を行うことはできません。これは Web サーバー・プラグインがアドバイザーの監視リクエストを障害の起きたサーバーには送らず、正常なサーバーに転送するため、Load Balancer から見るとどちらの Web サーバーに監視リクエストを送信しても、正常なレスポ

スがかえってくるためです。このような構成ではアプリケーション・サーバーのダウン検知は Load Balancer ではなく Web サーバー・プラグインで行うことになります。

Load Balancer はブラウザからのリクエストを最初に受け取るコンポーネントのため、Single Point Of Failure (単一障害点) とならないようにするために、バックアップ (スタンバイ) 機を用意して、Load Balancer 自体で HA (High Availability) 構成をとることが可能です。次節以降で紹介しているトポロジーではこの部分は省略していますが、実際にサーバー環境をデザインする際には、二重化を忘れないようにして下さい。Caching Proxy/CBR では HA 機能はサポートされていないのでご注意ください。(Caching Proxy/CBR の可用性を確保するために、通常は複数の Caching Proxy/CBR を配置すると共に、Caching Proxy/CBR の前段に Dispatcher を配置して、Caching Proxy/CBR をクラスター化する手法をとります。)

また、アクセスが集中して全てのアプリケーション・サーバーに高負荷がかかり、一定時間内にユーザーリクエストに対する応答が返せなくなる、という状況に陥ることも考えられます。このような状況を未然に防ぐ目的で、Load Balancer で接続数制限を行い、一定数以上のリクエストがきた場合には、混雑中であることを表示するサーバーに転送するような設定ができます。また、同様に、サーバー保守などの目的で、サーバーを停止した場合に、停止中を知らせるメッセージを表示するサーバーに振り分けたりするような制御も可能です。このようにユーザーにサービスできない旨をお知らせするサーバーを、Sorry Server と呼ぶ事もあります。

1.1.3.5 ジョブ・マネージャー

ジョブ・マネージャーを利用することによって、複数 Base/Express 環境や複数 ND セルに対して、運用、管理などの処理 (ジョブ) を一箇所にキューイングして、非同期に実行することが可能になります。ジョブ・マネージャーから実行できるジョブは以下の通りです。

- アプリケーション・サーバーの始動・停止・作成・削除
- クラスター (メンバー) の始動・停止・作成・削除
- アプリケーションのインストール・アンインストール・更新・始動・停止
- ファイルの配布・収集・除去
- wsadmin スクリプトの実行
- プロキシ・サーバーの作成・削除
- プロパティの構成
- インベントリー : キューイングされているジョブの表示
- 状況 : ジョブの実行結果の表示

これらのジョブに対して、利用可能な機能は以下の通りです。

- 指定した時刻にジョブを実行
- 特定の時間を越えたタイミングでの無効化
- 特定の時間間隔での繰り返し実行
- ジョブの完了通知メールの送信

ジョブ・マネージャーを利用するためには、WAS ND のセル内のデプロイメント・マネージャー・ノードか、WAS Base/Express の管理エージェント管理対象下のノードを登録する必要があります。

構成変更などの、ジョブ・マネージャーから行えない運用管理については従来どおり、ND セル環境の場合にはデプロイメント・マネージャーで、Base/Express 環境では個々のアプリケーション・サーバーに対する管理サービスで行う必要があります。

1.1.3.6 WebSphere プロキシ・サーバー

WebSphere プロキシ・サーバーとは、WAS ND V6.0.2 から登場した Java ベースの新しいプロキシ・サーバーです。Caching Proxy とは異なり、WAS の管理コンソール上での管理が可能です。WebSphere プロキシ・サーバーは以下の機能を持っています。

- ワークロード管理
： ODC (オンデマンド構成) と呼ばれる機能により、同一セル内のサーバーに対して追加構成をすることなくプロキシを構成できます。
- Cache 機能
： 静的コンテンツ、動的コンテンツ、SSL コンテンツのキャッシュが可能です。
- SSL 通信・復号機能
： SSL での通信と復号が可能です。クライアント - WebSphere プロキシ・サーバー間を SSL 通信した後、それ以降は通常の HTTP 通信を行うことも、復号して URL のチェックを行った後、再び暗号化を行い、後方へ SSL 通信を行うことも可能です。
- Web サーバー・プラグインとの連携
： Web サーバー・プラグインからのリクエストを WebSphere プロキシ・サーバーに割り振ることが可能です。
- セッション・アフィニティーの維持
： WebSphere プロキシ・サーバーにて URI や Cookie による Affinity 制御が可能です。

1.1.3.7 セキュア・プロキシ・サーバー

セキュア・プロキシ・サーバーとは、WAS V7.0 から登場した、セキュリティを強化した WebSphere プロキシ・サーバーです。具体的に強化された部分として、必要最低限のポートの開放や、デジタル署名された JAR のみのロード、非特権ユーザーとしての well known ポートへの接続などが挙げられます。

WebSphere プロキシ・サーバーは WAS ND V8.5 に同梱されていますが、セキュア・プロキシ・サーバーは「DMZ Secure Proxy Server for IBM WebSphere Application Server」に含まれています。IBM Installation Manager (IIM) にてインストールを行います。また、セキュア・プロキシ・サーバーを導入しても管理コンソールは導入されませんので、設定の変更を行うためには `wsadmin` コマンド (ローカル、Flexible Management 経由) もしくは、別ノードで Configuration-Only プロファイルを作成し、そのノード上の管理コンソールから設定した Proxy の構成情報を、`wsadmin` コマンドにて `import/export` する必要があります。もっとも安全な管理方法は、ローカルの `wsadmin` からのみ変更管理を行う方法です。このように管理すると、外部のリスニング・ポートを開く必要がないためです。

1.1.3.8 オンデマンド・ルーター (ODR)

ODR は、Java プロキシ上に実装されたインテリジェントなプロキシ・サーバーであり、プロトコルとして HTTP と SIP (Session Initiation Protocol) をサポートしています。ODR の内部では、オートノミック要求フロー・マネージャー及び動的ワークロード・マネージャーが動作しています。

ODR は、以下の動作をします。

- 重要度の低いアプリケーションに対するリクエストを一時的にキューに入れ、より重要度の高いアプリケーションに対するリクエストをより高速に処理できるようにする
- バックエンドのアプリケーション・サーバーを過負荷から保護する
- 各サーバーの使用状況から各サーバーへのトラフィックの量を調整する
- 管理者からの指示により特定のサーバーへの割り振りを停止する (保守モード)

オートノミック要求フロー・マネージャー (ARFM)

オートノミック要求フロー・マネージャーは、オンデマンド・ルーター内で、以下の動作を行います。

- 定義した条件に応じてリクエストにサービス・ポリシーを適用する
- サービス・ポリシー（重要度、目標応答時間）ごとにリクエストをキューイングする
- リクエストを処理する際に使用したリソースを計測する
- 新たなリクエストに対して使用するリソースを計測し、リソースが不足した場合はキューで保持（または拒否）する

動的ワークロード・マネージャー (DWLM)

動的ワークロード・マネージャーは、オンデマンド・ルーター内で、以下の動作を行います。

- サーバーの負荷状況に応じて重み付けを動的に決定する
- 重み付けに応じてリクエストのルーティングをラウンドロビンで行う

1.2 トポロジー

前節では、WAS の各コンポーネントについて解説しました。実際のシステムでは、これらのコンポーネントを実装するにあたって、スケーラビリティや耐障害性といった可用性、セッション管理、障害検知、運用、パフォーマンスなどを考慮してデザインする必要があります。これら様々な要件に対応できるように、WAS はさまざまなトポロジーを取ることができます。本章では、そうした観点から考慮点を説明し、WAS の代表的なトポロジーをとりあげ、その特徴を説明します。

1.2.1 WAS Base と WAS ND の選択

ここでは、WAS Base もしくは WAS ND のどちらを選択すべきかについて、要件別に説明します。

まず、大きく異なる点として、クラスター構成の使用の可否があります。WAS ND ではクラスター構成を使用できるため、負荷分散と可用性を実現でき、更に構成情報を一元管理することができます。WAS Base においては、マシンとアプリケーション・サーバーを複数台用意し、負荷分散装置を別途用意しデータベースで HttpSession を共有することで、Web コンテナの可用性と負荷分散を実現することができます。ですが、EJB コンテナやトランザクション・マネージャー、メッセージング・エンジンの負荷分散やフェイル・オーバーを実現することは出来ません。また、WAS ND クラスター機能のその他のメリットとして、ロールアウト・アップデート機能があります。これを用いることで、クラスター・メンバーのアプリケーションを時間差で更新し、クライアントにエラーを返す事なくアプリケーションを更新することが可能になります。

また、WAS ND ではデプロイメント・マネージャーを使用できるため、各ノードへのアプリケーション配布に整合性が保証され、セル内のアプリケーションやアプリケーション・サーバーの状態をすべて管理することができます。WAS Base の場合は、各サーバー単位で管理することになるため、同じアプリケーションを複数のノードにインストールする必要があり、構成情報なども手動で変更する必要があります。

ジョブ・マネージャーの使用可否も異なる点の 1 つです。WAS Base ではジョブ・マネージャーを利用することができません。WAS ND では利用可能ですので、ジョブ・マネージャーを利用することで、運用・管理などのジョブを一箇所にキューイングし、非同期に実行することが可能になります。詳細は 1.1.3.5 ジョブ・マネージャーをご参照下さい。

また、WAS ND では、データベースを利用するか、メモリー間のセッション複製機能により、セッション情報を異なるアプリケーション・サーバーで直接共有する事が可能です。WAS Base の場合データベースを利用したセッション情報共有のみが可能です。更に、WAS Base では Intelligent Management 機能を使用することができません。

まとめると、可用性や負荷分散が重要になり、構成情報の管理やアプリケーションの整合性を取る為に運用負荷をかけたくない場合には、WAS ND を選択することになります。以下は、WAS Base と WAS ND を使用したときの違いを表にしたものです。

表 1-6 WAS Base と WAS ND の比較

	WAS Base 1 台	WAS Base 複数台	WAS ND
可用性(冗長構成)	×	○	○
負荷分散	—	○	○
クラスター構成	×	×	○
Web コンテナのフェイル・オーバー	—	○	○
EJB コンテナのフェイル・オーバー	—	×	○
コンポーネント、構成ファイル、アプリケーションの一元管理	○	×	○
運用・管理のジョブのキューイングと非同期実行	○	○	○
SW ライセンスコスト	○	○	△
セッション・パーススタンス (同一 JVM での情報共有)	○	○	○
セッション・パーススタンス (複数 JVM 間での情報共有)	×	△ 5 サーバー まで DB 方式の み	○
ロールアウト・アップデート (クラスター内アプリケーション更新を順次実行)	×	×	○
トランザクション・マネージャーや メッセージング・エンジンの高可用構成 (HA マネージャー)	×	×	○
アプリケーション・サーバーのモニター (ノード・エージェント)	×	×	○
Intelligent Management 機能群	×	×	○

1.2.2 WAS Base トポロジー設計の考慮点

WAS Base におけるトポロジー設計における主な考慮点としては、以下の点があります。

- Web サーバーをアプリケーション・サーバーと同一マシンに配置するか否か
- Web コンテナと EJB コンテナを同一マシンに配置するか否か
- 複数アプリケーションを用いる際に、アプリケーション・サーバーを単一にするか複数にするか
- Web サーバーを単一にするか複数にするか
- 管理エージェントを配置するか否か
- セッション・データベースの配置について

これらについて、以下で説明していきます。

1.2.2.1 Web サーバーの配置

WAS では、Web サーバーの稼動するマシンとアプリケーション・サーバーの稼動するマシンを同一にすることも、分離することも可能です。同一筐体に配置した構成をローカル Web サーバー構成、別筐体に配置した構成をリモート Web サーバー構成と呼びます。前者では WAS の筐体がクライアントから直接リクエストを受ける事になりますが、後者では Web サーバーを DMZ に配置することで WAS のセキュリティを向上させることができます。また、IHS で SSL 復号化を行う場合など、IHS のマシンリソースが多く必要な時に、WAS とのリソースを分離する事ができます。ただし、その場合、必要なマシン数が増加します。以下は、ローカル Web サーバーとリモート Web サーバーの比較表です。

