Oracle® WebLogic Operations Control

Plain Agent Use Case Example

10*g* Release 3 (10.3)

September 2008

ORACLE

Oracle WebLogic Operations Control Plain Agent Use Case Example, 10*g* Release 3 (10.3)

Contents

Deploy the Service Against Available Resources

Monitor WLOC Services and Resources

CHAPTER 1

Introduction

Oracle WebLogic Operations Control (WLOC) is a management environment that can increase the efficiency of your operations center by hiding the complexity of the underlying operational environment and presenting the resources and Java applications in a simple supply-and-demand mode.

On the supply side of the equation, you use WLOC to organize the computing resources in your operations center into collections (pools) of resources. A WLOC resource pool can represent a single physical machine or a collection of virtualized resources that are made available through hypervisor software.

On the demand side of the equation, you use WLOC to organize Java applications (processes) into WLOC services. Typically, you organize a group of related processes into a single service and manage the group as a unit, but you can also create one service for each process.

A typical WLOC deployment contains a single WLOC Controller—the centralized component that gathers data about the operating environment—and multiple Agents that manage and monitor resources and communicate that information back to the Controller. A Plain Agent manages the computing resources for the physical machine on which the Agent is installed. The Controller hosts the WLOC Administration Console that enables you to visually configure, manage, and monitor the WLOC environment.

This document provides a basic use-case example for WLOC. In this use case we will describe the steps to:

- Establish a resource environment by installing and creating a Plain Agent and Controller.

- Establish the runtime environment by starting the Agent, Controller, and the WLOC Administration Console.
- Use the Administration Console to define a service to be managed.
- Deploy the service against the available resources.
- Monitor the WLOC service and the resource environment with the WLOC Administration Console.

Main Steps

The following table summarizes the main steps demonstrated in this example.

Table 1-1 Use Case Example Main Steps

To complete this task . . .	We demonstrate how to perform the following steps . . .
Establish the WLOC resource environment	• "Step 1: Install WLOC" on page 2-2 • "Step 2: Create the Plain Agent" on page 2-2 • "Step 3: Create the Controller" on page 2-12
Establish the WLOC runtime environment	• "Step 1: Start the Plain Agent" on page 3-1 • "Step 2: Start the Controller" on page 3-2 • "Step 3: Start the WLOC Administration Console" on page 3-2
Define services under management	• "Step 1: Create the Service and Process Groups" on page 4-3 • "Step 2: Define the Adaptive Runtime Policies" on page 4-25
Deploy services against available resources	• "Deployment Scenario" on page 5-2 • "Deploy the Service" on page 5-3
Monitor WLOC services and resource environment with the WLOC Administration Console	• "Create a View" on page 6-2 • "Browse the Resources Pane" on page 6-2

Related Documents

The WLOC documentation set includes the following:

- *Installation Guide*—Describes how to install and uninstall the WLOC components.

- *Plain Agent Use Case Example*—This document.
- *Configuration Guide*—Describes how to configure and manage the WLOC Controller and Agents, configure services and policies to manage services, and configure security. It also describes how to use WLOC to monitor, log, and audit the operations of your services and resources.
- *Securing a Production Environment*—Highlights essential security measures for you to consider before you deploy WLOC into a production environment.
- *WLOC Administration Console Help*—The online help for WLOC's graphical user interface. You can access the WLOC Administration Console Help either by clicking the Help link in the upper right corner of the Administration Console, or at `http://download.oracle.com/docs/cd/E13156_01/wloc/docs103/ConsoleHelp`.
- *Controller Configuration Schema Reference*—A reference to the XML schema used to persist the configuration of the WLOC Controller component.
- *Agent Configuration Schema Reference*—A reference to the XML schema used to persist the configuration of the WLOC Agent component.
- *Service Metadata Schema Reference*—A reference to the XML schema used to persist the configuration of WLOC services.
- *Message Catalog*—A reference to messages generated by WLOC.

CHAPTER 2

Configure the WLOC Resource Environment

In a WLOC environment, resource pools provide a virtual environment in which you can deploy WLOC services. Each resource pool provides access to physical computing resources (such as CPU cycles, memory, and disk space) and pre-installed software that a service needs to run.

To establish a WLOC resource environment, you need to configure a controller and one or more agents. You can do so using the WLOC Configuration Wizard. When you configure an Agent, you configure its resource pool. A Plain Agent manages the computing resources for the physical machine on which the Agent is installed.

When you configure the Controller, you bind it to the Agents so that it can get information about the resources and deploy services accordingly.

The Controller also hosts the WLOC Administration Console, which provides a graphical interface into the WLOC environment.

In this example, we will install and create a Plain Agent and a Controller on the same local Windows machine. For instructions about installing WLOC and starting the Configuration Wizard on UNIX or Linux platforms, see the following:

- *WLOC Installation Guide*.
- "Configuring the Controller and Agents" in the *Configuration Guide*

The main steps in this topic include:

- Step 1: Install WLOC
- Step 2: Create the Plain Agent

- Step 3: Create the Controller

Step 1: Install WLOC

In this example, we will install the Plain Agent and the Controller on the same local Windows machine. When you perform a Complete installation, all of the WLOC components are installed automatically. For details about installing WLOC, see the *WLOC Installation Guide*.

If the Plain Agent is managing resources on a different machine, be sure to install the Agent software on that machine.

In this example, we install WLOC into the `C:\WLOC_UseCase` directory.

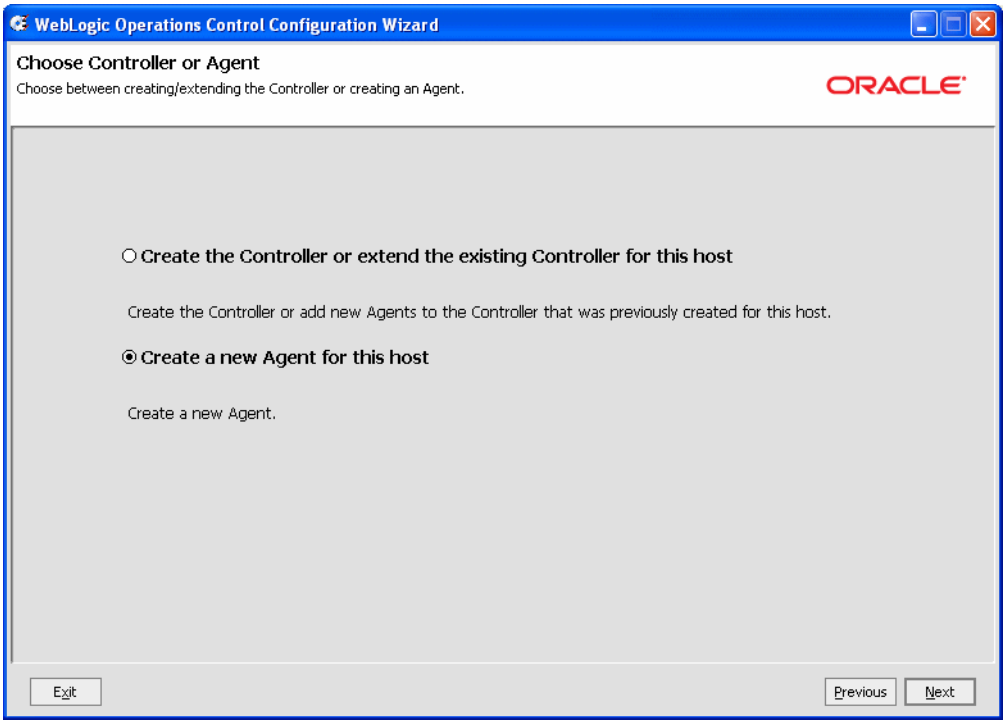
After you install the agent and the controller software, you create them using the WLOC Configuration Wizard.

Step 2: Create the Plain Agent

To create the Plain Agent, complete the following steps:

1. From the Start Menu, select Start > All Programs > WebLogic Operations Control 10gR3 > WLOC Configuration Wizard.

2. In the **Welcome** window, click **Next**.
3. In the **Choose Controller or Agent** window, select **Create a new Agent for this host** and click **Next**.

