Oracle® Fusion Middleware

Configuring Elasticity in Dynamic Clusters for Oracle WebLogic Server

14c (14.1.1.0.0)
F18328-03
February 2023

ORACLE

Oracle Fusion Middleware Configuring Elasticity in Dynamic Clusters for Oracle WebLogic Server, 14c (14.1.1.0.0)

F18328-03

Contents

5 Calendar-Based Scaling

6 Policy-Based Scaling

7 Configuring the Data Source Interceptor

8 Configuring the Script Interceptor

Preface

This document describes how to use elasticity to automatically scale up and scale down dynamic clusters in Oracle WebLogic Server.

Audience

This document is a resource for system administrators and operators responsible for monitoring and managing a WebLogic Server installation. It is relevant to all phases of a software project, from development through test and production phases.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

- See Overview of the WLDF Architecture in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server* for a description of the WebLogic Diagnostics Framework (WLDF), a monitoring and diagnostic framework that defines and implements a set of services that run within WebLogic Server processes and participate in the standard server life cycle.
- Dynamic Clusters in *Administering Clusters for Oracle WebLogic Server* describes how to create and configure dynamic clusters.

- See the following related task topics in the *Oracle WebLogic Server Administration Console Online Help* for more information:
  - Create dynamic clusters
  - Configure elasticity for a dynamic cluster
  - Perform on-demand scaling for a dynamic cluster
  - Configure policies and actions

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview

Elasticity enables the automatic scaling of dynamic clusters and re-provisioning of associated resources in Oracle WebLogic Server. The following sections provide an overview of elasticity and also introduces relevant terminology used in this guide, including the elasticity framework, smart rules and policies and actions:

What is Elasticity?

Elasticity provides the ability to configure a dynamic cluster so that it can be scaled up or down, with its resources provisioned automatically as appropriate. You can configure elastic scaling in a dynamic cluster in either of the following ways:

- Manually adding or removing a running dynamic server instance from an active dynamic cluster. This is called on-demand scaling. You can perform on-demand scaling using the Fusion Middleware component of Enterprise Manager, the WebLogic Server Administration Console, or the WebLogic Scripting Tool (WLST).
- Establishing policies that set the conditions under which a dynamic cluster should be scaled up or down and actions that define the scaling operations themselves. When the conditions defined in the scaling policy occur, the corresponding scaling action is triggered automatically.

  Policies and actions that scale a dynamic cluster are configured on a diagnostic system module that is targeted to the Administration Server.

The elasticity framework leverages the WebLogic Diagnostic Framework (WLDF) Policies and Actions system. Using the WLDF Policies and Actions component, you can write policy expressions that automatically scale up or scale down a dynamic cluster. Configuring elasticity for a dynamic cluster typically involves the following:

1. Creating a diagnostic system module for the scaling actions and policies
2. Defining scale up or scale down actions (that is, *what* and *how* to scale)
3. Defining scale up or scale down policies (that is, *when* to scale)
4. Assigning the scaling actions to the corresponding scaling policies
5. Targeting the diagnostic system module to the Administration Server

These policies will monitor one or more resources, such as memory, idle threads, and CPU. When the configured threshold is met, the scaling action is triggered. For more information on WLDF and diagnostic policies and actions, see Configuring Policies and Actions in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

Policies for elasticity can be based on the following types of data:

- Trends over time, or historical data, such as changes in average values during a specific time interval. For example, a policy can be based on average JVM heap usage above a certain threshold.
- Runtime metrics relevant to all server instances in a cluster, not just one server instance.
- Data from multiple services that are considered together. For example, a policy can be based on response-time metrics reported by a load balancer and message-backlog metrics from a message queue.
- Calendar-based schedules. Scaling policies can identify a specific calendar time, such as time of day or day of week, to define when an action triggers.
- Log rules or event data rules.

Elasticity is available only for scaling dynamic clusters.

Related Terminology

Learn a list of terminology associated with elasticity.

The following table defines terminology used in this guide, *Configuring Elasticity in Dynamic Clusters for Oracle WebLogic Server*:

Table 1-1 Elasticity Terminology

Term	Description
Elasticity framework	Component that builds upon the existing WLDF Policies and Actions system that allows administrators to construct complex rules to monitor resource usage in a cluster to effect scaling or administrative actions relevant particularly to dynamic clusters.
Policy	Sets the conditions under which a dynamic cluster should be scaled up or down. When the conditions defined in a scaling policy occur, the corresponding scaling action is triggered automatically.
Action	Defines the scaling operations themselves.
Dynamic cluster	A cluster that contains one or more generated (dynamic) server instances that are based on a single shared server template.
On-demand scaling	Scaling by manually adding or removing a running server instance from an active dynamic cluster.
Calendar-based scaling	Scaling based on calendar-based schedules and that specify a particular time or date when a scaling action is executed.
Policy-based scaling	Scaling based on policies and associated actions. A policy sets the conditions under which a scaling operation should occur, and when these conditions are met, the configured scaling action is executed.
Smart rule	Prepackaged policy expression with a set of configurable parameters that allow you to create a complex policy expression simply by specifying the values for those parameters.

Introducing the Script and Data Source Interceptors

Interceptors can be used to assist in coordinating with other WebLogic Server subsystems and other components during scaling operations. Interceptors are

particularly useful for performing functions such as reprovisioning resources prior to, or immediately following, a scaling operation.
Configuring Elasticity in Dynamic Clusters for Oracle WebLogic Server introduces two interceptors, the script interceptor and the data source interceptor.

For information on using the script interceptor to coordinate with other systems during scaling operations, see Configuring the Script Interceptor.

For information on using the data source interceptor to check database connections before a scale up operation, see Configuring the Data Source Interceptor .

2

Requirements for Configuring Elasticity

To configure elasticity for a dynamic cluster in Oracle WebLogic Server, you must have a dynamic cluster for which you specify a set of key attributes related to scalability, such as the initial, minimum, and maximum number of dynamic server instances that can be running.And depending on your scalability needs, you can have additional tasks, such as provisioning machines, configuring database resources, creating policies that automate scaling operations when certain conditions arise, and more.
This chapter includes the following sections:

Configuring Dynamic Clusters

Elasticity is only available for use with dynamic clusters. Before configuring elastic scaling, you must create and configure a dynamic cluster. See Creating and Configuring Dynamic Clusters in *Administering Clusters for Oracle WebLogic Server*.
Table 2-1 explains the `DynamicServersMBean` attributes that are used to configure a dynamic cluster for elasticity.