表 1-7 ローカル Web サーバーとリモート Web サーバー

	ローカル Web サーバー	リモート Web サーバー
セキュリティ	△ ゾーン分けが不可能なため、同一のセキュリティ・レベルとなる。	○ Web サーバーを DMZ に配置し、セキュリティ・レベルを分離可能。
パフォーマンス	△ リソースを共有する。処理におけるボトルネックの検出も困難。	○ リソースが分離される。Web サーバーでの SSL の暗号化・復号化や静的コンテンツの処理時に有利。処理のボトルネックの検出も容易。
拡張性	△ 単体のマシン性能を向上させる必要がある。	○ Web サーバーとアプリケーション・サーバーを分離して拡張可能。
保守容易性	○ マシン台数が少ない分、保守が容易となる。	△ マシン台数が増加する分、運用負荷が増加する。また、プラグイン構成ファイルの伝搬が必要となる。
コスト	○ マシン台数が少ない分、HW コストも少ない。ただし、拡張時のコストは高くなる可能性がある。	△ マシン台数の分、HW コストが増加する (リモート IHS の導入に追加 SW ライセンスの必要はない)。

1.2.2.2 Web コンテナと EJB コンテナの分離配置

WAS はサーブレットの実行を行うアプリケーション・サーバー (Web コンテナ) と、EJB の実行を行うアプリケーション・サーバー (EJB コンテナ) を別プロセスとして起動することが可能です。EJB コンテナのアプリケーション・サーバーを Web コンテナのアプリケーション・サーバーと分離することにより、拡張を柔軟に行えます。逆に、EJB コンテナと Web コンテナを同一のアプリケーション・サーバーで稼動させると、データがネットワーク上を流れることがなく、メモリー内部での処理になるため、パフォーマンス的には有利です。ただ、この場合は EJB 呼び出しの引数がコピー渡しではなく参照渡しになるので、呼び出し先で引数のオブジェクトを変更する操作には注意が必要となります。

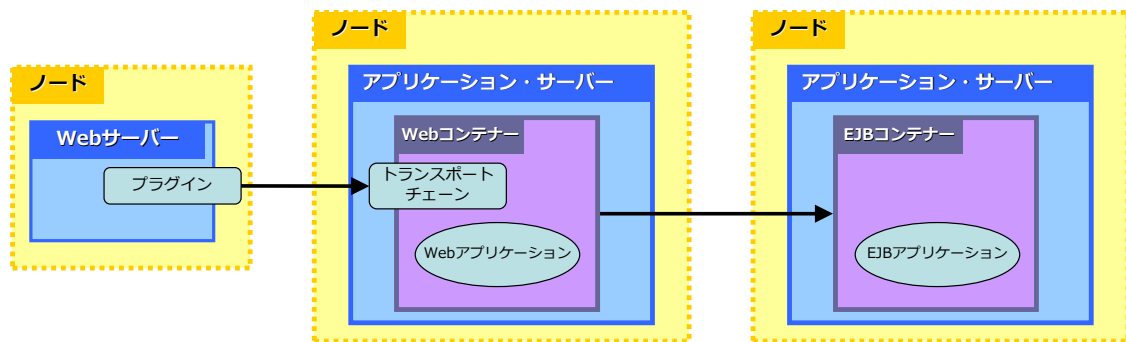


図 1-7 EJB コンテナの切り離し

1.2.2.3 複数アプリケーション・サーバーの配置

アプリケーションが複数ある場合には、アプリケーション単位でアプリケーション・サーバーを起動することができます (図 1-11 複数アプリケーション・サーバー/ローカル Web サーバー参照)。こうすることで、ログやサーバーの設定を分けての管理、停止時間が異なる場合の運用などを行え、アプリケーション・サーバーの障害による影響も最小限に抑える事ができます。その反面、アプリケーション・サーバーを増やした分だけメモリーを多く消費し、運用負荷も増えることになります。またこれらの中間形態の配置、つまりいくつかはグループ分けしたアプリケーションをグループ単位でアプリケーション・サーバーに分割する、複数アプリケーション・サーバー・複数アプリケーションという配置もよく行われています。

表 1-8 単一アプリケーション・サーバーと複数アプリケーション・サーバーの比較

	単一アプリケーション・サーバー	複数アプリケーション・サーバー (アプリケーションごと)
運用柔軟性	複数のアプリケーションで共通のパフォーマンス・パラメーターなどを使用。 サーバーの起動・停止・システムログなども相互に影響する。	アプリケーションごとに、アプリケーション・サーバーのパフォーマンス・パラメーターなどを設定可能。 サーバーの起動・停止・システムログも分離できる。
プロセス障害	別のアプリケーションにも影響	他のアプリケーションには影響しない
リソース使用量	アプリケーション・サーバー1 つ分だけ必要	アプリケーション・サーバーの数だけ増加
運用負荷	アプリケーション・サーバー1 つ分だけの管理	アプリケーション・サーバーの数だけ増加

1.2.2.4 複数 Web サーバーの起動

アプリケーションと同様に、Web サーバー (IHS) のプロセスも複数起動することができます。こうすることで、MaxClients などのパラメータ設定や、サービスの起動・停止を分離することができます。ただし、WAS Base でこの構成を取るには、Web サーバー・プラグインを手動で編集する必要があります。また同一 IP アドレスかつ同一のポートで複数の Web サーバーを起動することもできません。

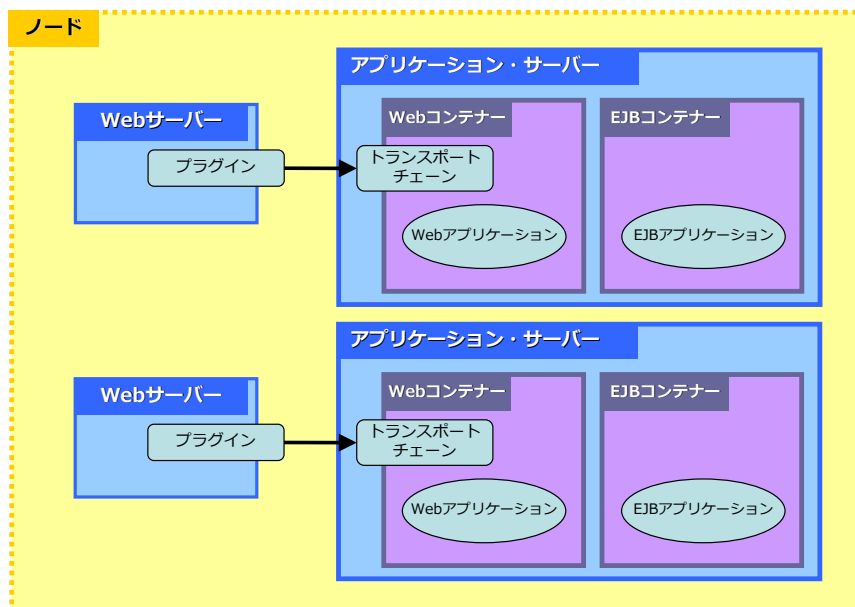


図 1-8 複数 Web サーバー

表 1-9 単一 Web サーバーと複数 Web サーバーの比較

	単一 Web サーバー	複数 Web サーバー
運用柔軟性	パフォーマンス・パラメーターなどの設定は共通の設定となる。 Web サーバー停止・起動が両方のアプリケーション・サーバーに影響する	Web サーバー毎にパフォーマンス・パラメーターなどの設定を分ける事が可能。 それぞれ異なるプラグイン・ファイルを読み込むことが可能。 停止・起動も柔軟に行うことが可能
リソース使用量	Web サーバー1 つ分	Web サーバーの数だけ増加 複数の Listen ポート、または IP アドレスが必要
運用負荷	Web サーバー1 つ分	Web サーバーの数だけ増加

1.2.2.5 管理エージェントの配置

管理エージェントを配置することで、複数アプリケーション・サーバーの集中管理が可能になります。またアプリケーション・サーバーと管理コンソールを実行するプロセスが分離されるため、管理コンソール経由のアプリケーション・サーバーの起動および停止ができるようになります。ある程度のリソースを使用する内部アプリケーションである管理コンソールをなくすることができるため、アプリケーション・サーバーのリソースをユーザー・アプリケーションで占有することができるというメリットもあります。これらのメリットを活用したい場合には、管理エージェントを構成します。

このほか管理エージェントが必要となる構成としては、フレキシブル・マネジメント構成を取ってジョブ・マネージャーから統合管理をする必要がある場合 (図 1-5 フレキシブル・マネジメント構成) があります。ただ将来的にそういった構成を取る計画がある場合でも、管理エージェントを予め配置しておく必要はありません。必要になった際に、管理エージェントを作り、アプリケーション・サーバーを登録することができます。

1.2.3 WAS Base 構成例

ここでは、1-2-1 項で紹介した基本トポロジーを用いた WAS Base での構成例を紹介いたします。

1.2.3.1 単一マシン構成

WAS の実行に必要なすべてのコンポーネントを 1 台のマシン上にインストールした構成で、もっとも基本的な構成です。この構成では Web サーバーとアプリケーション・サーバーは同じマシンで稼動しています。コストは一番安いですが、全てのコンポーネントが Single Point Of Failure となります。1 台の構成ですので、アフィニティーを気にする必要はありません。

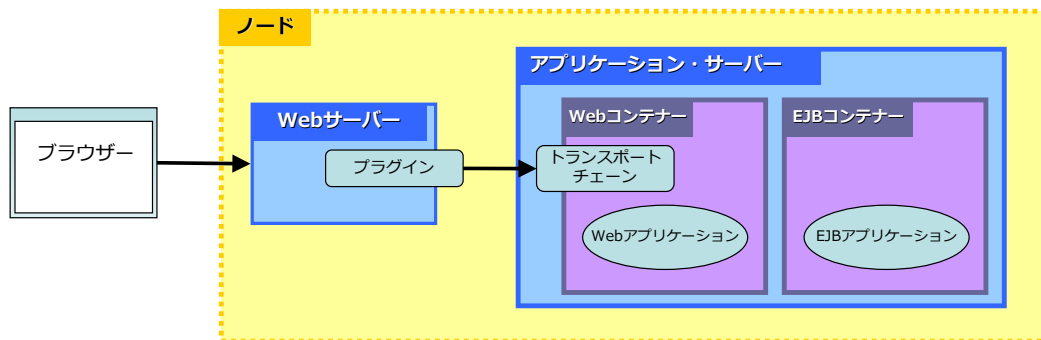


図 1-9 単一マシン構成

1.2.3.2 単一アプリケーション・サーバー/リモート Web サーバー

1.2.2.1項で説明した、Web サーバーとアプリケーション・サーバーの稼働するマシンを分離した構成です。この構成では、各サーバーの負荷を分散させることができます。また、Web サーバーを DMZ に配置し、アプリケーション・サーバーとデータベースをファイアウォールで保護された内部ネットワークに配置することにより、セキュリティ面でより安全性の高い構成をとることができます。反面、別々のマシンで構成することにより、マシン障害の発生する可能性のある個所が増えることにもなるため、冗長構成とする場合には、WAS ND を使用したクラスター構成にする必要があります。