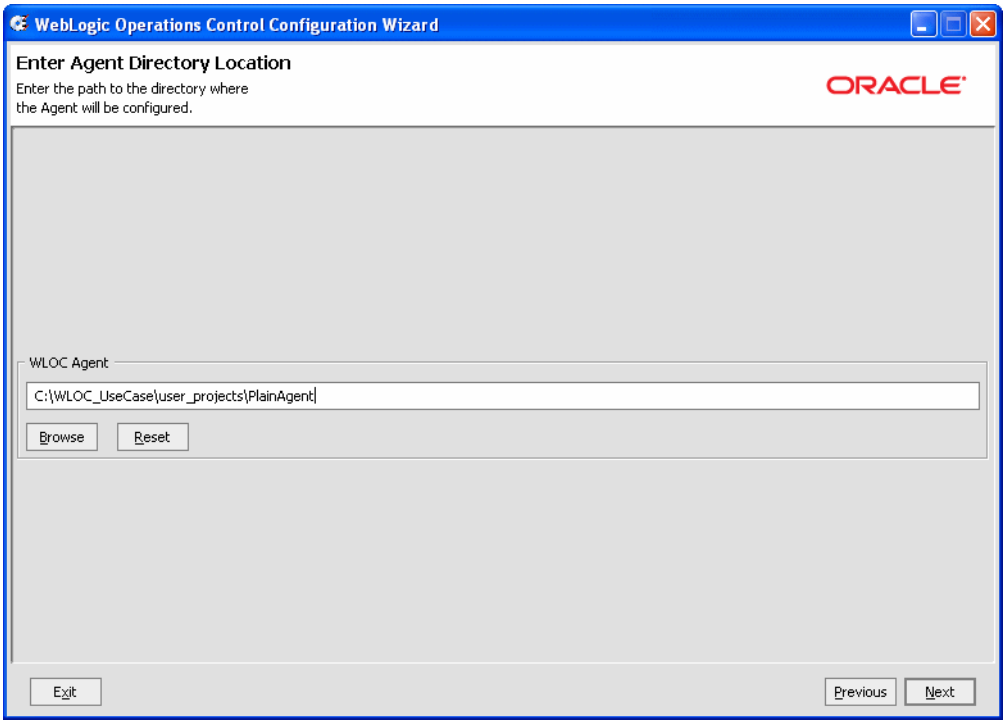


4. In the **Enter Agent Directory Location** window, specify the path to the directory for the Agent and click **Next**.

   By default, this directory is created in *BEA_HOME*\user_projects\agent1, but you can specify any name and directory location you choose.

   Note that we used C:\WLOC_UseCase as the *BEA_HOME* directory when we installed the WLOC software, therefore that *BEA_HOME* value is displayed as the default.

   For this example, we accept the default C:\WLOC_UseCase\user_projects location, and change the name of the directory to **PlainAgent**.



5. In the **Configure Agent Connection Details** window, specify the connection information shown in Table 2-1 for the Plain Agent.

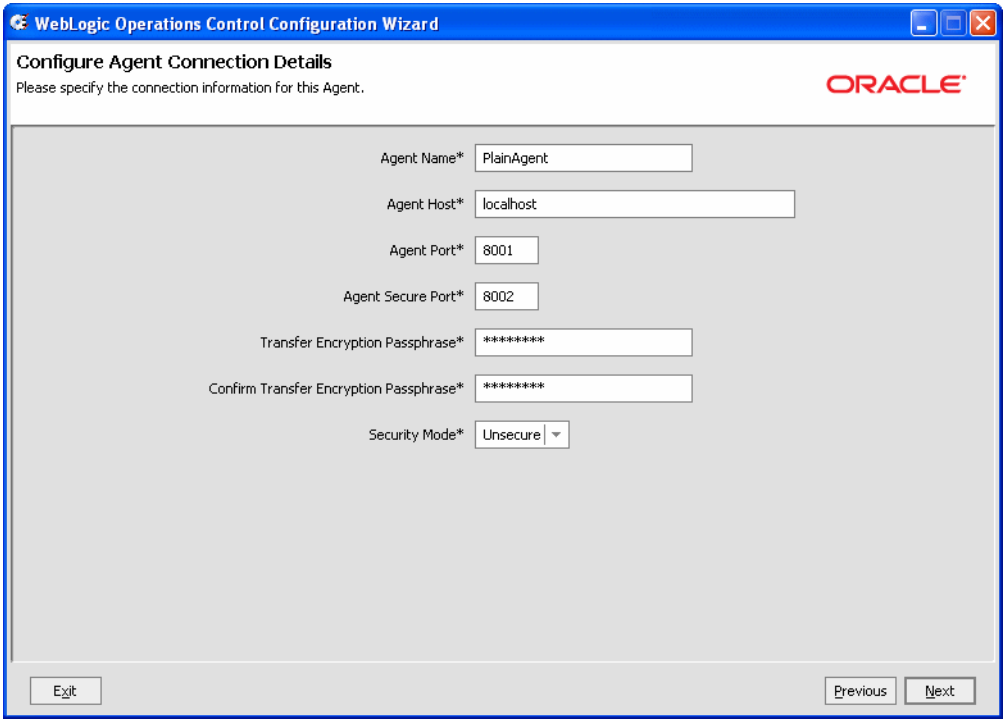
   Note: In this example, we do not need to select the Secure setting for the Security mode field because we do not need to use SSL between the Agent and the Controller. However, in a production environment, you need to use the secure settings, and set the passwords and passphrases accordingly. For details about secure communications, see "Configuring Security" in the *Configuration Guide*.

Table 2-1 Agent Connection Information

In this field . . .	Enter the following value . . .
Agent Name	PlainAgent
Agent Host	The URL for the host machine. In this example we use `localhost`

Table 2-1 Agent Connection Information (Continued)

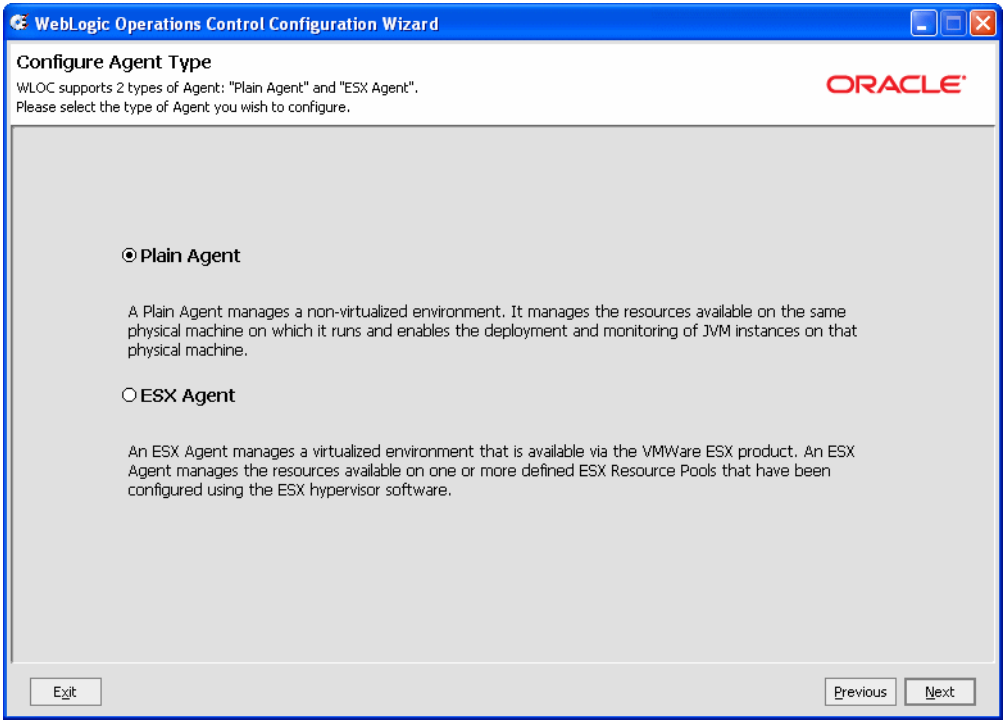
In this field . . .	Enter the following value . . .
Agent Port	8001 (the default)
Agent Secure Port	8002 (the default)
Transfer Encryption Passphrase	Default
Confirm Transfer Encryption Passphrase	Default
Security Mode	Unsecure (default)