Table 2-1 Attributes for Configuring Elasticity in Dynamic Clusters

Attribute	Description
`DynamicClusterCooloffPeriodSeconds`	The cool-off period, in seconds, used during scaling operations. After a scaling operation is performed, subsequent requests for scaling operations will be ignored during this cool-off period. The default value is `900` and the minimum value is `0`.
`DynamicClusterShutdownTimeoutSeconds`	The timeout period, in seconds, use while gracefully shutting down a dynamic server instance. If the dynamic server instance does not shut down before the specified timeout period, it will be forcibly shut down. The default value is `0`.
`DynamicClusterSize`	The initial number of dynamic server instances to be provisioned in the dynamic cluster. The minimum value for this attribute is `0` and the maximum value is `800`.
`IgnoreSessionsDuringShutDown`	Indicates if the Elasticity Framework should ignore inflight HTTP requests while shutting down dynamic server instances during scale down operations.
`MaxDynamicClusterSize`	The maximum number of dynamic server instances that can be used in scaling operations. The default value for MaxDynamicClusterSize is 8. The minimum value is `0` and the maximum value is `800`. Note that if a scale up operation attempts to exceed the limit set in `MaxDynamicClusterSize`, the operation will not fail, but WebLogic Server will only add running dynamic server instances until this limit is reached.

Table 2-1 (Cont.) Attributes for Configuring Elasticity in Dynamic Clusters

Attribute	Description
MinDynamicClusterSize	The minimum number of running dynamic server instances that the Elasticity Framework will attempt to keep in the dynamic cluster during scaling operations. The default value is 1. The minimum value is 0 and the maximum value is 800. Note that, during a scale down operation, any attempt to scale down below the limit set in MinDynamicClusterSize is not allowed. WebLogic Server will only scale down to this limit.
WaitForAllSessionsDuringShutdown	Indicates if the Elasticity Framework should wait for all persisted and non-persisted inflight HTTP sessions while shutting down dynamic server instances during scaling operations.

You can set these attributes and configure your dynamic cluster for elasticity using the WebLogic Server Administration Console, the Fusion Middleware Control component of Enterprise Manager (EM), or WebLogic Scripting Tool (WLST). For information about using the WebLogic Server Administration Console, see Configure elasticity for a dynamic cluster in the *Oracle WebLogic Server Administration Console Online Help*.

Example 2-1 demonstrates how to use WLST to define dynamic cluster attributes for elasticity in an existing dynamic cluster.

Example 2-2 demonstrates how to create and configure a dynamic cluster for elasticity. This dynamic cluster is used in the policy based scaling demonstration in Policy-Based Scaling Example.

Example 2-1 Configuring a Dynamic Cluster for Elasticity

```
startEdit()
cmo.setDynamicClusterSize(4)
cmo.setMaxDynamicClusterSize(8)
cmo.setMinDynamicClusterSize(2)
cmo.setDynamicClusterCooloffPeriodSeconds(120)
save()
activate()
```

Example 2-2 Creating and Configuring a Dynamic Cluster

```
edit()
startEdit()

dynCluster=cmo.createCluster(DynamicCluster)

  dynServers=dynCluster.getDynamicServers()
  dynServers.setServerTemplate(dynamicServerTemplate)
  dynServers.setMinDynamicClusterSize(2)
  dynServers.setMaxDynamicClusterSize(8)
  dynServers.setDynamicClusterSize(4)
  dynServers.setDynamicClusterCooloffPeriodSeconds(120)
  dynServers.setCalculatedListenPorts(true)
  dynServers.setCalculatedMachineNames(true)
  #
  # Dynamic server names will be dynamic-server-1, dynamic-server-2, ...,
  # dynamic-server-10.
  #
  dynServers.setServerNamePrefix(dynamic-server-)
```

```
save()
activate()
```

Configuring Policies and Actions(changed)

In policy-based scaling, a policy sets the conditions under which a scaling operation must occur. And when these conditions are met, an elastic action performs the scaling operation. To configure policy-based scaling of a dynamic cluster, you create a diagnostic system module that contains one or more policies, and then assign an elastic action (scale up or scale down) to each of those policies. For information on the Policies and Actions component of WLDF, see Configuring Policies and Actions in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.
To use the WebLogic Server Administration Console to create and configure policies and actions in a diagnostic system module, see Configure policies and actions in the *Oracle WebLogic Server Administration Console Online Help*.

Targeting the Diagnostic System Module

After you have defined the policies and elastic scaling actions in a diagnostic system module, you must target that diagnostic system module to the Administration Server.

Some of the smart rules and scaling actions used for elasticity must execute from the Administration Server so that they can have visibility across the dynamic cluster. As a result of this, diagnostic system modules intended for elastic scaling of dynamic clusters must be targeted to the Administration Server. If such a module is targeted to the cluster itself, or to an individual Managed Server, errors are generated and the policies and actions are consequently ignored by WLDF.

Provisioning Machines

To configure policy-based elastic scaling, you must appropriately provision machines for the maximum necessary scale-up capacity.

3
On-Demand Scaling

Oracle WebLogic Server supports on-demand scaling, which is the ability to scale a dynamic cluster up or down by manually adding or removing running dynamic server instances as needed. On-demand scaling is the simplest and most basic form of incorporating elasticity into a WebLogic domain.

This chapter includes the following sections:

What is On-Demand Scaling?

On-demand scaling allows you to manually add or remove running dynamic server instances from an active dynamic cluster as needed.For example, if the average user-request backlog in dynamic cluster members is trending up, indicating a need for higher processing capacity, you can add running dynamic server instances to the dynamic cluster. When the backlog for user-requests drops substantially, you can shut down idle dynamic server instances.
When you expand the cluster, the size must not exceed the limit set in Maximum Dynamic Cluster Size. WebLogic Server adds only running dynamic server instances up to the Maximum Dynamic Cluster Size limit. Also, when you shrink the cluster, the number of running dynamic servers cannot be below the limit set in Minimum Dynamic Cluster Size.

To perform on-demand scaling, you must have a dynamic cluster configured. See Dynamic Clusters in *Administering Clusters for Oracle WebLogic Server*.

You can perform on-demand scaling using the WebLogic Server Administration Console, Fusion Middleware Control, or WLST, as described in the following sections:

- On-Demand Scaling Using the WebLogic Server Administration Console
- On-Demand Scaling Using WLST

> **Note:**
>
> Before performing on-demand scaling of a dynamic cluster, make sure that the scaling operation will not have adverse effects on the rest of the system. For example, before a scale up, it may be necessary to adjust JDBC connection pool sizes and Work Manager parameters to avoid exceeding database capacity. Before a scale down, make sure that any inflight work has a chance to finish before the dynamic server instances are shut down. Also, if you are using JMS or JTA, do not use a scale down approach that lowers the dynamic cluster size configuration setting. See Best Practices for Using Clustered JMS Services in *Administering JMS Resources for Oracle WebLogic Server*.

On-Demand Scaling Using the WebLogic Server Administration Console

You can perform on-demand scaling using the WebLogic Server Administration console.

1. In the left navigation pane, select **Environment > Clusters**.
2. In the Clusters table, select the name of the dynamic cluster you want to scale up or scale down.
3. Select **Control > Scaling**.
4. In the **Desired Number of Running Servers** field, enter the number of running dynamic server instances you want in the dynamic cluster.
5. Click **Ok**.