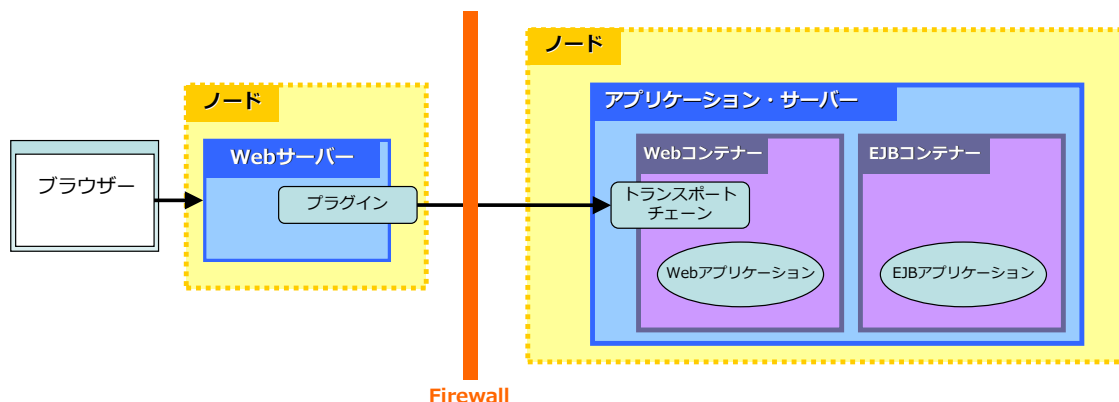


図 1-10 単一アプリケーション・サーバー/リモート Web サーバー

1.2.3.3 複数アプリケーション・サーバー/ローカル Web サーバー

1.2.2.3項で説明した、1台のマシン上で複数の異なるアプリケーション・サーバーを同時に実行する構成です。

次図は、Web サーバーと複数のアプリケーション・サーバーをローカル Web サーバーで 1 台のマシンで構成したものです。それぞれのアプリケーション・サーバー上では異なるアプリケーションが実行されています。それぞれのアプリケーションは、異なる URL で識別され、ユーザーからの要求された URL に含まれるホスト名、またはアプリケーション・パスにもとづいて、Web サーバー・プラグインで適切なアプリケーション・サーバーに転送されます。ただし、負荷分散が行われるわけではありません。同一ノード上でそれぞれのアプリケーション・サーバーは異なるポート番号で Web サーバーからのリクエストを受け付けます。

CPU やメモリー・リソースを十分に搭載したマシンでは、複数のアプリケーション・サーバー・プロセスを同時に実行することによって、CPU やメモリーの稼働率を上げることが期待できます。ただし、CPU やメモリー・リソースが充分でないマシンでは、複数プロセスの同時実行を制御するためのオーバーヘッドにより、全体としてシステムの負荷が大きくなり、かえってアプリケーションの実行効率が低下する場合があります。

この構成では、あるアプリケーション・サーバー・プロセスがダウンした場合は、そのアプリケーション・サーバーに属するアプリケーションは使用できなくなります。

注) WAS Base で 1 台のマシン上に複数アプリケーション・サーバーを作成する場合は、複数のプロファイルを作成します。管理コンソールを 1 つに統合したい場合、管理エージェント管理下におく構成とします。詳しくは以下のリンク先を参照下さい。

WAS V8.5 Information Center 「アプリケーション・サーバーの作成」

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.base.doc/ae/txml_createateserver.html

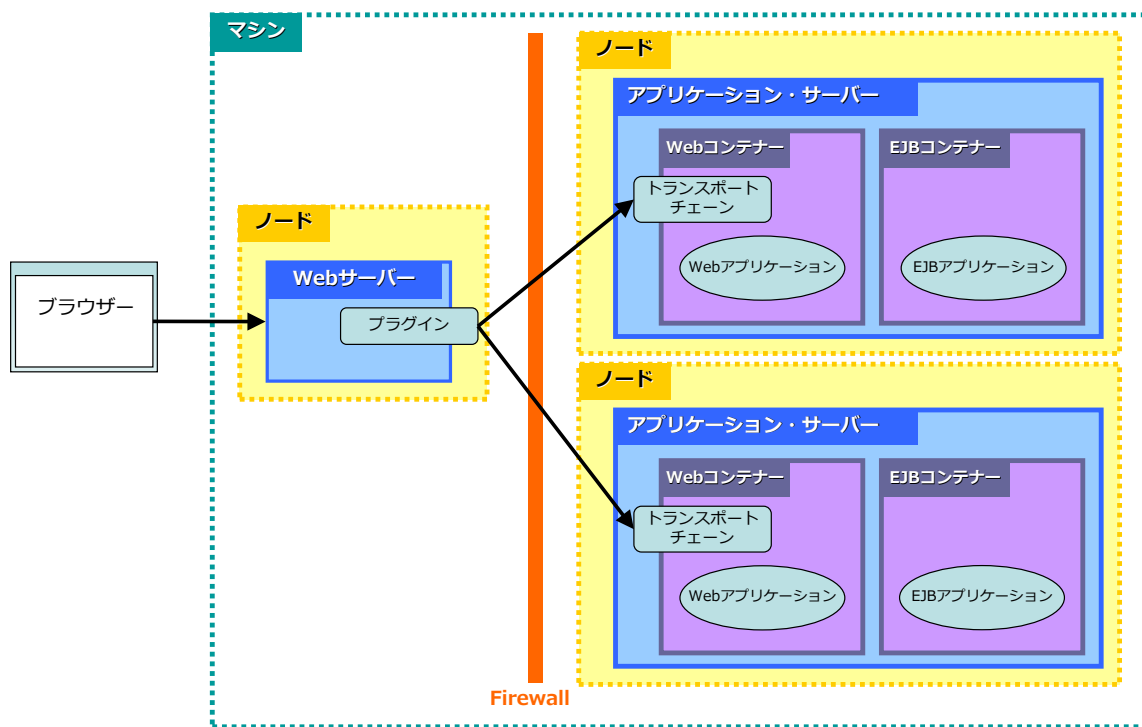


図 1-11 複数アプリケーション・サーバー/ローカル Web サーバー

1.2.3.4 WAS Base シンプル・クラスター

WAS V7.0 から、WAS Base エディションでも、ND 同様のクラスター構成を取ることができるようになりました。これにより、より安価に、可用性の高いシステムを構成することができます。

前項で説明した「1.2.3.3 複数アプリケーション・サーバー/ローカル Web サーバー」構成とは以下の点が異なります。

- 同一のアプリケーションを、複数アプリケーション・サーバー(シンプル・クラスター)にデプロイ可能
- 同一のアプリケーションに対するリクエストを、クラスター内で負荷分散される
- セッション・パーシスタンスが利用可能
- 複数アプリケーション・サーバー宛の割り振り定義を、プラグイン構成ファイルにマージするツールが提供されている

一方で、WAS ND エディションと異なり、以下の制約・考慮点があります。

- シンプル・クラスターのメンバーとできるアプリケーション・サーバー (JVM) 数は 25JVM 以内
- セッション・パーシスタンスを構成できるアプリケーション・サーバー数は 5JVM 以内
- Web コンテナ・カスタム・プロパティ「HttpSessionCloneId」を作成し、そのアプリケーション・サーバーのクローン ID をアプリケーションサーバーごとに明示的に設定する必要がある
- セッション・パーシスタンスのデータストアとして利用可能となるのは、データベースのみ。Mem-to-Mem は ND に限定される。
- クラスタリングされる対象は Web コンテナのみであり、その他のクラスタリング (EJB コンテナのクラスタリングなど) は行われない。

- (2013 年 10 月追加情報) 上述の通り WAS V8.5 までは、WAS Express や Base の場合は、Web サーバー・プラグインから WAS にロード・バランシング/フェイルオーバー可能な台数(※WAS プロファイル数)がライセンス上、制限されています。WAS V8.5.5 においてこの制限が廃止となり、ND 同様に、台数に制限なく負荷分散できるようになります。

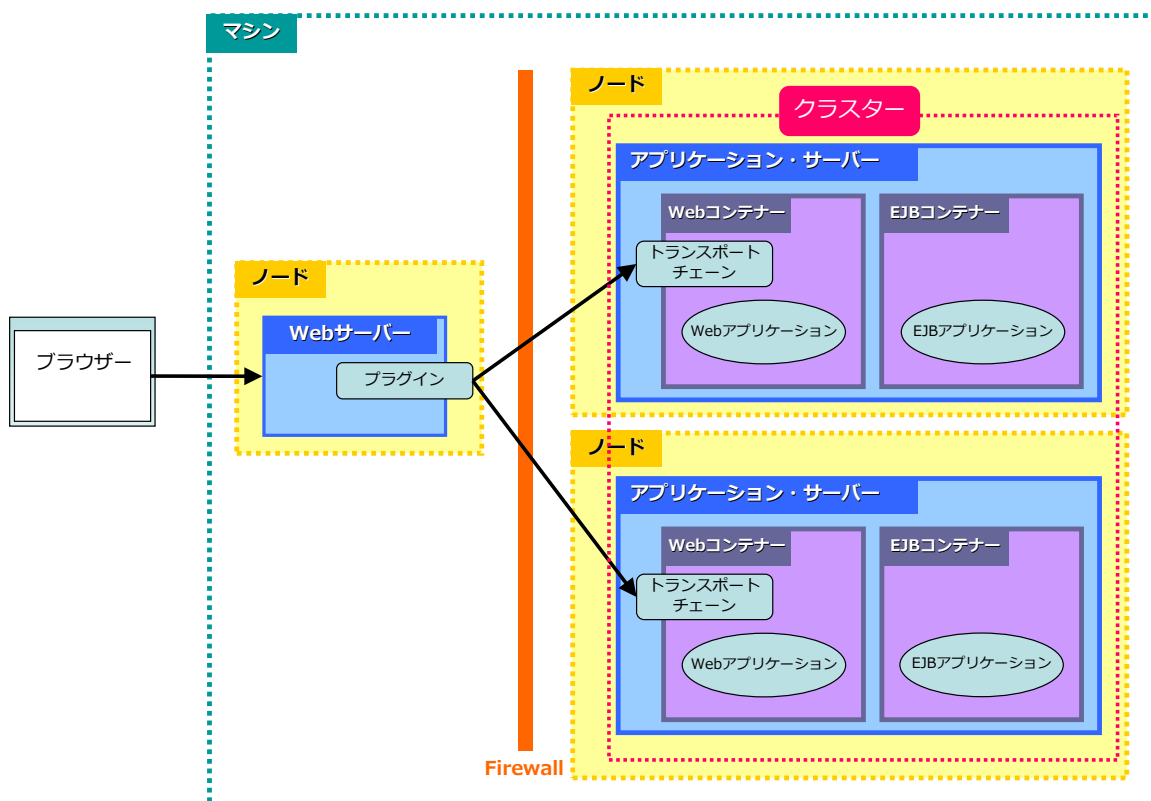


図 1-12 複数アプリケーション・サーバー/ローカル Web サーバー

WAS Base シンプル・クラスターの構成方法については、下記の記事を参照してください。

developerWorks

「WAS 小ワザ集: 第 24 回: WAS Base/Express でシンプル・クラスターを構成する方法」

http://www.ibm.com/developerworks/jp/websphere/library/was/was_tips/24.html

1.2.3.5 セッション・データベース

WAS Base でセッション・データベースを持った構成です。この構成ではアプリケーション・サーバーのプロセス障害や、マシン障害時からの復旧後、セッションを維持できます。