6. Click **Next** in the following two windows to accept the defaults:
   - **Configure Agent Logging**

– **Configure Agent Keystore Passwords**

7. In the **Configure Agent Type** window, select **Plain Agent** and click **Next**.



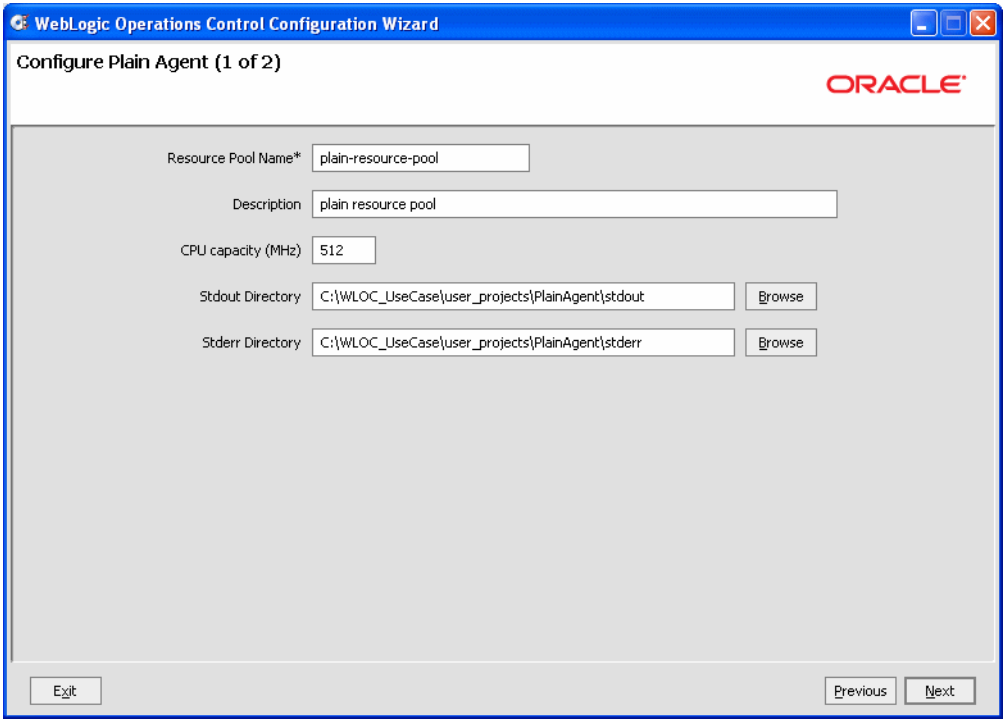
8. In the **Configure Plain Agent (1 of 2)** window, provide a name for the resource pool associated with this Agent and the CPU capacity available to the resource pool, as shown in the following table:

Table 2-2 Plain Agent Resource Pool Configuration

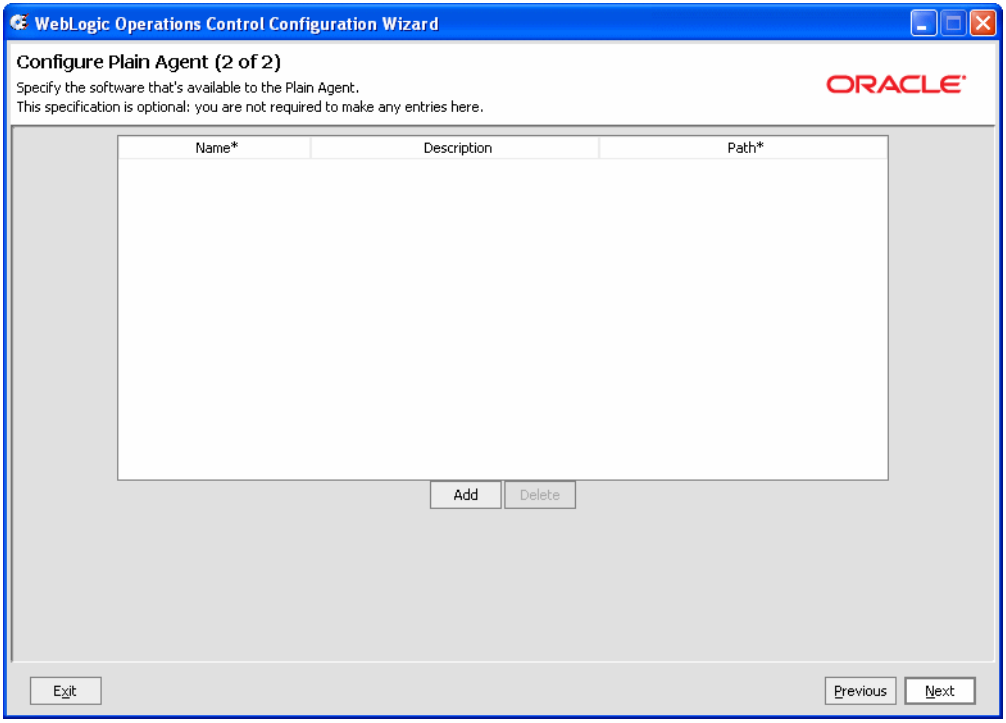
In this field . . .	Enter the following value . . .
Resource Pool Name	plain-resource-pool
Description	plain resource pool
CPU capacity (MHz)	The CPU capacity for your machine. In this example we specify 512.

Table 2-2 Plain Agent Resource Pool Configuration (Continued)

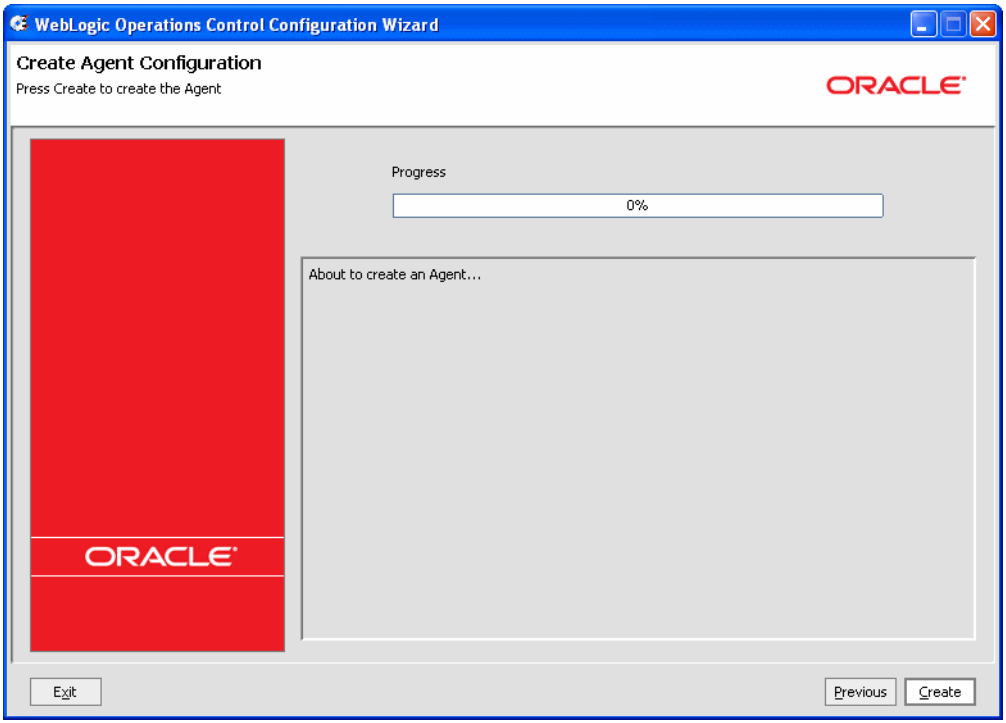
In this field . . .	Enter the following value . . .
Stdout Directory	Accept the default
Stderr Directory	Accept the default



9. In the **Configure Plain Agent (2 of 2)** window, you specify the available software you want to include in the resource pool. For this example, do not specify any additional software and click **Next**.



10. In the **Create Agent Configuration** window, click **Create**.



11. After the Agent has been created, click **Done** to exit the WLOC Configuration Wizard.

Agent Directory Structure

After completing the Plain Agent installation and creation, the following directory structure is created in the `C:\WLOC_UseCase\user_projects\PlainAgent` directory.

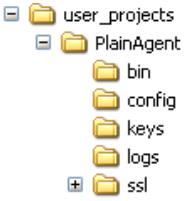


Table 2-2 describes the contents of these directories.

Table 2-3 Agent Directory Description

Directory	Description
`bin`	Commands to start the Agent, and to install and remove the Agent as a Windows service.
`config`	Agent configuration files.
`keys`	Encryption key used to encrypt clear text passwords.
`logs`	Agent log files.
`ssl`	Internal digital certificate and keystores for the Agent used for SSL communication with the Controller.

Agent Configuration File

When you create an Agent using the WLOC Configuration Wizard, the configuration is persisted in an XML file named `loc-agent-config.xml`. In this example, the file is created in the following directory:

`C:\WLOC_UseCase\user_projects\PlainAgent\config`

where:

`WLOC_UseCase` is the BEA Home directory containing the WLOC installation, and `PlainAgent` is the name that we specified for the Agent Directory location in the Configuration Wizard.