See Perform on-demand scaling for a dynamic cluster in the *Oracle WebLogic Server Administration Console Online Help*.

On-Demand Scaling Using WLST

You can manually scale a dynamic cluster up or down using the WLST commands.

To manually scale your dynamic cluster using WLST:

- Use the `scaleUp` command to increase the number of running dynamic servers in your dynamic cluster. When you use the `scaleUp` command and enable the `updateConfiguration` argument, WLST will increase the size of the dynamic cluster by the specified number of dynamic servers and start these server instances.
- Use the `scaleDown` command to decrease the number of running dynamic servers in your dynamic cluster. When you use the `scaleDown` command and enable the `updateConfiguration` argument, WLST will gracefully shut down the specified number of running dynamic servers and remove them from the dynamic cluster.

> **Note:**
>
> The `scaleDown` command lowers the cluster dynamic cluster size setting when its `updateConfiguration` option is enabled, which in turn can abandon JMS persistent messages and JTA transactions that are associated with retired servers. If you are using JMS or JTA, do not use `scaleDown` with its `updateConfiguration` option enabled. See Best Practices for Using Cluster Targeted JMS Services in *Administering JMS Resources for Oracle WebLogic Server*.

See scaleUp and scaleDown commands in *WLST Command Reference for WebLogic Server*.

See Expanding or Reducing Dynamic Clusters in *Administering Clusters for Oracle WebLogic Server*.

Example 3-1 demonstrates how to use WLST to scale a dynamic cluster named `DynamicCluster` up by two running dynamic server instances and then scale it back down by one running dynamic server instance.

Example 3-1 On-Demand Scaling Using WLST

```
wls:/offline> connect('weblogic','weblogic','t3://localhost:7001')
Connecting to weblogic server instance running at t3://localhost:7001 as username
weblogic...

Successfully connected to Admin Server 'AdminServer' that belongs to domain 'mydomain'.

wls:/mydomain/serverConfig/> scaleUp("DynamicCluster", 2, true ,true)
Remote ScaleUp started successfully after 38 seconds. Waiting for 2 servers to reach
the running state.  The timeout is 600 seconds.

All servers are now running.
wls:/mydomain/serverConfig/> scaleDown("DynamicCluster", 1, true ,true)
Remote ScaleDown started successfully after 0 seconds.
The servers were stopped successfully.
```

4
Elastic Actions

The Policies and Actions component of the WebLogic Diagnostics Framework (WLDF) in Oracle WebLogic Server provides two actions for performing elastic operations in a dynamic cluster: scale up, and scale down.The following sections describe the two elastic actions and the roadmap for configuring the elastic actions:

Introducing the Elastic Actions

The scale up and scale down actions are used to start or stop running dynamic server instances in a dynamic cluster during scaling operations.

Elastic actions are associated with policies. As a policy's conditions are met, the elastic action is triggered and the scaling operation begins. This type of scaling is called policy based scaling. See Policy-Based Scaling .

Only one elastic action can be assigned to a policy. However, any number of non-scaling actions can be assigned to a policy, and elastic actions can be used in conjunction with non-scaling actions.

> **Note:**
>
> To configure automated elasticity for a dynamic cluster, you must create a domain-scope diagnostic system module in which you define the scaling policies, along with their corresponding elastic actions, and then target that diagnostic module to the Administration Server.

See Configuring Policies and Actions in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server* for information on the Policies and Actions component of WLDF.

Configuring Scale Up Actions

When an associated policy is triggered, the scale up action adds running dynamic server instances to the specified dynamic cluster.

Example 4-1 shows a scale up action that will scale a dynamic cluster up by one running dynamic server instance.

Example 4-2 shows how to create and configure the scale up action shown in Example 4-1 using WLST. This scale up action is used as part of the policy-based scaling demo in Policy-Based Scaling Example.

Example 4-1 Sample Configuration for a Scale Up Action (in DIAG_MODULE.xml)

```
<wldf-resource>
   <name>ClusterManager</name>
   <watch-notification>
     <!-- One or more policy configurations -->
     <scale-up-action>
        <name>scaleUp</name>
        <cluster-name>DynamicCluster</cluster-name>
        <scaling-size>1</scaling-size>
     </scale-up-action>
     <!-- Other action configurations -->
   </watch-notification>
</wldf-resource>
```

Example 4-2 Configuring a Scale Up Action

```
startEdit()
scaleUp=wn.createScaleUpAction('scaleUp')
  scaleUp.setScalingSize(1)
  scaleUp.setClusterName(DynamicCluster)
save()
activate()
```

Configuring Scale Down Actions

When an associated policy is triggered, the scale down action shuts down running dynamic server instances in the specified dynamic cluster.

Example 4-3 shows a scale down action that will scale down a dynamic cluster by one dynamic server instance.

Example 4-4 shows how to create and configure the scale down action shown in Example 4-3. This scale down action is used as part of the policy-based scaling demo in Policy-Based Scaling Example.

Shutting down server instances during a scale down operation can take a significant amount of time if there are, for instance, unreplicated sessions. Until they time out, the server instance will not shut down.

To reduce the time it takes to complete a scale down operation, the following `DynamicServersMBean` attributes are used:

- `DynamicClusterShutdownTimeoutSeconds`
- `IgnoreSessionsDuringShutdown`
- `WaitForAllSessionsDuringShutdown`

By specifying a timeout period or ignoring sessions during server shutdown, the shutdown time can be limited. However, remaining sessions may be lost. See Table 2-1 for descriptions of these attributes.

Example 4-3 Sample Configuration for a Scale Down Action (in DIAG_MODULE.xml)

```
<wldf-resource>
   <name>ClusterManager</name>
   <watch-notification>
     <!-- One or more policy configurations -->
```

```
    <scale-down-action>
       <name>scaleDown</name>
       <cluster-name>DynamicCluster</cluster-name>
       <scaling-size>1</scaling-size>
    </scale-down-action>
    <!-- Other action configurations -->
  </watch-notification>
</wldf-resource>
```

Example 4-4 Configuring a Scale Down Action

```
startEdit()
scaleDown=wn.createScaleDownAction('scaleDown')
scaleDown.setScalingSize(1)
scaleDown.setClusterName(DynamicCluster)
save()
activate()
```

Shutting down server instances during a scale down operation can take a significant amount of time if there are, for example, unreplicated sessions. Until those unreplication sessions time out, the server instance is not shut down.

To reduce the time it takes to complete a scale down operation, you can configure the following `DynamicServersMBean` attributes:

- `DynamicClusterShutdownTimeoutSeconds`
- `IgnoreSessionsDuringShutdown`
- `WaitForAllSessionsDuringShutdown`

By specifying a timeout period or ignoring sessions during server shutdown, the shutdown time can be reduced. However, remaining sessions may be lost. See Configuring Dynamic Clusters for descriptions of these attributes.