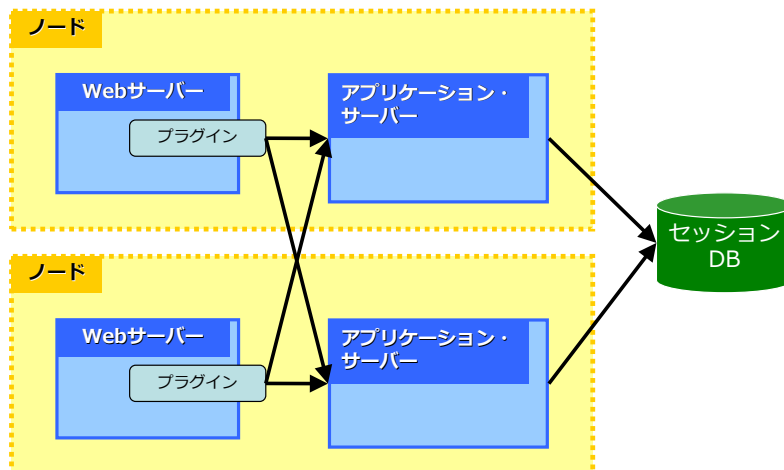


図 1-13 WAS Base とセッション・データベース

WAS Base シンプル・クラスターにおけるセッション DB の構成方法については、下記の記事を参照してください。

developerWorks「WAS 小ワザ集: 第 24 回: WAS Base/Express でシンプル・クラスターを構成する方法」

http://www.ibm.com/developerworks/jp/websphere/library/was/was_tips/24.html

1.2.4 WAS ND トポロジー設計の考慮点

WAS ND におけるトポロジー設計における主な考慮点は、以下の通りです。

- クラスター構成を取る際に垂直クラスターにするか、水平クラスターにするか
- デプロイメント・マネージャーをアプリケーション・サーバーと同一マシンに配置するか否か
- ジョブ・マネージャーを配置するか否か
- セッション・パーシスタンスを行う際に、セッション・データベースで行うか、メモリー間複製で行うか
- Web サーバーとアプリケーション・サーバーの対応を 1:1 にするか 1:N にするか
- Edge Component について
- WebSphere プロキシ・サーバーの配置について
- セキュア・プロキシ・サーバーの配置について
- ODR の配置について

このうち、Edge Components、WebSphere プロキシ・サーバー、セキュア・プロキシ・サーバーについては、第 1 部第 8 章をご参照下さい。また、WAS Base にも含まれているコンポーネントのトポロジー設計については、1.2.2項を参照下さい。ここでは、WAS ND に特有のトポロジー設計について説明していきます。

1.2.4.1 デプロイメント・マネージャーの配置の考慮点

WAS ND では、複数のサーバーをセルで管理する際には、デプロイメント・マネージャーをアプリケーション・サーバーと別筐体に配置することも、同一筐体に配置することも可能です。

別筐体に配置する構成では、メモリーなどのマシン負荷が少なくなりますが、ソフトウェア・ライセンスをその分多く購入する必要があります。

一方、同一筐体に配置する構成では、片方の筐体に負荷がかかりますが、ソフトウェア・ライセンスは前者と比較して少なく済みます。こちらの構成にする場合は、デプロイメント・マネージャーの稼動するマシンにメモリーを増設し、CPU 負荷がアプリケーションに影響しないようにし、デプロイメント・マネージャーを停止するなどの運用面での工夫をする必要があります。

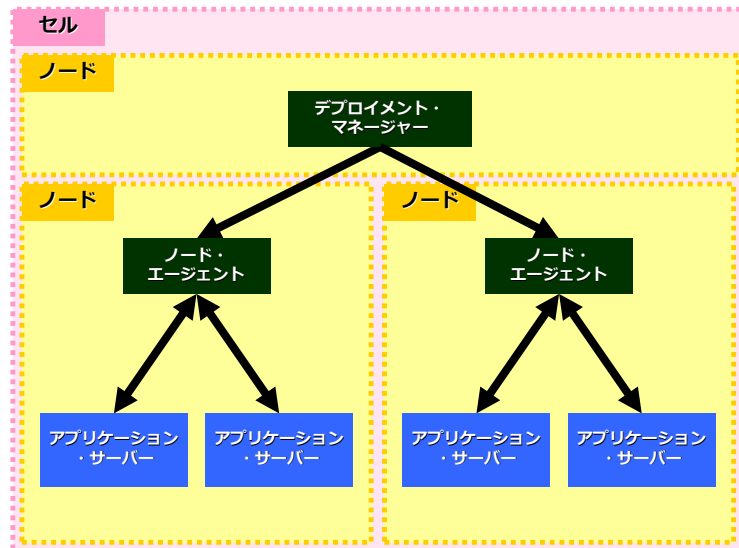


図 1-14 デプロイメント・マネージャーの配置 (別マシン)

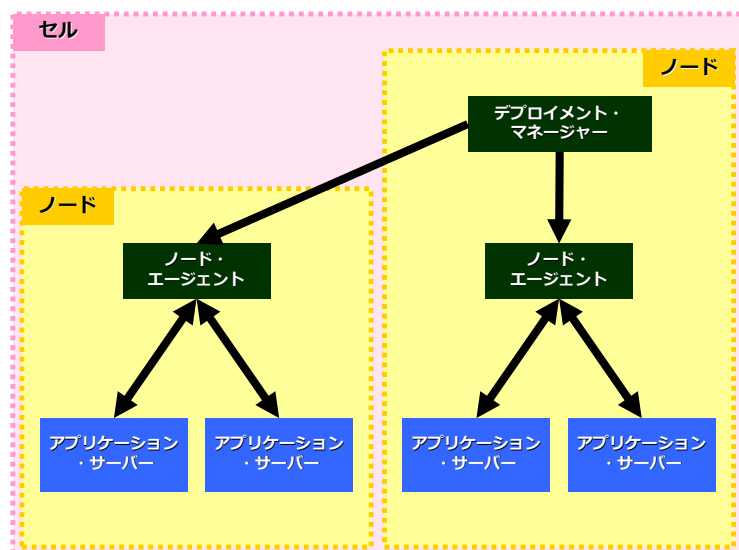


図 1-15 デプロイメント・マネージャーの配置 (アプリケーション・サーバー共存)

以下の表は、デプロイメント・マネージャーをアプリケーション・サーバーと別筐体に配置した場合と、同一筐体に配置した場合を比較したものです。

表 1-10 デプロイメント・マネージャーとアプリケーション・サーバーの配置

	同一筐体	別筐体
追加マシン	必要なし	必要あり
ソフトウェア・ライセンス	必要なし	必要あり
システム・リソース	デプロイメント・マネージャーの 分増加	影響なし

1.2.4.2 高可用性デプロイメント・マネージャー構成 (HA DM)

WAS ND V8.5 Intelligent Management 環境では、アクティブ-スタンバイによる デプロイメント・マネージャーの高可用性構成 (High Availability 構成、HA 構成) を取ることができます。高可用デプロイメント・マネージャー、DM HA などとも呼ばれる機能です。

通常、デプロイメント・マネージャーは管理作業を行わない間は停止していても構いませんので、多くの場合、DM を高可用性構成とする必要はありません。

しかし、アプリケーションのデプロイやサーバーの監視を含む作業の自動化を行っている場合や、WAS ND V8.5 Intelligent Management 環境における視覚化データ・サービスのログを中断させたくない場合など、常に起動していることが重要となることがあります。このような要件があるお客様環境では、HA DM 構成を検討します。

それぞれのデプロイメント・マネージャーは、異なる物理または論理マシン上に構成します。

デプロイメント・マネージャーを異なる物理マシンで稼働させる場合、OS の種類は同じとすることが推奨されますが、必須ではありません。また、デプロイメント・マネージャーが管理するリポジトリファイル群は、HA を構成するデプロイメント・マネージャー間で共有する必要があるため、NFS などの共有ファイル・システムが必須となります。

HA DM 構成は、2 つ以上のデプロイメント・マネージャーにて構成します。

高可用性の新しいデプロイメント・マネージャー・コンポーネントは、各デプロイメント・マネージャー内で実行され、どのデプロイメント・マネージャーをアクティブなものとして選択するかを制御します。

アクティブとなるのは 1 つのデプロイメント・マネージャーのみで、アクティブなデプロイメント・マネージャーのノードに障害が発生すると、1 つのスタンバイのデプロイメント・マネージャーをアクティブ・モードに動的に切り替え、スタンバイのデプロイメント・マネージャーが処理を引き継ぎ、新規のアクティブ・デプロイメント・マネージャーとなります。

管理コンソールや wsadmin など、管理クライアントからどのデプロイメント・マネージャーに接続するべきかは、オンデマンド・ルーター (ODR) により制御されます。どのデプロイメント・マネージャーのインスタンスがアクティブであるかを認識し、すべての管理のための通信をアクティブのデプロイメント・マネージャーのプロセスに接続します。デプロイメント・マネージャーの引き継ぎが発生した場合、ODR が新規アクティブのデプロイメント・マネージャーの選択を自動的に認識し、管理のリクエストを新規アクティブのデプロイメント・マネージャーに転送するため、作業内容は失われません。ただし、デプロイメント・マ

ネージャーへのフェイル・オーバーが完了するまでの間、デプロイメント・マネージャーを使用できなくなります。

なお、HA DM 機能は、JMX SOAP コネクターの使用のみをサポートします。JMX RMI コネクターはこの構成ではサポートされません。

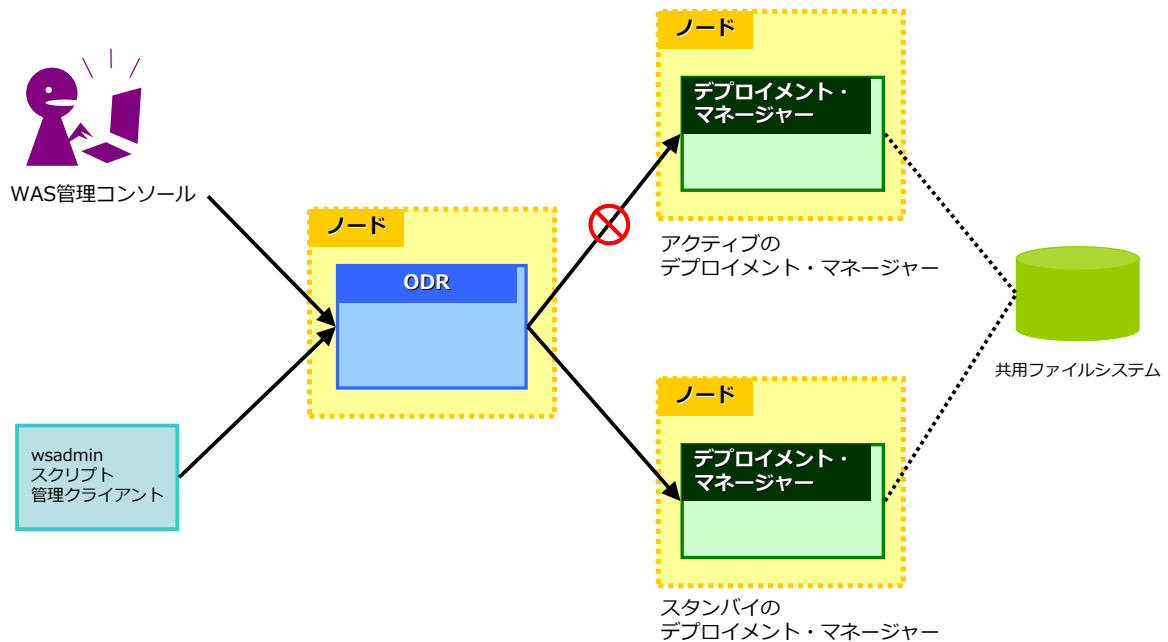


図 1-16 高可用性デプロイメント・マネージャー構成

1.2.4.3 ジョブ・マネージャーの配置の考慮点

ジョブ管理が必要な場合は、いくつか選択肢があります。

Tivoli Workload Scheduler(TWS)などのジョブスケジューラーソフトウェアがある環境であれば、そういった製品にてジョブ管理を行う選択肢がまず考えられます。

一方で WAS でも、ジョブ・マネージャーが提供されていて、TWS などが無い環境や、複数セル環境を一括管理したい場合に利用することができます。

ジョブ・マネージャーを配置することで、複数 Base/Express 環境と複数 ND セルに対して、運用、管理などの処理（ジョブ）を一箇所にキューイングして、非同期に実行することが可能になります。また、ND セルや Base/Express 環境が複数あって、その一括管理を行いたい場合にも利用することができます。将来的にそういった構成を取る場合でも、管理エージェントと同様、ジョブ・マネージャーを予め配置しておく必要はありません。必要になった際に、ジョブ・マネージャーを作り、管理対象を登録することができます。