After you have created the Agent using the Configuration Wizard, it can be modified using the Administration Console or by directly editing its configuration file.

The `loc-agent-config.xml` file created in this example is shown in Listing 2-1

Listing 2-1 Sample loc-agent-config.xml File

```
<?xml version="1.0" encoding="UTF-8"?><loc-agent xmlns="bea.com/loc/agent"
xmlns:loc="http://bea.com/loc">
  <name>PlainAgent</name>
  <description>PlainAgent</description>
  <network>
    <loc:host>localhost</loc:host>
    <loc:components>
      <loc:component>
```

```
        <loc:name>ListenPorts</loc:name>
        <loc:description>ListenPorts</loc:description>
        <loc:port>8001</loc:port>
        <loc:secure-port>8002</loc:secure-port>
      </loc:component>
    </loc:components>
  </network>
  <use-secure-connections>false</use-secure-connections>
  <logging>
    <loc:file-severity>Info</loc:file-severity>

<loc:base-file-name>C:\WLOC_UseCase/user_projects/PlainAgent/logs/Agent.log</l
oc:base-file-name>
    <loc:rotation-type>BySize</loc:rotation-type>
    <loc:rotation-size>5000</loc:rotation-size>
    <loc:rotation-time>00:00</loc:rotation-time>
    <loc:file-rotation-dir>./logs/logrotdir</loc:file-rotation-dir>
    <loc:number-of-files-limited>true</loc:number-of-files-limited>
    <loc:rotated-file-count>5</loc:rotated-file-count>
    <loc:rotation-time-span>24</loc:rotation-time-span>
    <loc:rotation-time-span-factor>3500000</loc:rotation-time-span-factor>
    <loc:rotation-on-startup-enabled>true</loc:rotation-on-startup-enabled>
    <loc:stdout-severity>Info</loc:stdout-severity>
  </logging>
  <audit>
    <loc:base-file-name>./logs/audit.log</loc:base-file-name>
    <loc:rotation-type>BySize</loc:rotation-type>
    <loc:rotation-size>300</loc:rotation-size>
    <loc:rotation-time>00:00</loc:rotation-time>
    <loc:file-rotation-dir>./logs/logrotdir</loc:file-rotation-dir>
    <loc:number-of-files-limited>true</loc:number-of-files-limited>
    <loc:rotated-file-count>50</loc:rotated-file-count>
    <loc:rotation-time-span>24</loc:rotation-time-span>
    <loc:rotation-time-span-factor>50</loc:rotation-time-span-factor>
    <loc:rotation-on-startup-enabled>true</loc:rotation-on-startup-enabled>
    <loc:enabled>true</loc:enabled>
    <loc:scope>
      <loc:type>All</loc:type>
    </loc:scope>
  </audit>
  <work-managers>
    <loc:work-manager>
      <loc:name>WM</loc:name>
      <loc:description>WM</loc:description>
      <loc:max-threads-constraint>64</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
    <loc:work-manager>
      <loc:name>ResourceBrokerAgent-WM</loc:name>
```

```
      <loc:description>ResourceBrokerAgent-WM</loc:description>
      <loc:max-threads-constraint>15</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
    <loc:work-manager>
      <loc:name>AgentRuntime-WM</loc:name>
      <loc:description>AgentRuntime-WM</loc:description>
      <loc:max-threads-constraint>15</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
  </work-managers>
  <encryption>
    <password>{Salted-3DES}8U+L9mNDmAFoUayTkWaFOQ==</password>
  </encryption>
  <resource-pools>
    <plain-resource-pool>
      <name>resource-resource-pool</name>
      <description>plain resource pool</description>
      <cpu-capacity>512</cpu-capacity>
      <stdout-dir>C:\WLOC_UseCase\user_projects\PlainAgent\stdout</stdout-dir>
      <stderr-dir>C:\WLOC_UseCase\user_projects\PlainAgent\stderr</stderr-dir>
    </plain-resource-pool>
  </resource-pools>
</loc-agent>
```

For information about the elements of the `loc-agent-config.xml` Agent configuration file, see the *Agent Configuration Schema Reference*.

Step 3: Create the Controller

Every WLOC environment includes a single Controller and one or more Agents. The Controller is the central component that gathers data about the operating environment from Agents. The Controller uses the data that it gathers to intelligently deploy new services and to evaluate and enforce policies for all services in the environment. The Controller also hosts the WLOC Administration Console.

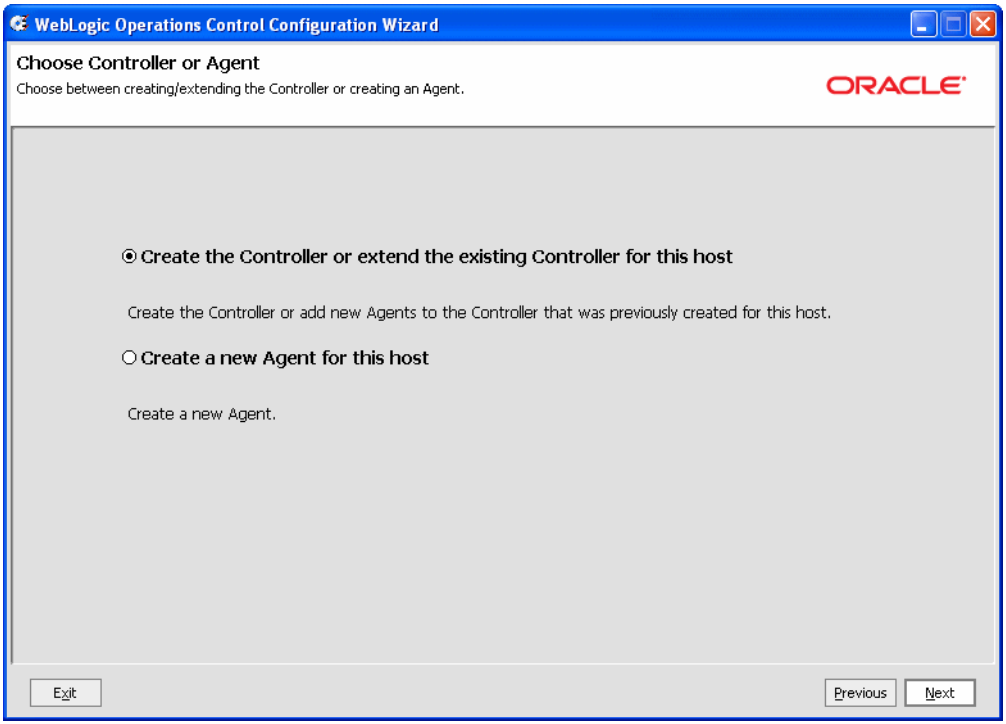
Although you can install the Agent and the Controller on different physical machines, in this example, we installed the Controller as part of a complete installation on the local machine as described in "Step 1: Install WLOC" on page 2-2. After the controller is installed, you create it using the WLOC Configuration WIzard.

To create the Controller, complete the following steps:

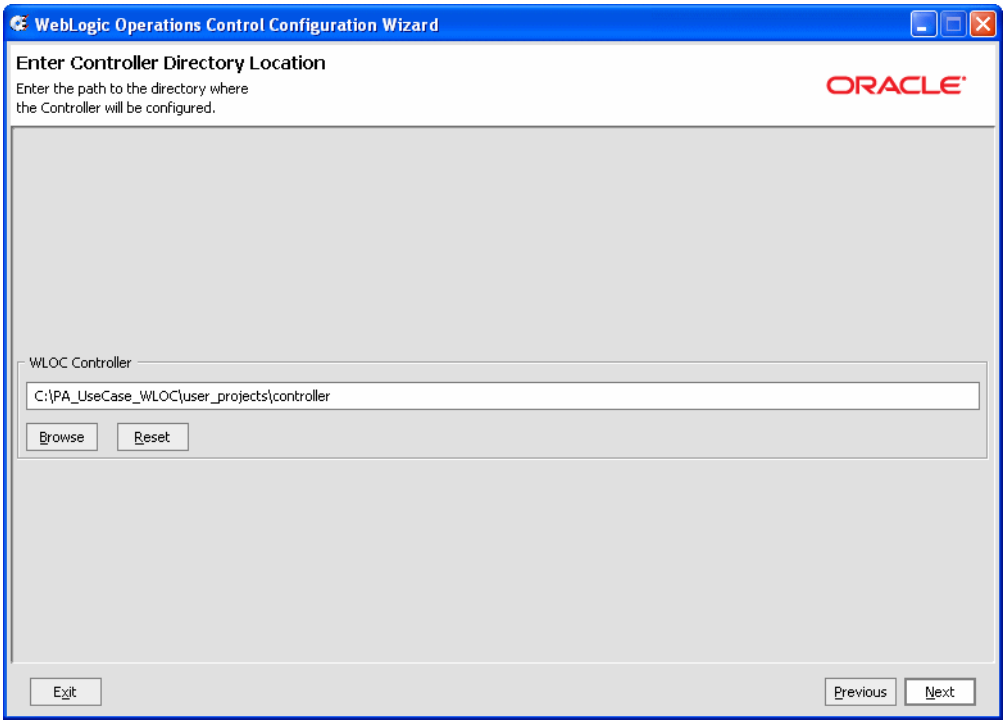
1. From the Start Menu, select Start > All Programs > WebLogic Operations Control 10gR3 > WLOC Configuration Wizard.