Roadmap for Creating and Configuring Elastic Actions

The WebLogic Server documentation includes several topics to help you learn how to configure the elastic scaling actions using either the WebLogic Server Administration Console or Fusion Middleware Control.

Table 4-1provides the roadmap for creating and configuring elastic actions.

Table 4-1 Elastic Actions Roadmap

For more information on...	See...
Creating actions in the WebLogic Server Administration Console	Create actions for policies in a diagnostic system module in the *Oracle WebLogic Server Administration Console Online Help*
Creating elastic actions in the WebLogic Server Administration Console	Create a scale up action and Create a scale down action in the *Oracle WebLogic Server Administration Console Online Help*
Configuring elastic actions in the WebLogic Server Administration Console	Configure a scale up action and Configure a scale down action in the *Oracle WebLogic Server Administration Console Online Help*

sample link REST API Link

5

Calendar-Based Scaling

In Oracle WebLogic Server, you use calendar-based scaling to scale a dynamic cluster up and down based on a calendar schedule.The following sections describe how to configure calendar-based scaling with examples:

Overview of Calendar-Based Scaling

WLDF supports actions for policies to be triggered according to a calendar schedule at a specific time, after a duration of time, or at timed intervals. Calendar-based scaling executes a scaling action based on a defined schedule.

Calendar-based rule schedules are only supported for `Harvester` rules that utilize the Java Expression Language (EL) as the expression language. The following policy types support calendar-based rule schedules:

- Calendar-based
- Smart rule-based
- Collected metrics

Configuring Calendar-Based Scaling

To configure calendar-based scaling, you create a policy and define the policy's schedule, then create and assign a scaling action to that policy.

Policy schedules can be set to set to execute based on the following:

- Every N seconds
- Every N minutes
- Every N hours
- Specific days of the week (at a specific time)
- Specific days of the month (at a specific time)

Policy schedules are based on the `WLDFScheduleBean`. Table 5-1 lists the attributes used for setting policy schedules.

Table 5-1 WLDFScheduleBean Attributes

Attribute	Default Value	Allowable Values
dayOfMonth	* (every day)	• last, which specifies the last day of the month • 1st to 31st • 1 to 31 • Sun, Mon, Tues, Weds, Thurs, Fri, Sat • -*x* where *x* is the number of days before the last day of the month (for example, -7 would be seven days before the last day of the month)
daysOfWeek	* (every day)	• 0 to 7 (0 and 7 both represent Sunday) • Sun, Mon, Tues, Weds, Thurs, Fri, Sat
hour	* (every hour)	• 0 to 23
minute	*/5 (every five minutes)	• 0 to 59
month	* (every month)	• 1 to 12 • Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
second	0	• 0 to 59
timezone	n/a	• Defaults to the local VM time zone
year	* (every year)	• A four-digit calendar year, for example: 2015

These attributes can be specified in several ways using the following syntax . This syntax is supported by the EJB `ScheduleExpression` class. See the `ScheduleExpression` documentation for full details about this syntax.

1. As a single value. For example:

```
second = "10"
month = "Oct"
```

2. Using a wild card (*), which represents all of the attribute's allowable values. For example, to specify every day of the week:

```
dayOfWeek = "*"
```

3. As a range, which constrains the attribute to an inclusive range of values. You specify the two ends of the range, separated by a dash (-). For example:

```
second = "1-10"
dayOfWeek = "Mon-Fri"
```

4. As an increment, which constrains the attribute based on a starting point and an interval, using a slash (/) to separate the two values. The use of increments is supported only for specifying `second`, `minute`, and `hour` attributes. For example:

   - `minute = "*/5"` specifies every five minutes. This syntax is equivalent to `minute = "0,5,10,15,20,25,30,35,40,45,50,55"`
   - `second = "30/10"` specifies every 10 seconds within the minute starting at second 30. This syntax is equivalent to `second = "30,40,50"`

> **Note:**
>
> When the maximum value for an attribute is reached, the incrementing stops. That is, it will not exceed the maximum allowable value.

5. As a list, which constrains the attribute to two or more allowable values, or range of values, using a comma-separated list. Each list item must be a value or range; list items cannot be wild cards, lists, or increments. For example:

```
second = "10,20,30"
dayOfWeek = "Mon,Wed,Fri"
minute = "0-10,30,40"
```

> **Note:**
>
> - Before you configure calendar-based scaling in your domain, make sure that you have created a dynamic cluster and completed the prerequisite tasks that are described in Requirements for Configuring Elasticity .
> - Calendar-based scaling policies must be configured in a domain-wide diagnostic system module that is targeted to the Administration Server.

You can configure calendar based scaling using the WebLogic Server Administration Console, Fusion Middleware Control, or WLST, as described in the following sections:

- Configuring Calendar-Based Scaling Using the WebLogic Server Administration Console
- Configuring Calendar-Based Scaling Using WLST

Configuring Calendar-Based Scaling Using the WebLogic Server Administration Console

To configure calendar-based scaling using the WebLogic Server Administration Console:

1. Create and enable a policy that supports calendar-based rules. See the following topics in the *Oracle WebLogic Server Administration Console Online Help*:
   - Create calendar based policies for a diagnostic system module
   - Create smart rule based policies for a diagnostic system module
   - Create collected metrics policies for a diagnostic system module

> **Note:**
>
> When creating a calendar-based policy using the WebLogic Server Administration Console, you can specify the schedule for the policy, and also create a scaling action and assign it to the policy. If you choose not to specify the policy schedule and scaling action at the time you're creating the policy itself, you can use the following steps to complete the configuration at a later time.

2. Set or update the schedule for your policy:
   a. In the left pane of the Administration Console, expand **Diagnostics** and select **Diagnostic Modules**.
   b. Select the diagnostic module containing the policy you want to configure.
   c. Select **Configuration > Policies and Actions > Policies** and click the name of the policy you want to configure.
   d. Select the **Schedule** page and define the desired schedule settings for your policy.
   e. Click **Save**.
3. Create and enable a scaling action. See the following topics in the *Oracle WebLogic Server Administration Console Online Help*:
   - Create a scale up action
   - Create a scale down action
4. Assign a scaling action to your policy:

> **Note:**
>
> You can assign only one scaling action to a given policy.

   a. In the left pane of the Administration Console, expand **Diagnostics** and select **Diagnostic Modules**.
   b. Select the diagnostic system module containing the policy you want to configure.
   c. Select **Configuration > Policies and Actions > Policies** and click the name of the policy you want to configure.
   d. Select the **Actions** page.
   e. In the **Scaling Actions** section, select either **Scale Up Action** or **Scale Down Action** and choose the scale up or scale down action you want to assign to this policy. Alternatively, you can click **New Scale Up Action** or **New Scale Down Action** to create a new scaling action for this policy.
   f. Click **Save**.