1.2.4.4 垂直クラスター構成と水平クラスター構成

WAS (Express/Base/ND) のクラスター構成には、垂直クラスターと水平クラスターがあります。

垂直クラスター

同一ノード上で複数のアプリケーション・サーバー・プロセスを実行することによって、プロセス障害時の耐障害性が向上し、またメモリー・リソースを効率的に利用できます。

この構成では、Web サーバー・プラグインが重み付きラウンドロビンまたはランダムによって割り振り先を決定します。また既存のリクエスト(セッション ID をもったリクエスト)については、そこに含まれる Clone ID をもとに、前回処理したのと同じアプリケーション・サーバーへ割り振りがおこなわれます (セッション・アフィニティ)。

またプラグインはクラスター・メンバーの障害・再起動を自動的に検知して、実行可能なアプリケーション・サーバーにクライアントからのリクエストを割り振ります。Web コンテナでリクエストの処理中にクラスター・メンバーに障害が発生した場合、Web サーバー・プラグインはそのリクエストを別のクラスター・メンバーに送りなおすことにより、アプリケーションのフェイル・オーバーを実現しています (この再送信は POST リクエストについては制限することも可能です)。クラスター・メンバーが障害から復旧した場合、Web サーバー・プラグインは一定時間間隔で、そのクラスター・メンバーを再度割り振り先に組み込むので、そのタイミングでメンバーとしてリクエストが転送されるようになります。

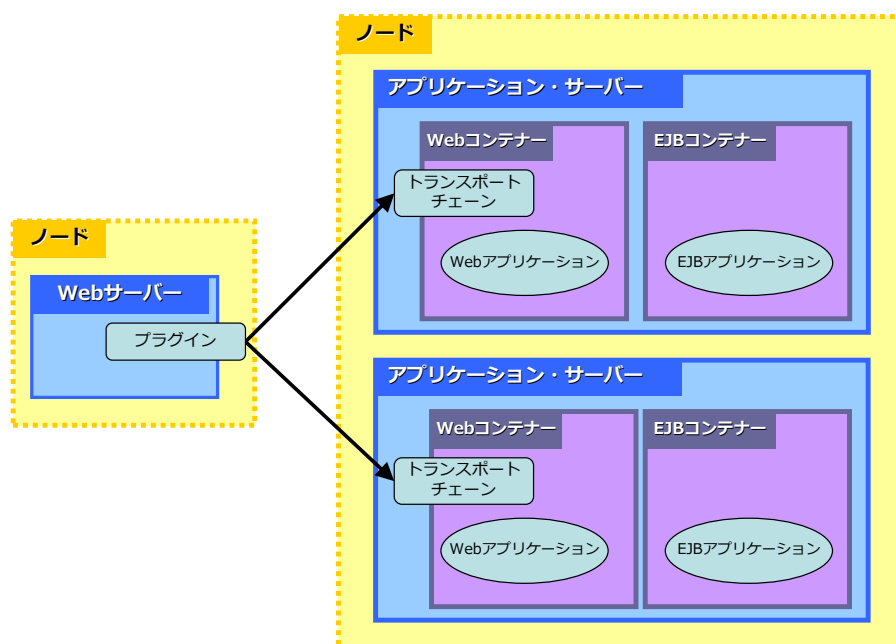


図 1-17 垂直クラスター

水平クラスター

複数のクラスター・メンバーをそれぞれ別のノード上で実行することが可能です。それぞれのクラスター・メンバーとなるアプリケーション・サーバーの構成情報は、ノード・エージェントによって、デプロイメント・マネージャーから各ノードに配布されます。この場合も、各クラスター・メンバーに対するリクエストの割り振りは、Web サーバー・プラグインによって、セッション・アフィニティーと重み付きラウンドロビンまたはランダムを組み合わせることで実行されます。障害検知とフェイル・オーバーも同様におこなわれます。この構成では、WAS プロセスの障害だけでなくアプリケーション・サーバー・マシンのハードウェア障害やオペレーティング・システムの障害にも対応することが可能です。

一般的に 2 倍の性能のマシンを用意するより同じ性能のマシンを 2 台用意した方が安価にすむため、WAS では水平クラスターリングが広く使われています。

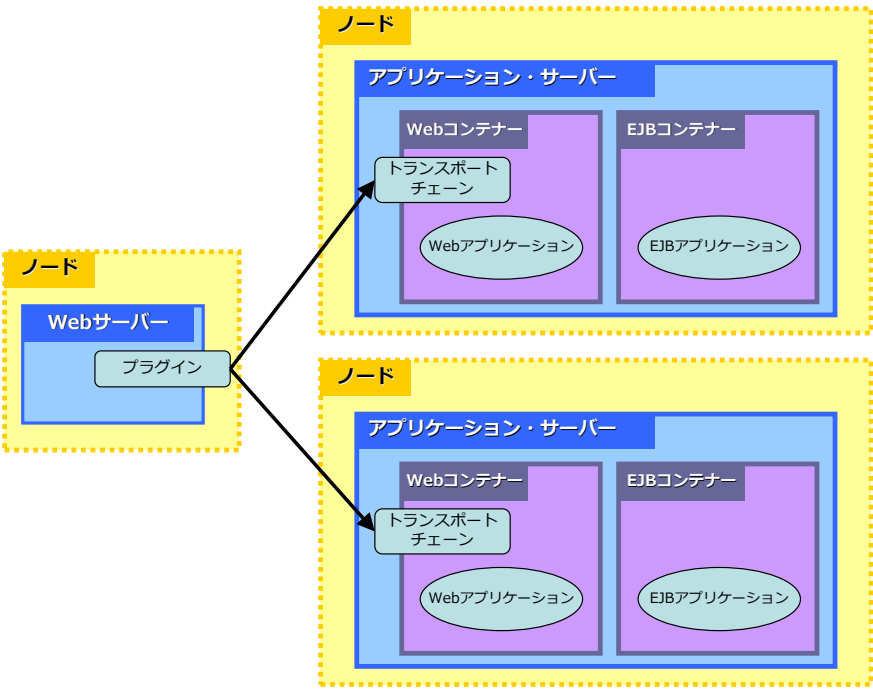


図 1-18 水平クラスター

以下の表は垂直クラスターと水平クラスターの比較をまとめたものです。垂直クラスターと水平クラスターは併せて構成することが可能です。

表 1-11 垂直クラスターと水平クラスターの比較

	垂直クラスター	水平クラスター
負荷分散	○	○
フェイル・オーバー（プロセス障害）	○	○
フェイル・オーバー	×	○

バー (筐体障害)		
リソースの有効活用	○ リソースに余裕がある場合、有効活用可能	△ 複数台を管理する必要があるため、台数分のライセンス、サーバー・リソースが必要となる
拡張性	△ 作成できるクラスター・メンバーの数は対象マシンのリソースに応じて限界がある	○ マシン追加により、垂直クラスターよりも柔軟な拡張が可能

1.2.4.5 静的クラスター構成と動的クラスター構成

WAS ND V8.5 でクラスター構成を取る場合には、静的クラスターと動的クラスターいずれかの選択肢を取ることができます。

静的クラスター

WAS V7.0 までの「クラスター」と基本的には同義で、予め作成しておいたクラスター・メンバーに対して、静的な重み付け計算を元に、静的なワークロード分散を行います。

トポロジーとしては、前述の水平クラスター、垂直クラスターいずれの構成を取することもできます。

WAS V8.5 の静的クラスターでは、動的ワークロード管理やヘルス・アクションなどの Intelligent Management の機能を利用することもできます。Base や Express でクラスターを利用する場合は、Intelligent Management の機能を利用することはできません。

動的クラスター

動的クラスターとは、重みおよびワークロード管理を使用して、クラスター・メンバーから収集されたパフォーマンス情報に基づき、動的にクラスター・メンバーのワークロードの バランスを取るサーバー・クラスターです。

動的クラスターを作成すると、動的に拡張可能な仮想化されたクラスターとして構成され、また負荷状況や重要度に基づいてクラスター・メンバー (アプリケーション・サーバー) を自動的に追加作成・起動・停止するなど、より柔軟で最適化された管理が行われます。

動的クラスターには、自動・監視・手動の 3 種類のモードがあります。

自動

クラスター・メンバーは自動的に作成・起動・停止されます。

監視

クラスター・メンバーは手動で作成・起動・停止します。

管理者に対し一定時間にサーバーを始動または停止するように指示するランタイム・タスクが生成されます。

手動

クラスター・メンバーを手動で作成・起動・停止します。

トポロジーとしては静的クラスター同様、水平クラスター、垂直クラスター、いずれの構成も取ることができます。クラスター・メンバーを自動的に追加作成するような動的クラスターを作成する際には、同一ノードで複数のクラスター・メンバーを稼働させることを許容する構成（垂直クラスター）とすることも、同一ノードで複数のクラスター・メンバーを稼働させることを許容しない構成（水平クラスター）とすることもできます。

動的クラスターは、負荷状況や重要度に基づいて動的にクラスター構成を拡張することが可能な、仮想化されたクラスター環境です。動的クラスターのメリットを生かすには、動的にクラスター・メンバーが追加定義・開始されたり、停止されている状況を把握しつつ、各クラスター・メンバーの処理状況に応じて適切な割合で負荷分散を行うモジュールが必要です。WAS ND では、オンデマンド・ルーター（ODR）がこの役割を果たします。

ODR は、後段のサーバー各々の使用状況を確認した上で各サーバーへのトラフィック量を調整したり、重要度の高いアプリケーションに対するリクエストをより高速で処理できるような各種の制御を行います。（詳細は[1.1.3.8 オンデマンド・ルーター（ODR）]の章を参照）

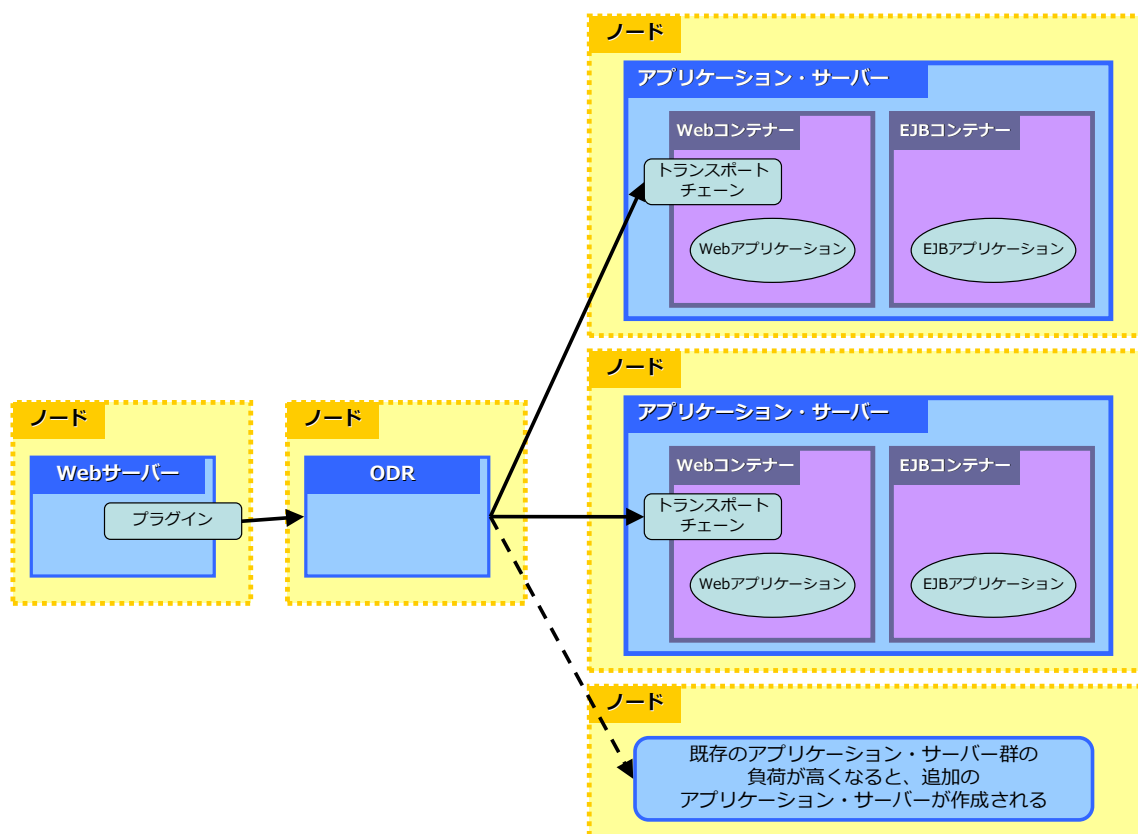


図 1-19 動的クラスター

動的クラスター分離 (アプリケーション分離)