2. In the **Welcome** window, click **Next**.

3. In the **Choose Controller or Agent** window, select **Create the Controller or extend the existing Controller for this host** and click **Next**.



4. In the **Enter Controller Directory Location** window, we accept the default path and filename for the Controller and click **Next**.



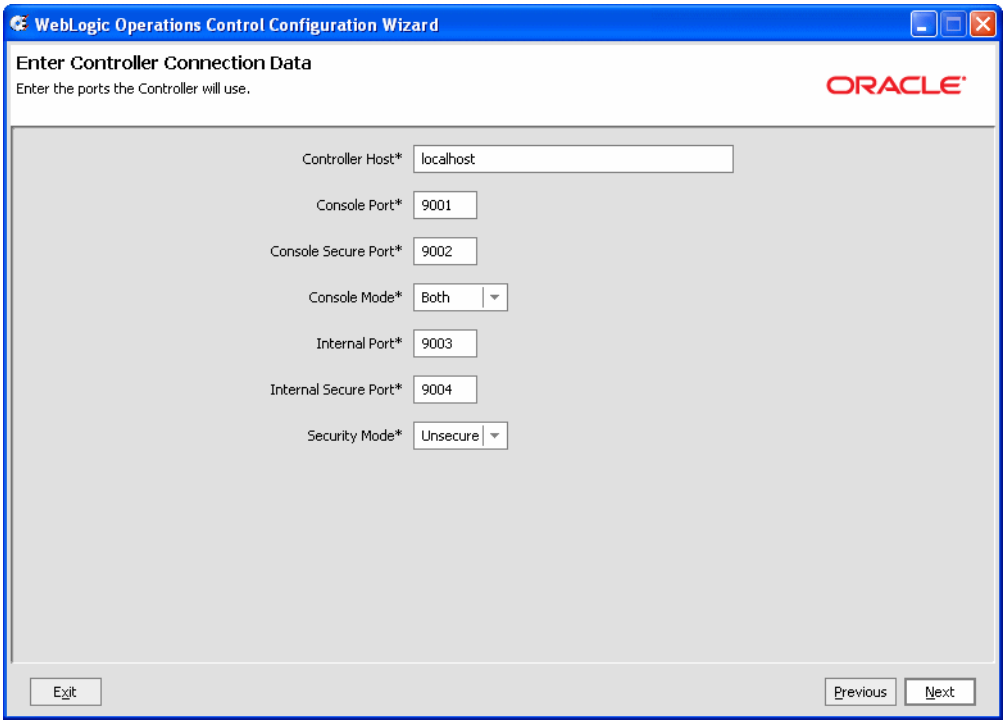
5. In the **Enter Controller Connection Data** window, specify the following connection information for the Controller.

Table 2-4 Controller Connection Information

In this field . . .	Enter the following value . . .
Controller Host	localhost
Console Port	9001 (the default)
Console Secure Port	9002 (the default)
Console Mode	Both
Internal Port	9003 (the default)

Table 2-4 Controller Connection Information (Continued)

In this field . . .	Enter the following value . . .
Internal Secure Port	9004 (the default)
Security Mode	Unsecure (default)



6. Accept the default options in the following windows and click **Next**:
   - **Configure Controller Logging**
   - **Configure Controller Notifications (1 of 3)**
   - **Configure Controller Notifications (2 of 3)**
   - **Configure Controller Notifications (3 of 3)**

7. In the **Configure Agents for this Controller** window, click **Add** to bind the Plain Agent we created to this Controller. The fields are populated with the default data for your machine. Edit the fields as follows:

   Note: To edit a field, you need to double-click the value in the field.

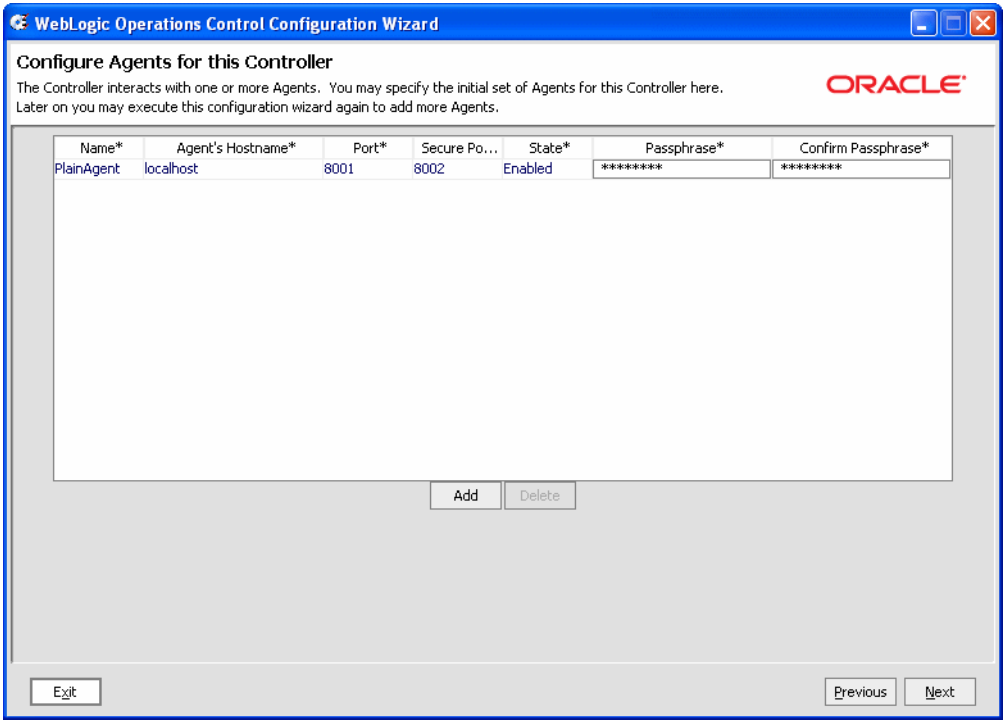
   a. Enter **PlainAgent** in the **Name** field.

   b. Enter **localhost** in the **Agent's Hostname** field.

   c. Accept the defaults for the remaining fields.

   d. Click **Next**.

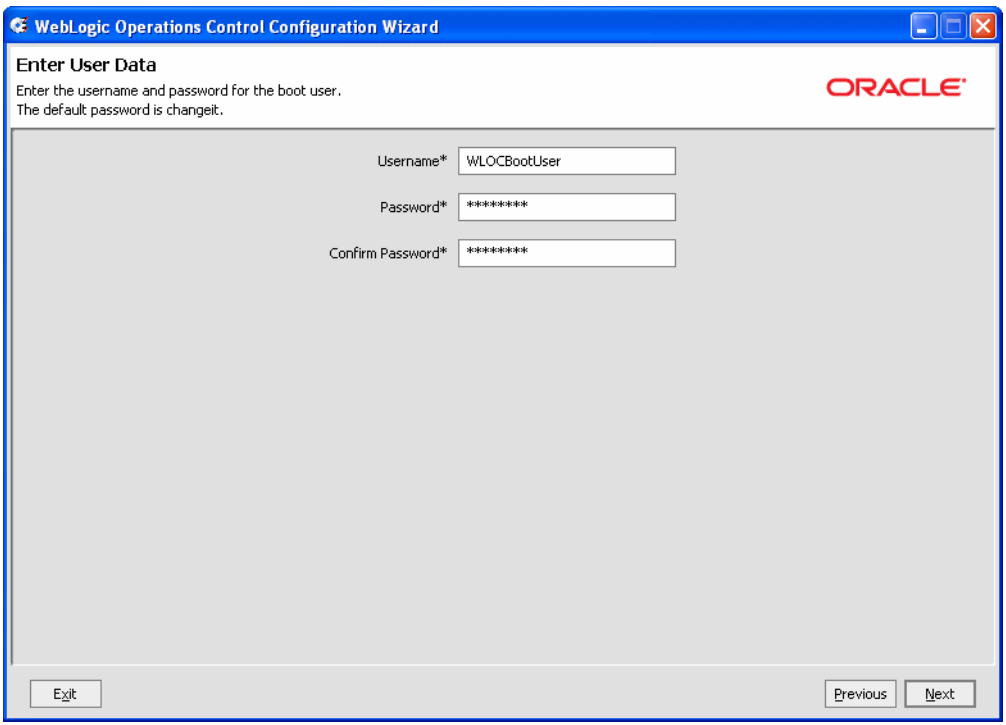
   Note that the passphrase specified here must match the passphrase that you defined when you created the Agent. Because we accepted the default when we created the Agent, we can accept the same default here.