Configuring Calendar-Based Scaling Using WLST

The following examples show how to use WLST to create and configure calendar-based policies that scale a dynamic cluster at a certain time on a certain date. Example 5-1 shows the commands to configure a calendar-based policy that executes a scale up action at 3:00 a.m. on December 26. Example 5-2 shows the commands to configure a calendar-based policy that executes a scale down action at midnight on January 15.

Example 5-1 Calendar-Based Policy With a Scale Up Action

```
calendarScaleUp=wn.createWatch('ChristmasReturnsScaleUpWatch')
  calendarScaleUp.setExpressionLanguage('EL')
  calendarScaleUp.getSchedule().setHour('3')
```

```
  calendarScaleUp.getSchedule().setMinute('0')
  calendarScaleUp.getSchedule().setSecond('0')
  calendarScaleUp.getSchedule().setDayOfMonth('26')
  calendarScaleUp.getSchedule().setMonth('Dec')
  calendarScaleUp.setEnabled(false)
  calendarScaleUp.addNotification(scaleUp)
```

Example 5-2 Calendar-Based Policy With a Scale Down Action

```
calendarScaleDown=wn.createWatch('PostChristmasReturnsScaleDownWatch')
  calendarScaleDown.setExpressionLanguage('EL')
  calendarScaleDown.getSchedule().setHour('0')
  calendarScaleDown.getSchedule().setMinute('0')
  calendarScaleDown.getSchedule().setSecond('0')
  calendarScaleDown.getSchedule().setDayOfMonth('15')
  calendarScaleDown.getSchedule().setMonth('Jan')
  calendarScaleDown.setEnabled(false)
  calendarScaleDown.addNotification(scaleDown)
```

6
Policy-Based Scaling

Oracle WebLogic Server supports policy-based scaling, which is the ability to automatically execute a scaling operation on a dynamic cluster in response to a change in demand, or any other condition, that is monitored by a policy that has been configured in a domain-scope diagnostic system module.The following sections describe policy-based scaling in WebLogic Server and include an example of policy-based scaling using smart rules to track the average throughput in a cluster:

What is Policy-Based Scaling?

Policy-based scaling is based on policies and associated actions, leveraging the Policies and Actions component of the WebLogic Diagnostics Framework (WLDF). A policy sets the conditions under which a scaling operation on a dynamic cluster must occur, and when these conditions are met, the scaling action performs the scaling operation. For more information on the Policies and Actions component of WLDF, see Configuring Policies and Actions in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

WLDF provides two elastic actions, scale up and scale down, that you can assign to a policy to perform a scaling operation. For more information on the scale up and scale down actions, see Elastic Actions . Only one scaling action can be assigned to a given policy.

For example, Example 6-1 shows the WLST commands to create and configure a policy that, when triggered, executes a scale up action. Note the following:

- The scale up action, `scaleUp`, is created. This action is configured to scale up the dynamic cluster, `DynamicCluster`, by one server instance.
- The policy, named `highMark`, is configured to be evaluated every 10 seconds.
- When the policy is triggered — that is, it is evaluated to `true` — the corresponding scale up action is executed.

Similarly, Example 6-2 shows the configuration of a policy that, when triggered, executes a scale down action. Note the following:

- The scale down action, `scaleDown`, is created. This action is configured to scale down the dynamic cluster, `DynamicCluster`, by one server instance.
- The policy, named `LowMark`, is configured to be evaluated every 10 seconds.
- When the policy is triggered — that is, it is evaluated to `true` — the corresponding scale down action is executed.

Example 6-1 Creating a Policy to Scale Up a Dynamic Cluster

```
scaleUp=wn.lookupScaleUpAction('scaleUp')
if scaleUp == None:
  print "Creating scale up action"
```

```
  scaleUp=wn.createScaleUpAction('scaleUp')
scaleUp.setScalingSize(1)
scaleUp.setClusterName(DynamicCluster)
high=wn.createWatch('highMark')
high.setExpressionLanguage('EL')
high.getSchedule().setMinute('*')
high.getSchedule().setSecond('*/10')
high.getRuleType()
high.setAlarmType('AutomaticReset')
high.setAlarmResetPeriod(150000)
high.setRuleExpression("wls:ClusterHighThroughput('"+DynamicCluster+"', '10s',
'30s', 250, 60)")
high.addNotification(scaleUp)
high.setEnabled(true)
```

Example 6-2 Creating a Policy to Scale Down a Dynamic Cluster

```
scaleDown=wn.lookupScaleDownAction('scaleDown')
if scaleDown == None:
  print "Creating scale down action"
  scaleDown=wn.createScaleDownAction('scaleDown')
scaleDown.setScalingSize(1)
scaleDown.setClusterName(DynamicCluster)
low=wn.createWatch('lowMark')
low.setExpressionLanguage('EL')
low.getSchedule().setMinute('*')
low.getSchedule().setSecond('*/10')
low.getRuleType()
low.setAlarmType('AutomaticReset')
low.setAlarmResetPeriod(60000)
low.setRuleExpression("wls:ClusterLowThroughput('"+DynamicCluster+"', '10s',
'60s', 50, 75)")
low.addNotification(scaleDown)
low.setEnabled(true)
```

> **Note:**
>
> To configure automated elasticity for a dynamic cluster, you must create a domain-scope diagnostic system module in which you define the scaling policy, along with its corresponding elastic action, and then target that diagnostic module to the Administration Server.

Introducing Smart Rules

Smart rules are prepackaged functions with a set of configurable parameters that allow you to create complex policy expressions just by specifying the values for these configurable parameters.

A smart rule can be used as a predicate within a policy expression, either alone or with other predicates to create a more complex policy expression. Smart rules policy expressions are specified using the Java Expression Language (EL). See Configuring Smart Rule Based Policies in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

Smart rules can be used in policy-based scaling. Policy-Based Scaling Example shows the use of policies that use smart rules, `wls:ClusterHighThroughput()` and

`wls:ClusterLowThroughput()`, in their rule expressions. These two smart rules are also shown in Example 6-1 and Example 6-2:

- Example 6-1 shows the `wls:ClusterHighThroughput()` smart rule. This smart rule measures whether the average throughput in a cluster is increasing, as indicated by the average value of the `ThreadPoolRuntimeMBean.Throughput` attribute in each Managed Server.

  As specified in this example, this smart rule is triggered — that is, it is evaluated to `true` — if the average `Throughput` value, collected at 10-second intervals over the previous 30 seconds, is greater than or equal to 250 on at least 60 per cent of the servers in the dynamic cluster, `DynamicCluster`. When this smart rule is triggered, it executes the scale up action, `scaleUp`, which starts up a server instance in the cluster.

- Example 6-2 shows the `wls:ClusterLowThroughput()` smart rule. This smart rule measures whether the average throughput in a cluster is decreasing, as indicated by the average value of the `ThreadPoolRuntimeMBean.Throughput` attribute in each Managed Server.