「図 1-19 動的クラスター」では、1 つの WAS ノード上に 2 つの動的クラスターのインスタンスが稼動する構成となっていますが、動的クラスターの「分離設定」の指定を変更することで、1 つのクラスター・メンバーが他のクラスター・メンバーと同居しない構成とすることもできます。

動的クラスターの分離機能を使用することで、同一セル内に複数あるアプリケーションのうち、特定のアプリケーションを他のアプリケーションの実行環境（ノード）から分離することができます。

動的クラスターを利用すると、負荷状況に応じてアプリケーションは WAS ND Intelligent Management の機能によって自動的に配置、移動されます。逆に言うと管理者が明示的に、ノード・グループ内のどのクラスターでどのアプリケーションを稼働させるかを指定することはできません。そのため予期せぬアプリケーションの組み合わせが一つのノードで稼働して、互いに干渉するようなことが起こります。

特定のアプリケーションについて、他のアプリケーションの稼働状況に影響を受けないようにするために、明示的にノードを分けたい場合に、動的クラスターの「分離」を検討します。

アプリケーション遅延スタート

アプリケーションのリクエスト頻度が少なく長時間アクセスがない場合は、アプリケーション遅延スタートという機能を使用できます。この機能を有効にすると、アプリケーションをサーバーごとに停止しておき、リクエストが来た時にアプリケーション・サーバー・インスタンスを起動させることができます。リソースに対してクラスター数（サーバー・インスタンス数）が多い場合に、リソースを有効利用するために使用されます。

非アクティブなアプリケーションへのリクエストが来るとサーバーが開始されますが、その間クライアントには 503 の HTTP エラーが返ってしまうため、ODR のカスタム・ページを使用して、サーバーが起動中であり、まもなくリクエストが実行されるというメッセージをクライアントに返すように構成してください。

静的クラスター と 動的クラスターの違い

動的クラスターは、静的クラスターよりも柔軟な管理機能を提供します。

表 1-12 動的クラスターと静的クラスターの機能比較

機能	動的クラスター	静的クラスター
動的ワークロード管理	○ (デフォルト 有効)	○ (デフォルト 無効)
エディション・コントロール	○	○
ヘルス・ポリシー	○	○
ヘルス・アクション	○	○
保守モード	○	○
メンバーシップ	ルールによってクラスター・メンバーを自動的に定義するか、もしくは手動でクラスター・メンバーを定義します。	静的クラスターに、クラスター・メンバーを手動で定義します。
クラスター管理	動作ポリシーに応じて自動的に始動・停止を行うことが可能です。監視モードを指定し、ランタイム・タ	管理者がアプリケーション・サーバー (クラスター・メンバー) を定義し、アプリケーション・サーバーの始動・停

	スクに操作の指示を上げることや、手動での始動・停止を行うことも可能です。	止は手動で実施します。
クラスター・テンプレート	動的クラスターを定義するとき、アプリケーション・サーバー・インスタンスのアプリケーション・サーバー・テンプレートを定義できます。 動的クラスターを定義したら、動的クラスター・サーバー・テンプレートを使用して、クラスター・メンバーのプロパティを編集することができます。その変更は、すべてのアプリケーション・サーバー・インスタンスに反映されます。	静的クラスターを定義するとき、作成するすべてのアプリケーション・サーバー・インスタンスを配置するアプリケーション・サーバー・テンプレートを選択できます。しかし、インスタンス作成後にテンプレートを変更しても、インスタンスは変更されません。

既存の静的クラスターを動的クラスターに変換することもできます。

動的クラスターの作成時に、オプションで「既存の静的クラスターからの変換」で既存の静的クラスターを指定することで動的クラスターに変換することができます。ただし考慮点として、静的クラスターから動的クラスターに変換するとクラスター・テンプレートを使用することはできません。

動的ワークロード管理

動的ワークロード管理は、従来の静的なワークロード管理（ラウンドロビン、重み付けによる割り振り）とは異なり、関連コンポーネントから取得した情報を元に動的な重み付けやリクエストのルーティングや、アフィニティー、フェイル・オーバーが行われます。

エディション・コントロール

アプリケーションの複数のバージョンを並行して WAS で管理し、それらを切り替えることにより短時間でのアプリケーション・リリースを実現する機能です。

ここで言う「エディション」はアプリケーションを WAS 上で複数バージョン保持するためのバージョン情報であり、1.0, 1.1..., のように数字でバージョン情報を付与することもできますし、英字や記号を用いた文字列でエディション名を定義することもできます。

エディション・コントロール機能を使用可能にした環境では、新しい「エディション」のアプリケーションを予め WAS 上にデプロイしておき、アプリケーションを切り替えたいタイミングがきたら現行の「エディション」から新しい「エディション」に切り替えることで、リリースが完了します。

ショッピング・サイトなど、アプリケーションのリリースをおこなうための計画停止時間が十分確保できないシステムにおけるアプリケーションのリリースに有用な機能です。また新バージョンのアプリケーションに異常があった際に、古いバージョンに切り戻すためにも使用できます。

セッション情報を使用しているサイトの場合には、セッション・パーシスタンスを構成し、セッション情報を共有できる構成としておく必要があります。

ヘルス・ポリシー / ヘルス・アクション

期間、要求タイムアウト超過、応答時間超過、メモリー使用量、GC 実行時間 などの条件を、ヘルス・ポリシーとして定義しておき、ヘルス・ポリシーを満たしていないことが検出された場合には、その問題を修正するためのアクションを実行させることができます。

保守モード

保守モードはノードまたはサーバーを稼働させたまま、ODR からそれらに対するリクエストの割り振りを停止する機能です。保守モードを使用すれば、プロセスを稼働させたまま問題判別を行ったり、ダンプ・ファイルなどの資料取得を行ったり、必要なチューニングを施したり、といったメンテナンス作業を行うことができます。保守モードはノード・レベル、アプリケーション・サーバー・レベル、ODR レベルで設定可能です。

また、縮退の際にはアフィニティの管理が重要なポイントとなりますが、保守モード機能を使用するとアフィニティを維持しながら縮退する事も可能です。通常の「保守モード」を選択すると、既存のリクエストの割り振りを続けたまま、新規のリクエストのみ割り振りを停止します。「保守モード・アフィニティの中断」を選択すると、既存のリクエストを含めた全てのリクエストの割り振りを停止します。

1.2.4.6 セッション・パーシスタンス・トポロジー

WAS では、アプリケーション・サーバーの分散環境で、HttpSession に格納されたセッション情報を複数のサーバー間で共有・永続化するセッション・パーシスタンスの機能をサポートしています。セッション情報をセッション・データベースとしてリモート・データベース上で提供する形態とアプリケーション・サーバー間のメモリ上でセッション情報のコピーを行う形態の 2 つの形態がサポートされています。

以下で、アプリケーション・サーバー間でセッション情報を共有するためのトポロジーについて見ていきます。

セッション・データベース

WAS ND での基本的なセッション・データベースの共有形態はクラスター内のアプリケーション・サーバーで共用する構成です。クラスター構成された複数のアプリケーション・サーバーでデータベースを使用したセッションの共有は、WAS ND の標準的な形態になります。以下のような構成の場合、片方のアプリケーション・サーバーがダウンした場合でも、もう一台のアプリケーション・サーバーでセッションを引き継ぐ事が可能となります。

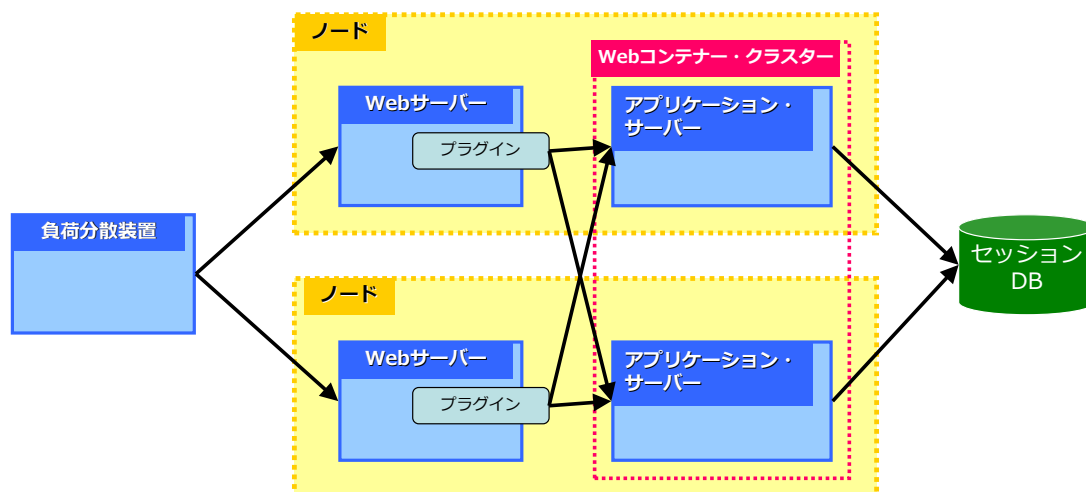


図 1-20 WAS ND (クラスター) と共用セッション・データベース

メモリー間複製

各アプリケーション・サーバー間でメモリー複製によって共有する形態です。

メモリー間複製のトポロジーとしては、ピアツーピアとクライアント・サーバーの2つがサポートされています。このタイプのセッション・パーシスタンスを行うには、セッション情報の複製を行う各アプリケーション・サーバーが WAS ND のクラスターとして構成されている必要があります。