8. Click **Next** in the **Use SSH for WLOC ESX Agents** window. In this example, only a Plain Agent is configured.

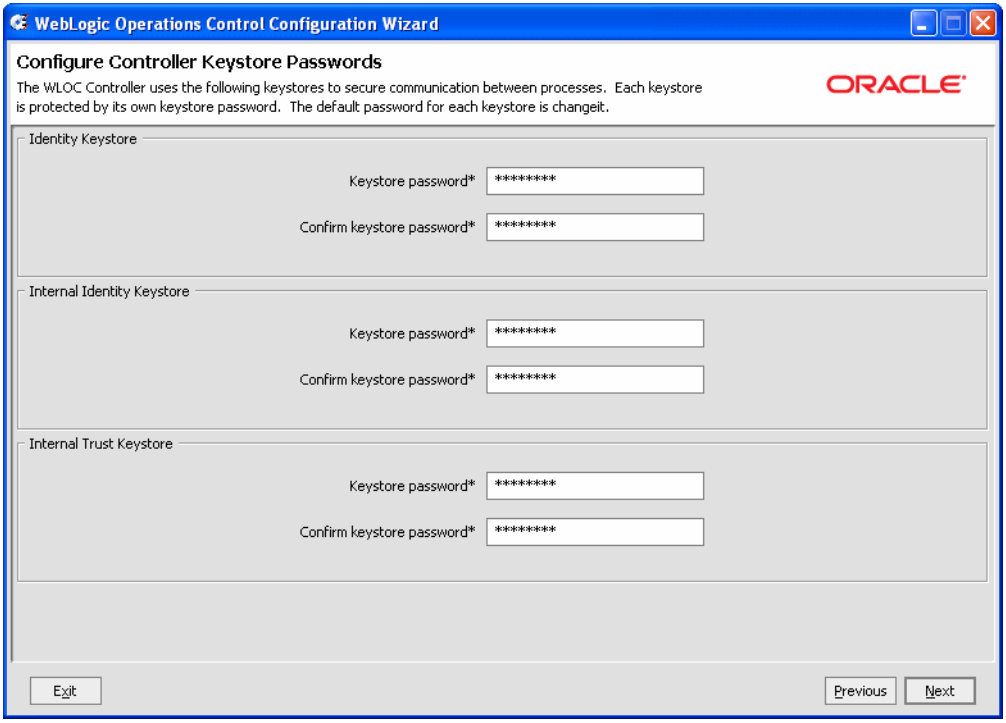
9. In the **Enter User Data** window, specify a username and password for the boot user. For this example, accept the defaults.

   Note: The default username is **WLOCBootUser** and the default password is **changeit**:

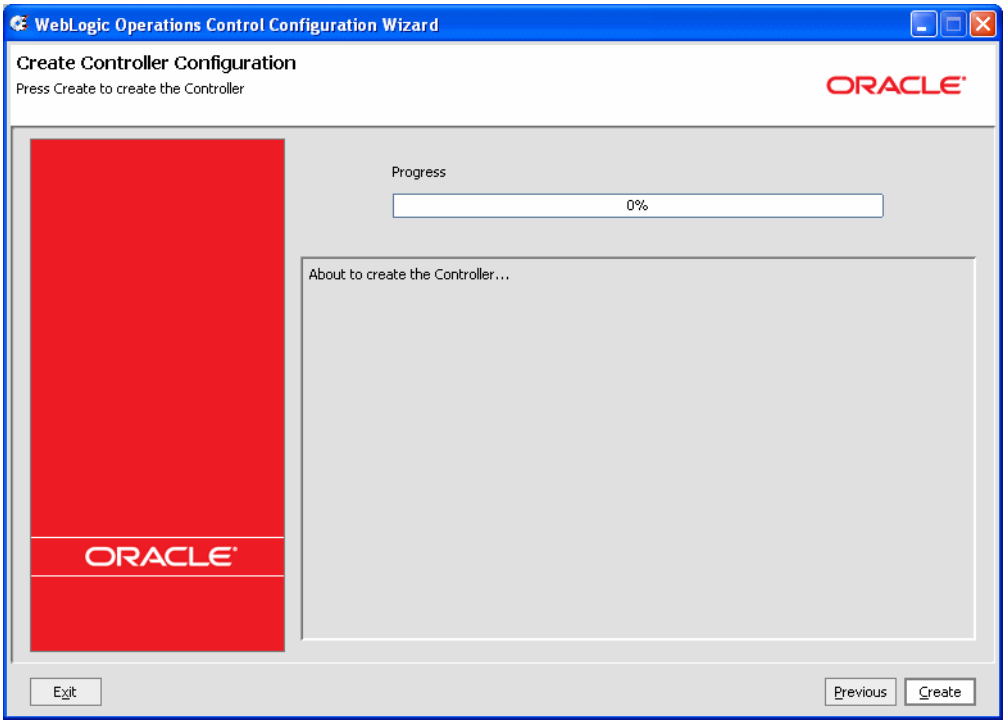


10. In the **Configure Controller KeyStore Passwords**, accept the default passwords and click **Next**.

    Note: The default passwords are **changeit**.



11. In the **Create Controller Configuration** window, click **Create**.



12. After the Controller has been created, click **Done** to exit the WLOC Configuration Wizard.

Controller Directory Structure

After completing the Controller installation and creation, the following directory structure is created in the `C:\WLOC_UseCase\user_projects\controller` directory.

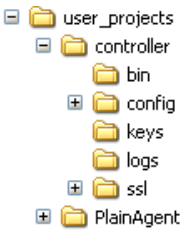


Table 2-2 describes the contents of these directories.

Table 2-5 Controller Directory Description

Directory	Description
`bin`	Commands to start the Controller, and to install and remove the Controller as a Windows service.
`config`	Controller configuration files.
`keys`	Encryption keys used to encrypt clear text passwords and data.
`logs`	Controller log files.
`ssl`	Internal digital certificate and keystores for the Controller used for SSL communication with the Agents.

Controller Configuration File

When you create a Controller using the WLOC Configuration Wizard, the configuration is persisted in an XML file named `loc-controller-config.xml`. In this example, the file is created in the following directory:

`C:\WLOC_UseCase\user_projects\controller\config`

where:

`WLOC_UseCase` is the BEA Home directory containing the WLOC installation, and `controller` is the name that we specified for the Controller Directory location in the Configuration Wizard.

After you have created the Controller using the Configuration Wizard, it can be modified using the Administration Console or by directly editing its configuration file.

The `loc-controller-config.xml` file created in this example is shown in Listing 2-2.