  As specified in this example, this smart rule is triggered — that is, it is evaluated to `true` — if the average `Throughput` value, collected at 10-second intervals over the previous 60 seconds, is less than 50 on at least 75 per cent of the servers in the dynamic cluster, `DynamicCluster`. When this smart rule is triggered, it executes the scale down action, `scaleDown`, which stops a server instance in the cluster.

For more information about creating and configuring smart rule-based policies:

- See Create smart rule based policies for a diagnostic system module in the *Oracle WebLogic Server Administration Console Online Help* using the WebLogic Server Administration Console
-
- See Smart Rule Reference in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

Policy-Based Scaling Example

You can create policies that use the `wls:HighThroughput()` and `wls:LowThroughput()` smart rules to scale a dynamic cluster up under high load, and then scale the dynamic cluster back down when the demand decreases.

The following topics describe the tasks for running a demonstration that shows policy-based scaling of a dynamic cluster:

- Prerequisites
- Setting the Environment
- Creating the Domain and its Resources
- Configuring and Starting the Apache Web Server
- Viewing Dynamic Server Configuration and Activity
- Using JMeter to Drive the Demo
- Using the Monitoring Dashboard to Monitor Scaling Operations
- Re-running and Stopping the Elasticity Demo

Prerequisites

To successfully run this example, first prepare your environment and download the required files used in this example.

This section includes the following tasks:

- Installing Required Software
- Downloading and Unpacking Required Files

Installing Required Software

To run this example, download and install the following software:

- Oracle WebLogic Server—for instructions and more information, see Installing the Oracle WebLogic Server and Coherence Software in *Installing and Configuring Oracle WebLogic Server and Coherence*.

> **Note:**
>
> You must have a valid JDK 8 to install and use WebLogic Server for this example. See JDK 8 Certification in *What's New in Oracle WebLogic Server*.

- Apache HTTP Server - Ensure that you have a supported Apache HTTP Server installation. See https://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html.
- Apache HTTP Server 2.2.x plug-ins for WebLogic Server are available for download on the My Oracle Support (http://support.oracle.com) and Software Delivery Cloud (http://edelivery.oracle.com) web sites as zip files containing the necessary binary and helper files. See Configure the Apache HTTP Server Plug-In for more information about configuring the plug-in for Apache HTTP Server.
- Apache JMeter—you can view instructions and download and install Apache JMeter from `http://jmeter.apache.org/download_jmeter.cgi`.

Downloading and Unpacking Required Files

The elasticity demo JAR files are available for download from the article Elasticity for Dynamic Clusters in the WebLogic Server Blog at the following location:

https://cdn.app.compendium.com/uploads/user/e7c690e8-6ff9-102a-ac6d-e4aebca50425/92a4be11-24be-41af-a122-be88dcc2684d/File/bd70c19e88c1927f0919d42171cbfe2b/wls_elasticity_demo.zip

Unpack the following files from the elasticity JAR that are used in this example:

> **Note:**
>
> Save all of the following files in the same directory. This directory is referred to as *DEMO_HOME* in this example.

- `create-domain` script - this script creates the domain in this example.
- `Elasticity5MinuteDemo.jmx`—the JMeter test plan used in this example.
- `Elasticity1HourDemo.jmx`—optional, longer JMeter test plan.

Setting the Environment

After installing WebLogic Server in the *ORACLE_HOME* directory, run the `setWLSEnv` script command to set your environment variables for this example.

1. Open a command window.

   > **Note:**
   >
   > If you are running this demo on a Windows system, you may need to have administrator privileges. If so, open the command window as follows:
   >
   > a. Click **Start**.
   >
   > b. Right-click **Command Prompt**, then select **Run as administrator**.
   >
   > If **Command Prompt** is not listed in the Start Menu, enter `command` in the Search field, then right-click **Command Prompt**.
   >
   > c. Enter the administrator credentials, if requested.

2. In the command window, change to the *ORACLE_HOME*`/wlserver/server/bin` directory.
3. Run the `setWLSEnv` script. For example:

   Windows:

   ```
   ORACLE_HOME\wlserver\server\bin> setWLSEnv.cmd
   ```

   Unix:

   ```
   prompt> sh setWLSEnv.sh
   ```

Creating the Domain and its Resources

Creating the domain using the `create-domain` script sets up your domain for this example, including the following:

- Creates the `elasticity_domain` domain and directory.
- Creates and configures the dynamic cluster, `DynamicCluster`, and four dynamic servers, `DynamicCluster-1`, `DynamicCluster-2`, `DynamicCluster-3`, and `DynamicCluster-4`.
- Create and configures the smart rule-based policies, `highMark` and `lowMark`, and the elastic actions, `scaleDown` and `scaleUp`.

To create the domain:

1. Change to the directory in which you unpacked the demo JAR file. For example, on Windows:

   ```
   ORACLE_HOME\wlserver\server\bin> cd DEMO_HOME
   ```

2. Run the `create-domain` script. For example:

   Windows:

   ```
   DEMO_HOME> create-domain.cmd username password
   ```

   Unix:

   ```
   DEMO_HOME> sh create-domain.sh username password
   ```

   Use the WebLogic Administrator's username and password to run the script.

Configuring and Starting the Apache Web Server

After the installation of Apache HTTP Server and the plug-in for Apache HTTP Server, ensure that you configure the `httpd.conf` file with the following details:

```
LoadModule weblogic_module /home/myhome/weblogic-plugins/lib/mod_wl.so
```

Also, specify the WebLogic cluster and all the ports of the managed servers in the cluster. You can also provide the cluster DNS name instead. For example:

```
<IfModule mod_weblogic.c>
WebLogicCluster localhost:8001,localhost:8002
</IfModule>
<Location>
SetHandler weblogic-handler
</Location>
```

Starting the Apache Web Server

Start the Apache WebServer using the following command from the Apache `bin` directory:

Windows

```
httpd.exe
```

Unix

```
sudo systemctl start httpd
```

Viewing Dynamic Server Configuration and Activity

When you run the `start-servers` script, the following WebLogic Server instances are started:

- The Administration Server

- Two server instances in the dynamic cluster named `DynamicCluster`:
  - `DynamicCluster-1`
  - `DynamicCluster-2`

You can use either the WebLogic Server Administration Console or Fusion Middleware Control to view the configuration of `DynamicCluster` and also the number of currently running server instances in the cluster as the demo runs.

For example, to view `DynamicCluster` in the WebLogic Server Administration Console:

1. Launch the WebLogic Server Administration Console by entering the following URL in your browser's address bar:

   ```
   localhost:20001/console
   ```

2. Log in using the WebLogic administrator's credentials that was used to execute the `create-domain` script.
3. In the console home page, click **Servers**.

   The Summary of Servers page is displayed, listing the servers that are currently running and shut down. You can refer to this page as the demo runs to see the state of server instances.

See Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

Using JMeter to Drive the Demo

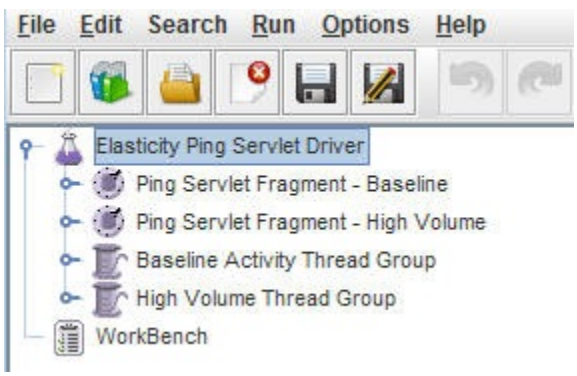
To execute the test plan from JMeter:

1. Open a command prompt and change to the `bin` directory of the JMeter installation. For example, `C:\JMeter\apache-jmeter-2.13\bin`.
2. Enter the following command to start JMeter in GUI mode:

   ```
   jmeter.bat
   ```

3. In the JMeter console, select **File > Open** and select `Elasticity5MinuteDemo.jmx`.
4. From the **Options** menu, make sure that **Log Viewer** is selected.
5. In the navigation pane on the left, make sure **Elasticity Ping Servlet Driver** is selected, shown in Figure 6-1.

Figure 6-1 JMeter Test Navigation Pane



6. Select **Run > Start**.

Using the Monitoring Dashboard to Monitor Scaling Operations

After you start the test plan in JMeter, you can use the WebLogic Server Administration Console Monitoring Dashboard to view the server activity generated by the JMeter test. You will see the number of running server instances ramp up to four in response to the load. When the scenario completes, the additional instances will be scaled down until the minimum number of two running server instances is reached. This entire scenario lasts for five minutes.

To access the Monitoring Dashboard:

1. Launch the WebLogic Server Administration Console, as described in Viewing Dynamic Server Configuration and Activity, if it is not already running.
2. Under Charts and Graphs, click **Monitoring Dashboard**.
3. In the Monitoring Dashboard, click the start button, shown in Figure 6-2.

Figure 6-2 Monitoring Dashboard Start Button

4. As JMeter ramps up the volume of simulated client invocations on the dynamic cluster, scaling policies are triggered that execute scaling actions.

   Notice the throughput of the currently running server instances at the moment new server instances are started. Figure 6-3 shows example output. The graphs you obtain may vary depending on your machine's performance characteristics.

Figure 6-3 Dynamic Server Scaling Activity

In Figure 6-3, notice the following:

- The top four charts show the throughput on the Managed Server instances `DynamicCluster-1`, `DynamicCluster-2`, `DynamicCluster-3`, and `DynamicCluster-4`.

- The server instances `DynamicCluster-1` and `DynamicCluster-2`, shown in the top two charts, show throughput spikes that begin at approximately 2:42:00.
- The chart for `DynamicCluster-3` shows that this third server was started when the throughput spikes occurred on the server instances `DynamicCluster-1` and `DynamicCluster-2`.
- The bottom chart shows the number of active Managed Servers in the cluster, as indicated by the value of the `AliveServerCount` attribute of the `ClusterRuntimeMBean` instances. This chart shows approximately when a third and a fourth server instance are started.
- The third server instance, `DynamicCluster-3`, is started in the interval between 2:42:00 and 2:43:30.
- Notice that a fourth server instance, `DynamicCluster-4`, is started shortly after 2:46:00, to coincide with an increase in throughout on the first three server instances.
- After a new server instance is started, notice the corresponding decrease in throughput on the other server instances.

Re-running and Stopping the Elasticity Demo

You can re-run the elasticity demo from the JMeter GUI, or you can stop the demo and all running server instances by running the stop-servers script as follows:

Windows:

```
DEMO_HOME> stop-servers.cmd
```

Unix:

```
DEMO_HOME> sh stop-servers.sh
```

7

Configuring the Data Source Interceptor

Before a scale up operation is executed in Oracle WebLogic Server, a database interceptor can be automatically invoked to determine the number of database connections that can be created from a WebLogic domain to a database. This interceptor can be used to ensure that the capacities of the available connections are not exceeded as a result of a subsequent scale up operation.The following sections describe how to configure and use the data source interceptor:

- Overview of the Data Source Interceptor
- Configuring Data Source Interceptors

Overview of the Data Source Interceptor

In a scale up operation, new dynamic server instances are started in a dynamic cluster. If any data sources are targeted to this dynamic cluster, then starting a new server instance can result in the creation of more connections to the databases. However, the creation of these additional database connections has the potential to exceed database capabilities. The data source interceptor provides the means to determine whether database capacity would be exceeded by the addition of a new server instance and, if so, to prevent a scale up operation from proceeding.

Before a scale up operation, the data source interceptor determines the maximum number of connections that may be created if additional dynamic server instances are added to the dynamic cluster. If this number is not yet currently met, the interceptor allows the scale up operation to proceed. However, if the number is already met or exceeded, then the interceptor stops the scale up operation from being invoked.

> **Note:**
>
> If the execution of an interceptor fails, then neither the associated scaling operation nor any subsequently configured interceptors are invoked.

Configuring Data Source Interceptors

When you configure a data source interceptor, you specify the database capacities in terms of the total number of connections allowed.Data source interceptors are configured using the `DatasourceInterceptorMBean`. The connection URL specified in a data source interceptor configuration also identifies the database to which the connections are made. It is possible that several databases, with a certain total capacity, map to the same machine that hosts those databases. The data source interceptor configuration identifies a quota corresponding to a group of database URLs, and this quota is the maximum total number of connections that may be created from the domain.
If data sources are targeted to the dynamic cluster that is being scaled up, the data source interceptor will:

- Determine the maximum number of projected connections that may be created on groups of databases if additional dynamic server instances are added to the dynamic cluster.
- Allow the scaling operation to proceed if the configured quota is not exceeded.
- Generate an exception if the configured quota is exceeded, which prevents the scaling operation from proceeding.

Example 7-1 shows a sample configuration for a data source interceptor in the `config.xml` file.

Example 7-2 shows the WLST commands to create and configure a new data source interceptor.

Example 7-1 Sample Configuration for a Data Source Interceptor (in CONFIG.XML)

```
<interceptors>
   <interceptor xsi:type="datasource-interceptorType">
     <name>datasource-interceptor-1</name>
     <priority>1073741823</priority>
     <connection-urls-pattern>jdbc:derby://host:1527/(.*)
     </connection-urls-pattern>
     <connection-quota>20</connection-quota>
   </interceptor>
 </interceptors>
```

Example 7-2 Creating a Data Source Interceptor Using WLST

```
startEdit()

cd('/Interceptors/wl_server')
cmo.createDatasourceInterceptor('datasource-interceptor-1')

cd('/Interceptors/wl_server/Interceptors/datasource-interceptor-1')
cmo.setPriority(1073741823)
cmo.setConnectionQuota(20)
cmo.setConnectionUrlsPattern(jdbc:derby://host:1527/(.*))

activate()
save()
```

You can also create and configure data source interceptors using the WebLogic Server Administration Console. See Create data source interceptors and Configure data source interceptors in the *Oracle WebLogic Server Administration Console Online Help*.