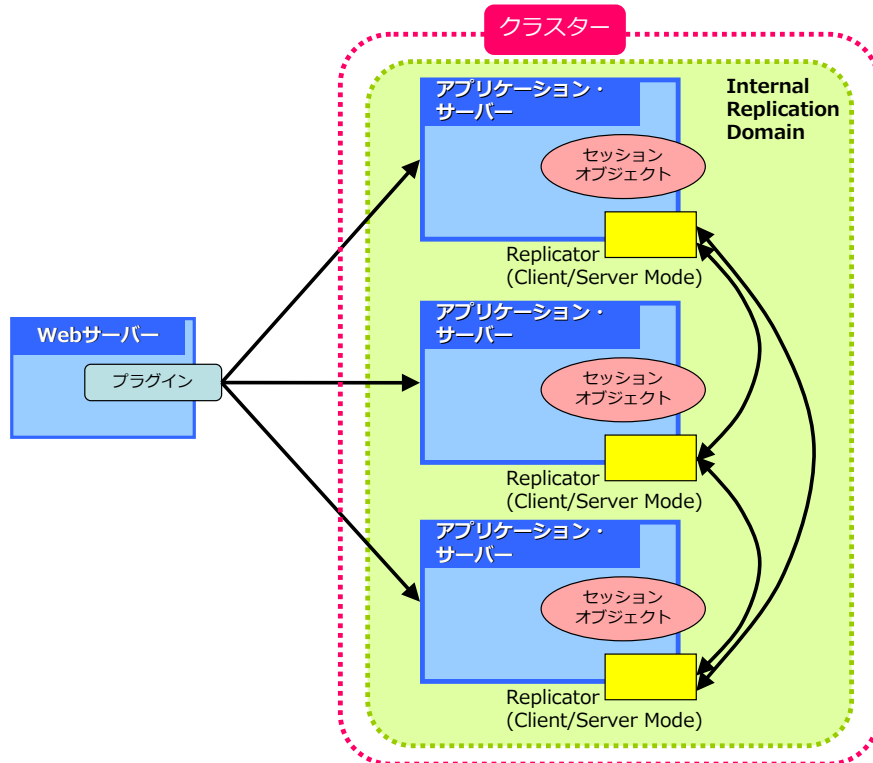


図 1-21 メモリー間複製トポロジー (1) ピアツーピア

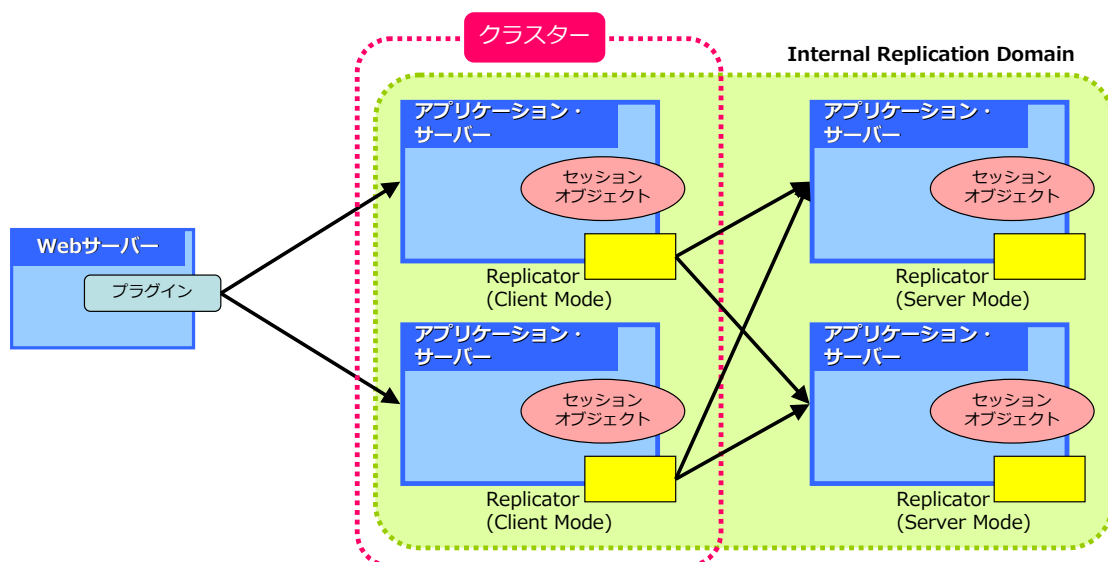


図 1-22 メモリー間複製トポロジー (2) クライアント・サーバー

その他のセッションパーシスタンス・ストア

先に紹介しているセッション・データベース、メモリー間複製以外にも、インメモリー・データ・グリッド製品である IBM WebSphere eXtreme Scale (WXS) や、そのハードウェア・アプライアンス機器である IBM WebSphere DataPower XC10 (XC10) を、WAS のセッション・オブジェクトの格納先として利用することもできます。

WXS は既存のハードウェアをインメモリー・データ・グリッドとして利用することのできるソフトウェアです。WXS の JVM をセッションパーシスタンス・ストアとして利用する仕組みとなっており、WAS ND の JVM をセッションパーシスタンス・ストアとして利用するメモリー間複製と似た仕組みといえます。

WXS を使う構成は、WAS ND のメモリー間複製に比べて以下のアドバンテージがあります。

セッション数やセッション・オブジェクトのサイズの増大に対応できる直線的なスケーラビリティを備えています。

セッション・オブジェクトをグリッドの中で暗黙的に複製および管理することができます。

ゾーンをサポートすることにより、地理的に分散した複数のデータ・センター（およびセル）にわたってセッション・オブジェクトを保管することができ、メモリー間複製の単一セル内という複製の制約を克服しています。

一貫性のある複製を保証しないメモリー間複製と比べ、サービスの品質と複製の信頼性が向上しています。WXS は、より信頼性の高い複製メカニズムを提供します。

XC10 は、大容量キャッシュを搭載したアプライアンス・マシンであり、WAS セッション・オブジェクトを XC10 上に保持する仕組みとすることで、従来のセッション・データベースやメモリー間複製に比べ、応答時間の高速化が実現されます。

詳細は、以下製品紹介ページ・製品マニュアル等の参照情報をご参照ください。

1) IBM WebSphere eXtreme Scale

<http://www.ibm.com/software/jp/websphere/apprtransaction/extremescale/>

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp>

http://www.ibm.com/developerworks/jp/websphere/library/wxd/wxs_http_session/

2) IBM WebSphere DataPower XC10

<http://www.ibm.com/software/jp/websphere/apprtransaction/xc10/>

<http://pic.dhe.ibm.com/infocenter/wdpxc/v2r1/topic/com.ibm.websphere.datapower.xc.doc/welcome/ic-homepage.html>

1.2.4.7 Web サーバー配置の 1:N 構成の検討

1:N 構成について説明します。これは、1 つの Web サーバーからリクエストを割り振るアプリケーション・サーバーの数の比を意味しています。1:N 構成とは、Web サーバーとアプリケーション・サーバー数の比が 1:N (複数) である構成を意味します (図 1-23 負荷分散装置とクラスター (ローカル Web サーバー 1:N 構成参照)。以下では、1:N 構成について、障害検知の特徴を説明します。

1:N 構成での障害検知

IHS にて受け付けられたリクエストをアプリケーション・サーバーへ割り振る役割は、Web サーバー・プラグインが担っています。Web サーバー・プラグインは、アプリケーション・サーバーのプロセスダウンを検知する事はできますが、アプリケーションのダウンやサブシステムのダウンまでは検知できません。仮に負荷分散装置からアプリケーション・サーバーまで障害検知を行おうとしても、負荷分散装置からの検知リクエストは Web サーバー・プラグインによってダウンしていないアプリケーション・サーバーへと割り振られてしまいます。また、Web サーバー・プラグインの障害検知では、Socket への接続で障害を検知しますので、アプリケーションのハングのような状態を障害として検知することができません (Web サーバー・プラグインの構成ファイルである plugin-cfg.xml の ServerExtendedHandshake エlement を利用した場合に限って検知することは可能です)。また、Web サーバー・プラグインでは、障害を検知しても、それを通知することができません。したがって、1:N の構成の場合は、アプリケーションの正常稼動を確認する方法を別途検討することが推奨されます (アプリケーション・サーバーを直接監視した場合はアプリケーション・サーバーの障害を検知することが可能です)。

尚、1:N の構成では、Cookie または URL 再書き込みを使用して、Web サーバー・プラグインによるセッション維持が可能ですので、負荷分散装置によるアフィニティは完全である必要はありません。

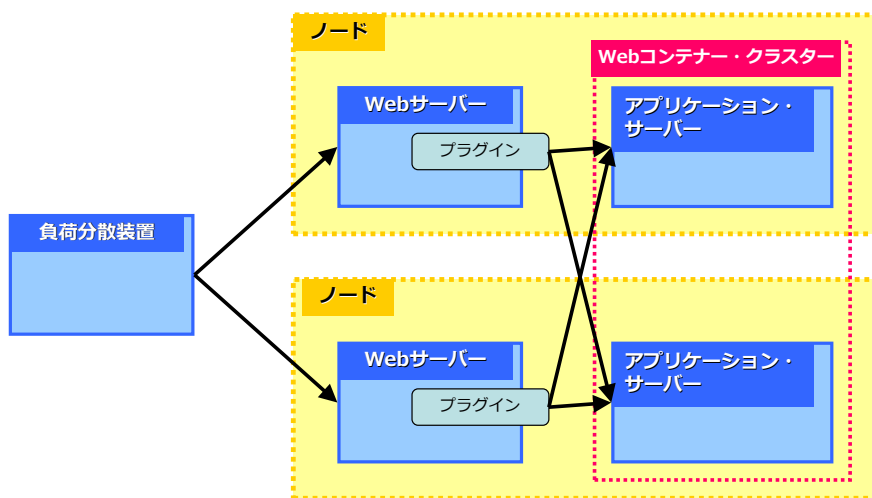


図 1-23 負荷分散装置とクラスター (ローカル Web サーバー 1:N 構成)

表 1-13 1:N 構成のまとめ

	1:N 構成
負荷分散装置からの障害検知の範囲	負荷分散装置で Web サーバーまで検知可能 (アプリケーション・サーバーを直接監視することでアプリケーション・サーバーの障害を検知することができる) ※Web サーバー・プラグインはアプリケーション・サーバーまで検知可能だがサブシステムの検知はできない
アフィニティー	Web サーバー・プラグインに依存
定期・緊急時メンテナンス片肺運転について	以下いずれかの方法で対応 •Web サーバー・プラグインを手動編集し、割り振り先から除外 •アプリケーション・サーバーを停止 •クラスターの場合は、重みづけを 0 とする(Web サーバー・プラグイン構成ファイルへの変更が自動伝播される場合)
ロールアウト・アップデート	クライアントにエラーを返す事無く、アプリケーションの更新が可能
プラグイン構成ファイルの生成	管理コンソールからプラグイン構成ファイルの生成が可能
Web サーバー可用性	アプリケーション・サーバーのアベイラビリティに影響なし
アプリケーション・サーバー障害時	Web サーバーのアベイラビリティに影響なし

1.2.4.8 Web サーバー配置の 1:1 構成の検討

1:1 構成について説明します。これは、Web サーバーとアプリケーション・サーバーの数の比が 1:1 である構成を意味します。つまり、1つの Web サーバーが1つのアプリケーション・サーバーにリクエストを割り振る構成です。

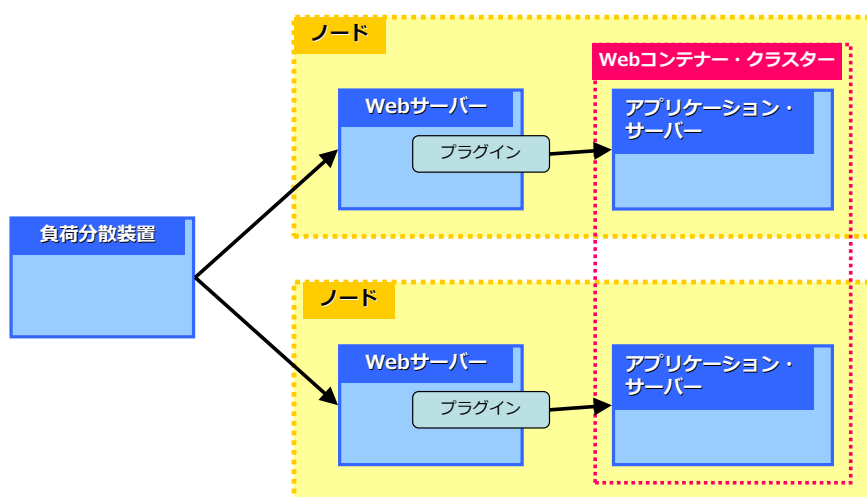


図 1-24 負荷分散装置とクラスター (ローカル Web サーバー 1:1構成)

1:N 構成と比較した場合の差異は、以下の 3 つです。

1. 負荷分散装置からアプリケーション・サーバーまでの障害検知が可能となること

アプリケーション・サーバー上にヘルスチェック用のアプリケーションを配置しておくことで、負荷分散装置から Web サーバーを経由しアプリケーション・サーバーまでの稼働状況を確認することができます。必要であれば、データベース・サーバーなどの稼働確認を行うようにヘルスチェック用のアプリケーションで実装しておくこともできます。

Web サーバーもしくはアプリケーション・サーバーに障害が発生している場合に、負荷分散装置から障害の起きた系の Web サーバーへのリクエストの転送を止める形で、Web サーバーとアプリケーション・サーバーをセットで切り離す運用となります。

2. Web サーバーとアプリケーション・サーバーのログの突合せが容易であること

1つのアプリケーション・サーバーには、1つの Web サーバーからのリクエストしか到達しないため、パフォーマンス問題判別時などには、単一の Web サーバーとアプリケーション・サーバーのセットのログのみ解析することで傾向を判別することができます。