Listing 2-2 Sample loc-controller-config.xml File

```
<?xml version="1.0" encoding="UTF-8"?><loc-controller
xmlns="bea.com/loc/controller" xmlns:loc="http://bea.com/loc">
  <network>
    <loc:host>localhost</loc:host>
    <loc:components>
      <loc:component>
        <loc:name>Console</loc:name>
        <loc:description>Console</loc:description>
```

```
        <loc:port>9001</loc:port>
        <loc:secure-port>9002</loc:secure-port>
      </loc:component>
      <loc:component>
        <loc:name>InternalCommunication</loc:name>
        <loc:description>InternalCommunication</loc:description>
        <loc:port>9003</loc:port>
        <loc:secure-port>9004</loc:secure-port>
      </loc:component>
    </loc:components>
  </network>
  <use-secure-connections>false</use-secure-connections>
  <console-mode>BOTH</console-mode>
  <logging>
    <loc:file-severity>Info</loc:file-severity>

<loc:base-file-name>C:/WLOC_UseCase/user_projects/controller/logs/Controller.l
og</loc:base-file-name>
    <loc:rotation-type>BySize</loc:rotation-type>
    <loc:rotation-size>500</loc:rotation-size>
    <loc:rotation-time>00:00</loc:rotation-time>
    <loc:file-rotation-dir>./logs/logrotdir</loc:file-rotation-dir>
    <loc:number-of-files-limited>true</loc:number-of-files-limited>
    <loc:rotated-file-count>5</loc:rotated-file-count>
    <loc:rotation-time-span>24</loc:rotation-time-span>
    <loc:rotation-time-span-factor>3500000</loc:rotation-time-span-factor>
    <loc:rotation-on-startup-enabled>true</loc:rotation-on-startup-enabled>
    <loc:stdout-severity>Info</loc:stdout-severity>
  </logging>
  <audit>
    <loc:base-file-name>./logs/audit.log</loc:base-file-name>
    <loc:rotation-type>BySize</loc:rotation-type>
    <loc:rotation-size>300</loc:rotation-size>
    <loc:rotation-time>00:00</loc:rotation-time>
    <loc:file-rotation-dir>./logs/logrotdir</loc:file-rotation-dir>
    <loc:number-of-files-limited>true</loc:number-of-files-limited>
    <loc:rotated-file-count>50</loc:rotated-file-count>
    <loc:rotation-time-span>24</loc:rotation-time-span>
    <loc:rotation-time-span-factor>50</loc:rotation-time-span-factor>
    <loc:rotation-on-startup-enabled>true</loc:rotation-on-startup-enabled>
    <loc:enabled>true</loc:enabled>
    <loc:scope>
      <loc:type>ControllerConfiguration</loc:type>
      <loc:type>ServiceConfiguration</loc:type>
      <loc:type>Rules</loc:type>
      <loc:type>ControllerAction</loc:type>
      <loc:type>Adjudication</loc:type>
      <loc:type>AgentConfiguration</loc:type>
    </loc:scope>
```

```
  </audit>
  <work-managers>
    <loc:work-manager>
      <loc:name>WM</loc:name>
      <loc:description>WM</loc:description>
      <loc:max-threads-constraint>64</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
    <loc:work-manager>
      <loc:name>ResourceBroker-WM</loc:name>
      <loc:description>ResourceBroker-WM</loc:description>
      <loc:max-threads-constraint>15</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
    <loc:work-manager>
      <loc:name>Action-Purge-WM</loc:name>
      <loc:description>Action-Purge-WM</loc:description>
      <loc:max-threads-constraint>15</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
    <loc:work-manager>
      <loc:name>ExecuteEngine-WM</loc:name>
      <loc:description>ExecuteEngine-WM</loc:description>
      <loc:max-threads-constraint>15</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
    <loc:work-manager>
      <loc:name>ProcessRuntime-WM</loc:name>
      <loc:description>ProcessRuntime-WM</loc:description>
      <loc:max-threads-constraint>15</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
    <loc:work-manager>
      <loc:name>Actions-WM</loc:name>
      <loc:description>Actions-WM</loc:description>
      <loc:max-threads-constraint>15</loc:max-threads-constraint>
      <loc:min-threads-constraint>3</loc:min-threads-constraint>
    </loc:work-manager>
  </work-managers>
  <heartbeat-interval>20</heartbeat-interval>
  <reconnect-attempts>3</reconnect-attempts>
  <agents>
    <agent>
      <name>PlainAgent</name>
      <host>localhost</host>
      <port>8001</port>
      <secure-port>8002</secure-port>
      <state>Enabled</state>
      <password>{Salted-3DES}P6sAuDdtOnRp7Q/eDWnhKg==</password>
```

```
    </agent>
  </agents>
  <lvm-ssh-config>
    <public-key-file/>
  </lvm-ssh-config>
  <notification>
    <smtp>
      <name>LOC EMail Notification Service</name>
      <description>LOC EMail Notification Service</description>
      <to-address>bogus</to-address>
      <from-address>bogus</from-address>
      <smtp-server>bogus</smtp-server>
      <enabled>false</enabled>
    </smtp>
    <jms>
      <name>LOC JMS Notification Service</name>
      <description>LOC JMS Notification Service</description>

<destination-jndi-name>com.bea.adaptive.loc.notification.JMSNotifier</destinat
ion-jndi-name>

<connection-factory-jndi-name>QueueConnectionFactory</connection-factory-jndi-
name>
      <jndi-properties>
        <initial-factory>org.mom4j.jndi.InitialCtxFactory</initial-factory>
        <provider-url>bogus</provider-url>
        <security-principal>bogus</security-principal>
        <password>{Salted-3DES}ZlcPX6S/hpM=</password>
      </jndi-properties>
      <enabled>false</enabled>
    </jms>
    <jmx>
      <name>JMX Notification Service</name>
      <description>JMX Notification Service</description>
      <enabled>false</enabled>
    </jmx>
    <snmp>
      <name>LOC SNMP Notification Service</name>
      <description>LOC SNMP Notification Service</description>
      <agent>
        <name>MySNMPAgent</name>
        <description>MySNMPAgent</description>
        <host>bogus</host>
        <port>5555</port>
        <trap-version>SNMPv2</trap-version>
        <enable-inform>false</enable-inform>
      </agent>
      <trap-destinations>
        <destination>
```

```
          <name>testTrapDest</name>
          <description>testTrapDest</description>
          <host>bogus</host>
          <port>5555</port>
          <community>public</community>
          <security-level>noAuthNoPriv</security-level>
        </destination>
      </trap-destinations>
      <enabled>false</enabled>
    </snmp>
  </notification>
</loc-controller>
```

For information about the elements of the `loc-controller-config.xml` Controller configuration file, see the *Controller Configuration Schema Reference*.

What's Next?

After installing and creating the Plain Agent and Controller, go to Chapter 3, "Establish the WLOC Runtime Environment," which describes how to start the Agent, the Controller, and the WLOC Administration Console.

Establish the WLOC Runtime Environment

Now that we have installed and configured the Plain Agent and the Controller to establish the resource environment for this example, we need to start each of them and the Administration Console. This will establish the runtime environment needed to define the service to be managed.

The tasks in this topic include:

- Step 1: Start the Plain Agent
- Step 2: Start the Controller
- Step 3: Start the WLOC Administration Console

Step 1: Start the Plain Agent

To start the Plain Agent:

1. Open a Command Prompt window.
2. Navigate to `\bin` of the directory in which we created the Plain Agent:

   ```
   C:\WLOC_UseCase\user_projects\PlainAgent\bin
   ```

3. Enter `startAgent` at the prompt.

As the Plain Agent starts, status messages are displayed in the Command Prompt window. After the start sequence is complete, the following information message is displayed in the window:

```
<Aug 4, 2008 3:05:27 PM> <Info> <ServiceInspector> <All internal systems are
now RUNNING.>
```

The Command Prompt window remains open while the Agent is running.

Step 2: Start the Controller

To start the Controller:

1. Open a new Command Prompt window.
2. Navigate to `\bin` of the directory in which we created the Controller:

   ```
   C:\WLOC_UseCase\user_projects\controller\bin
   ```

3. Enter `startController` at the prompt.

As the Controller starts, status messages are displayed in the Command Prompt window. After the start sequence is complete and the Controller establishes the connection with the Plain Agent, information messages similar to the following are displayed in the window:

```
<Aug 4, 2008 3:07:04 PM EDT> <Info> <ResourceBrokerCommon> <BEA-2012151>
<Established connection with WLOC Agent running at http://localhost:8001.>
<Aug 4, 2008 3:07:04 PM EDT> <Info> <ResourceBrokerPool> <BEA-2012203>
<ResourceBroker agent with id plain-resource-pool - Registered>
[AGENTS-HANDLING] : Getting the Observer service for :
http://localhost:8001/AgentLumperObserver/Default
[AGENTS-HANDLING] : ObserverService URL =
http://localhost:8001/AgentLumperObserver/Default
<Aug 4, 2008 3:07:07 PM> <Info> <ServiceInspector> <All internal systems are now
RUNNING.>
```

The Command Prompt window remains open while the Controller is running.