8

Configuring the Script Interceptor

In Oracle WebLogic Server, you can configure and use script interceptors in conjunction with elastic scaling operations performed in dynamic clusters.The following sections describe how to configure the script interceptor:

Overview of the Script Interceptor

Script interceptors are provided as a means to perform tasks before or immediately after a scaling operation has been completed.During a scaling operation, it is necessary to perform functions with various entities or systems in the WebLogic Server environment. For example, before starting a new server instance in a dynamic cluster, it is necessary to coordinate with a virtualization manager to provision a new VM or to configure a machine.
Script interceptors can execute two main types of scripts:

- Preprocessor scripts execute tasks prior to a scaling operation
- Postprocessor scripts execute tasks immediately after a scaling operation

Script interceptors can also execute error handler scripts if an error occurs during the execution of the preprocessor or postprocessor scripts. And multiple script interceptor instances can be used in an interceptor chain for a scaling operation.

> **Note:**
>
> If the execution of a preprocessor script fails, then neither the associated scaling operation nor any subsequently configured interceptors are invoked. If a postprocessor script fails, the scaling action that preceded the invocation of that script cannot be reverted or otherwise cancelled.

Script interceptors are configured using the `ScriptInterceptorMBean`. Example 8-1 offers a sample script interceptor configuration.

Example 8-1 Configuring a Script Interceptor

```
EditServiceMBean editService = mbsFactory.getEditService();
ConfigurationManagerMBean configManager = editService.getConfigurationManager();
configManager.startEdit(5000, 5000);

DomainMBean domainMBean = editService.getDomainConfiguration();
InterceptorsMBean interceptors = (InterceptorsMBean) domainMBean.getInterceptors();
```

```
ScriptInterceptorMBean scriptInterceptor1 =
interceptors.createScriptInterceptor("script-1");
scriptInterceptor1.setPriority(50);
PreProcessorScriptMBean pre1 = scriptInterceptor1.getPreProcessor();
pre1.setWorkingDirectory(domainMBean.getRootDirectory());
pre1.setPathToScript("/bin/sh");
pre1.setArguments(new String[] {domainMBean.getRootDirectory() + "/scripts/
interceptors/echo-arg-interceptor.sh", "first"});
Properties envAsProps = new Properties();
envAsProps.put("env-key-1", "env-1-value");
envAsProps.put("env-key-2", "env-2-value");
envAsProps.put("env-key-3", "env-3-value");
pre1.setEnvironment(envAsProps);

ScriptInterceptorMBean scriptInterceptor2 =
interceptors.createScriptInterceptor("script-2");
scriptInterceptor2.setPriority(20);
PreProcessorScriptMBean pre2 = scriptInterceptor2.getPreProcessor();
pre2.setWorkingDirectory(domainMBean.getRootDirectory());
pre2.setPathToScript("/bin/sh");
pre2.setArguments(new String[] {domainMBean.getRootDirectory() + "/scripts/
interceptors/echo-arg-interceptor.sh", "second"});

configManager.save();
configManager.activate(5000);
```

You can also create and configure script interceptors using the WebLogic Server Administration Console. See Create script interceptors and Configure general settings for a script interceptor in the *Oracle WebLogic Server Administration Console Online Help*.

Configuring Preprocessor Scripts

Preprocessor scripts are executed by the script interceptor before a scaling operation has begun.Note that if the execution of a preprocessor script fails, neither any subsequent preprocessor script nor the corresponding scaling operation is invoked.
For more information about configuring preprocessor scripts for a script interceptor using the WebLogic Server Administration Console, see Configure preprocessor script settings for a script interceptor in the *Oracle WebLogic Server Administration Console Online Help*.

Configuring Postprocessor Scripts

Post-processor scripts are executed by the script interceptor after a scaling operation has been completed. Note that if the execution of a postprocessor script fails, the scaling operation that preceded it cannot be cancelled.
For more information about configuring postprocessor scripts for a script interceptor using the WebLogic Server Administration Console, see Configure post-processor script settings for a script interceptor in the *Oracle WebLogic Server Administration Console Online Help*.

Configuring Error Handling for a Script Interceptor

When configuring a preprocessor or postprocessor script for a script interceptor, you can specify an error handler script to use if an error occurs during the execution of the main script.

The error handler script is executed using the same arguments and environment properties as the command script.

Reserved Environment Variables

Script interceptors can pass both computed and dynamic parameters as environment variables to scripts. In addition, WLDF includes a set of reserved environment variables that can be used in scripts. These reserved environment variables are listed and described in the following table. Note that with the exception of the first variable in this table, `WLS_SCRIPT_THIS_STEP_FAILED`, each of these variables can be passed to preprocessor scripts, postprocessor scripts, and error handler scripts.

Table 8-1 Reserved Environment Variables

Environment Variable	Description
`WLS_SCRIPT_THIS_STEP_FAILED`	This variable is passed to error-handler scripts only. Value 'false' indicates some subsequent step failed. • A value of `true` indicates that current step failed. • A value of `false` indicates that a subsequent step failed.
`WLS_SCRIPT_OUTPUT_FILE`	Represents the path to the output properties file that scripts may create. The contents of the file will be read and passed to subsequent scripts.
`WLS_SCRIPT_TEMP_DIR`	Represents the path to the temporary directory in which scripts may create temporary files.
`WLS_SCRIPT_DYNAMIC_CLUSTER_NAME`	Represents the name of the dynamic cluster that is being scaled up or down.
`WLS_SCRIPT_DYNAMIC_CLUSTER_OPERATION_NAME`	Represents the current scaling operation, which is either `scaleUp` or `scaleDown`.
`WLS_SCRIPT_DYNAMIC_CLUSTER_MIN_SIZE`	Represents the minimum size of the dynamic cluster.
`WLS_SCRIPT_DYNAMIC_CLUSTER_MAX_SIZE`	Represents the maximum size of the dynamic cluster.
`WLS_SCRIPT_DYNAMIC_CLUSTER_SIZE`	Represents the current size of the dynamic cluster.
`WLS_SCRIPT_DYNAMIC_CLUSTER_CANDIDATE_MEMBER_NAMES`	Represents the names of candidate servers that may be chosen to be started or stopped.
`WLS_SCRIPT_DYNAMIC_CLUSTER_REQUESTED_SCALING_SIZE`	Represents the number of running servers in the cluster that are requested to be either incremented or decremented.
`WLS_SCRIPT_DYNAMIC_CLUSTER_ALLOWED_SCALING_SIZE`	Represents the actual number of servers that will be started or stopped (which may be smaller than the requested scaling size).
`WLS_SCRIPT_DYNAMIC_CLUSTER_SCALED_MEMBER_NAMES`	Represents the names of the servers that are chosen to be started or stopped during scale up or scale down operations.