1:N 構成で同様の作業を行うには、アプリケーション・サーバー台数分のログ解析が必要です。

3. Web サーバーの配置の自由度が少ないこと

Web サーバーを分離して DMZ に置く場合は、DMZ 上の Web サーバーの台数をアプリケーション・サーバーの台数とそろえる必要があります。一般的に、アプリケーション・サーバーに比べて Web サーバーは負荷が少なくてすみませんが、台数の調整はできません。1:N 構成であれば、Web サーバーの台数とアプリケーション・サーバーの台数は個別に設定できるため、適切なリソース配置をとることができます。

また、一つの Web サーバー・クラスターから異なるアプリケーションが稼働する複数のアプリケーション・サーバー・クラスターへの割り振りをおこなうような場合も、構成が複雑になりすぎるため適しません。

表 1-14 1:1 構成のまとめ

	1:1 構成
負荷分散装置からの障害検知の範囲	負荷分散装置で Web サーバーからアプリケーション・サーバーまで、さらには、データベース・サーバーまで検知可能
アフィニティー	負荷分散装置に依存
定期・緊急時メンテナンス片肺運転について	負荷分散装置で対応
ロールアウト・アップデート	クライアントにエラーが返る事がある
プラグイン構成ファイルの生成	GenPluginCfg コマンドで生成する必要がある (管理コンソールからは生成できない)。
Web サーバー障害時	アプリケーション・サーバーのアベイラビリティに影響
アプリケーション・サーバー障害時	Web サーバーのアベイラビリティに影響

1.2.5 WAS ND 構成例

この節では、WAS ND を使用した構成例を説明します。

1.2.5.1 負荷分散装置とクラスター (リモート Web サーバー)

この構成では、負荷分散装置から送られたリクエストは Web サーバー・プラグインで負荷分散されるため、1:N の構成となります。リモート Web サーバーのため、アプリケーション・サーバーをファイアウォールの内側に置くことができる点が利点といえます。

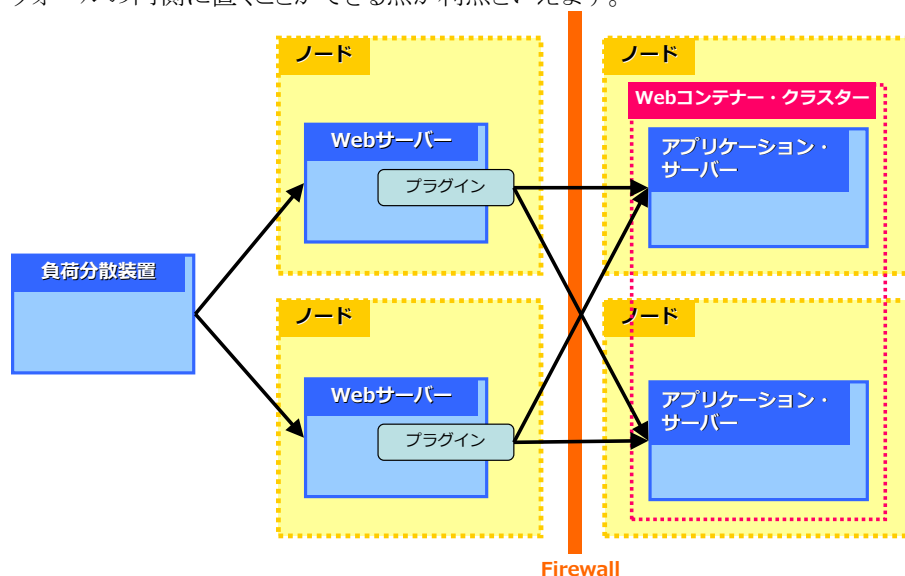


図 1-25 負荷分散装置とクラスター (リモート Web サーバー)

1.2.5.2 負荷分散装置とクラスター (リモート Web サーバー 1:1 構成)

1-2-5-1 章のトポロジーと同じ機器構成で、Web サーバー:アプリケーション・サーバーの関係を 1:1 に構成することもできます。

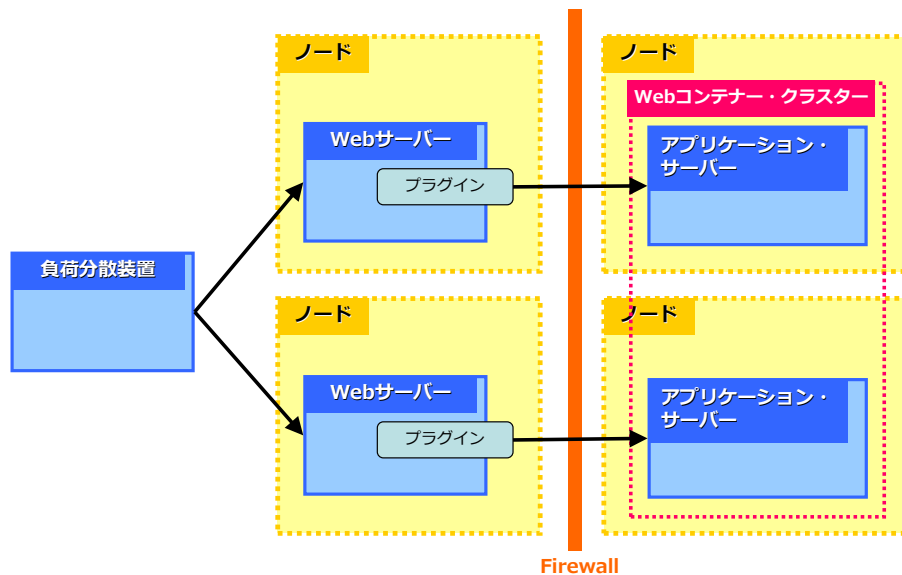


図 1-26 負荷分散装置とクラスター (リモート Web サーバー)

1.2.5.3 Web コンテナと EJB コンテナの分離

Web コンテナと EJB コンテナを別筐体で構成しています。また、それぞれのコンテナ毎にクラスターを構成しています。EJB の負荷を複数マシンに分散することが可能です。

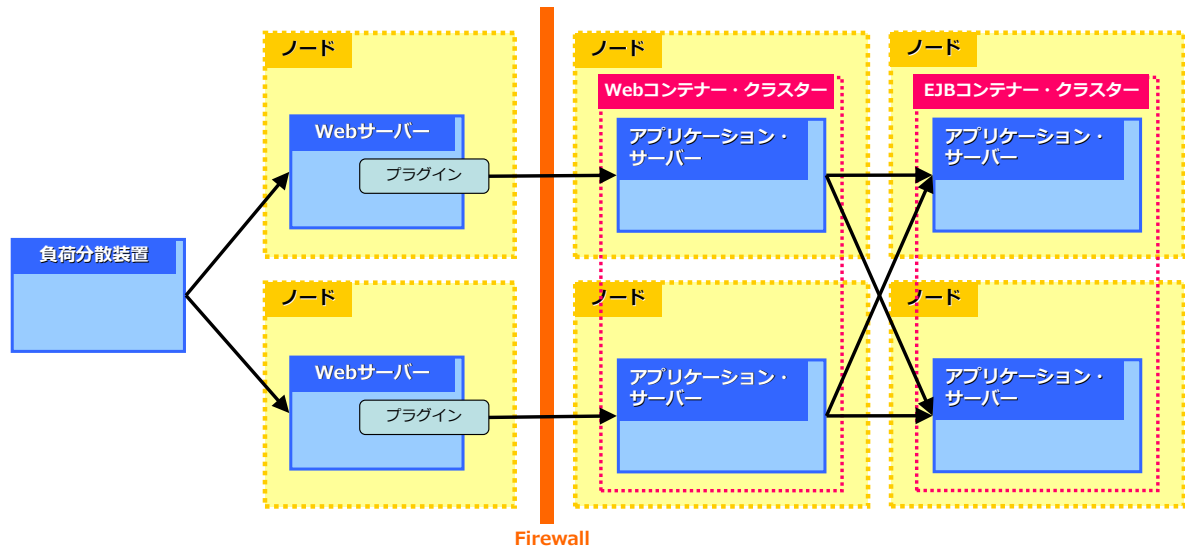


図 1-27 負荷分散装置とクラスター (リモート Web サーバー)

1.2.5.4 動的クラスター

Web コンテナを動的クラスターとして作成し、Web コンテナの動的クラスターに対してはオンデマンド・ルーター (ODR) からリクエストの割り振りを行います。

「図 1-28 負荷分散装置と ODR と動的クラスター」で、赤太点線枠で囲まれているのが、ODR のクラスターです。ODR はアプリケーション・サーバーと同様に、単体でも構成できますが、単一障害点 (SPOF) とならないよう下図のように ODR クラスターとして構成します。

ここでは図が煩雑化しすぎないよう EJB コンテナは含めていませんが、Web コンテナの後段に動的クラスターとして作成した EJB コンテナを配置し、Web コンテナから EJB コンテナ宛にリクエストを動的に振り分けするような構成を取ることができます。

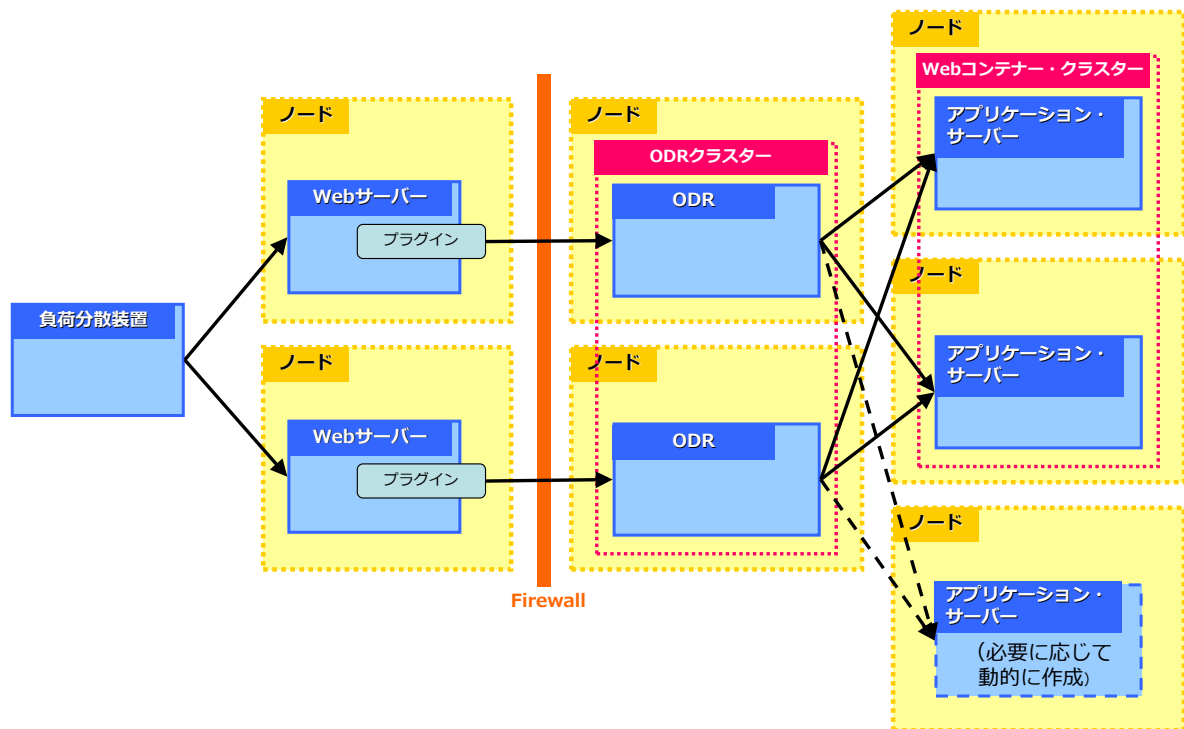


図 1-28 負荷分散装置と ODR と動的クラスター