Step 3: Start the WLOC Administration Console

To start the WLOC Administration Console:

1. Open a Web Browser.
2. Enter the following URL:

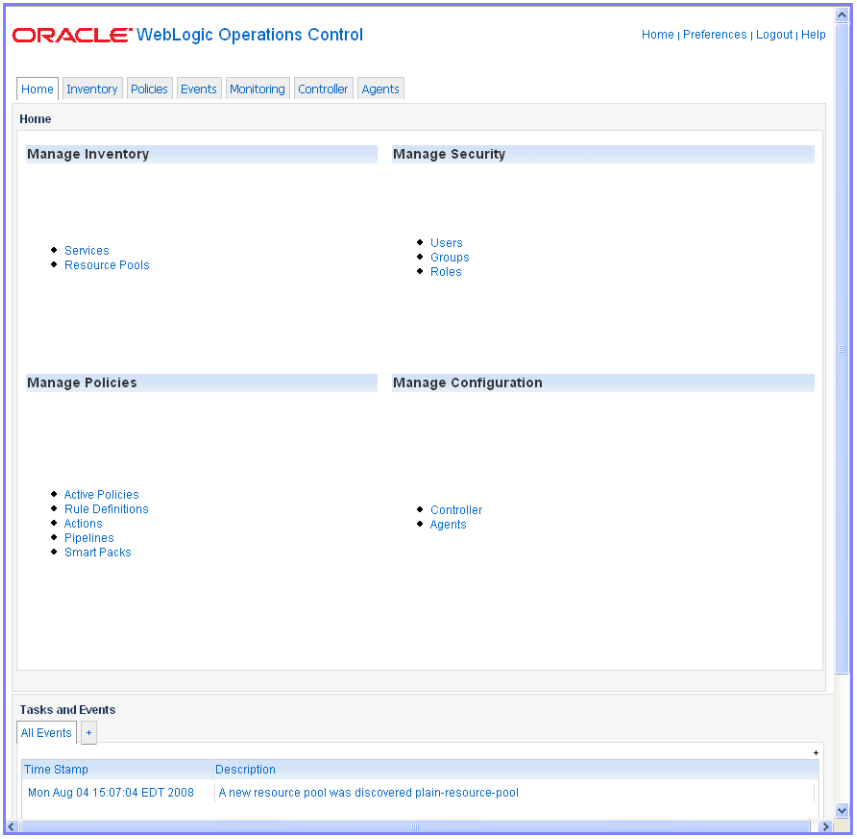
   ```
   http://localhost:9001/wloc-console
   ```

   Note: Because we are running the Controller on the local machine, we can use `localhost`. If the Controller was installed on a remote machine, you need to specify the host name for the machine hosting the Controller.

3. In the Console Welcome window, enter the user name and password required to access the Controller. Because we accepted the defaults when we created the Controller, we enter the following values in this example:

In this field	Enter the following
Username	WLOCBootUser
Password	changeit

After logging in successfully, the Home page of the WLOC Administration Console is displayed.



Note that the Task and Event Viewer at the bottom of the Console indicates that the resource pool, named plain-resource-pool, that we defined when we created the Plain Agent was discovered.

What's Next?

After establishing the runtime environment, we need to define the service to be managed. For the steps in this process, go to Chapter 4, "Define the Service Under Management."

CHAPTER 4

Define the Service Under Management

The next task in this use case example is to define the service to be managed and to create and assign deployment and runtime policies that ensure the application deploys and performs as required.

A service is a grouping of processes and a process type is a sub-group of processes within a service. (A process type is referred to as a process group in the WLOC Administration Console.) The purpose of a service is to group a collection of processes that work together. The purpose of the process group is to define instances within the service that perform the same function and can be treated as equivalent when an action is taken.

For example, you might have a two tier application with a Web app in one tier and business logic in the other. If the instances of each tier are homogeneous, then each tier could be organized as a process group and the two process groups together could comprise a service.

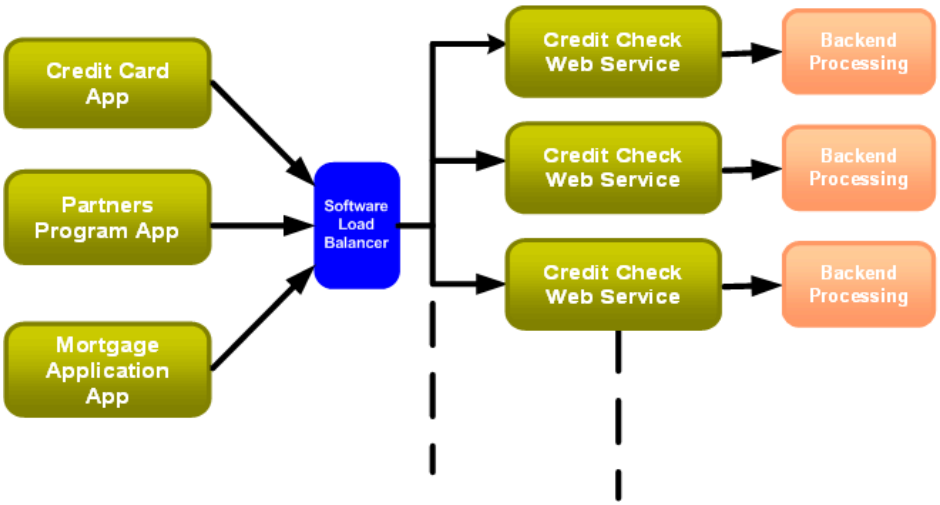
The distinction between these two groupings becomes important when defining policies. Generally speaking, policies related to deployment are applied across the whole service, whereas runtime policies are applied to a specific process group. Furthermore, process type actions, by default, will generally pick one member of the group to act on. For example, if an action is to stop an instance, the instance that gets stopped is typically not specified and the instance is chosen by the controller.

You configure services using the WLOC Administration Console, on the Inventory > Services page. A service's configuration is persisted in an XML file named `metadata-config.xml`. By default, this service metadata configuration file is located in the `BEA_HOME/user_projects/controller/config` directory. It is possible to configure a service by modifying this service metadata configuration file. However, if you configure a

service by modifying its metadata configuration file, you do not receive the benefits of validation and error checking that you get when configuring a service using the Administration Console.

Figure 4-1 illustrates an example of a SOA application that consists of multiple client applications calling into a back-end Web Service using a software load balancer. The back-end Web Service can be hosted on WLS Managed Servers.

Figure 4-1 Sample SOA Application Managed by WLOC

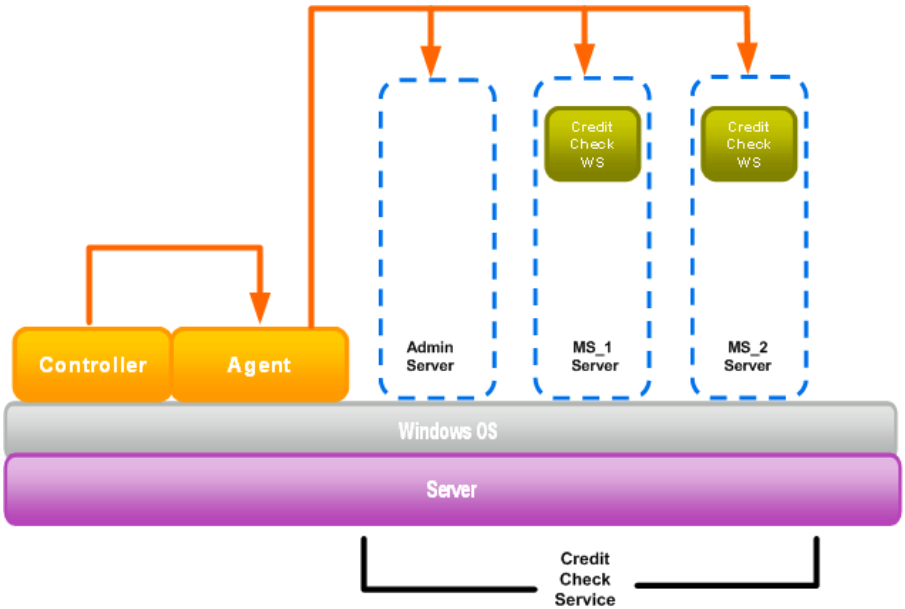


In this use case example, we will do the following:

- Use the WLOC Administration Console to create a service named CreditCheckService that will manage the backend Managed Servers and an Administration Server. The Managed Servers host the Credit Check Web Services as shown in Figure 4-2.
- Create two process groups: an Administration Server process group and a Managed Server process group.
- Specify a process requirement for each of the process groups that requires a minimum of one Admin Server and one Managed Server be started before the service is deployed.
- Define a runtime policy that executes only after the service is deployed, and will start the second Managed Server if the rule definition (constraint) is violated.

Figure 4-2 illustrates the topology in this example. Note that we are using a Windows environment in this example, but other platforms are also supported. For a list of supported platforms, see *BEA WebLogic Operations Control Supported Configurations.*

Figure 4-2 Use Case Example Topology



The tasks in this topic include:

- Step 1: Create the Service and Process Groups
- Step 2: Define the Adaptive Runtime Policies

Step 1: Create the Service and Process Groups

The first step in defining the service metadata is to define the general properties for the service, and then define the process groups that it will contain.

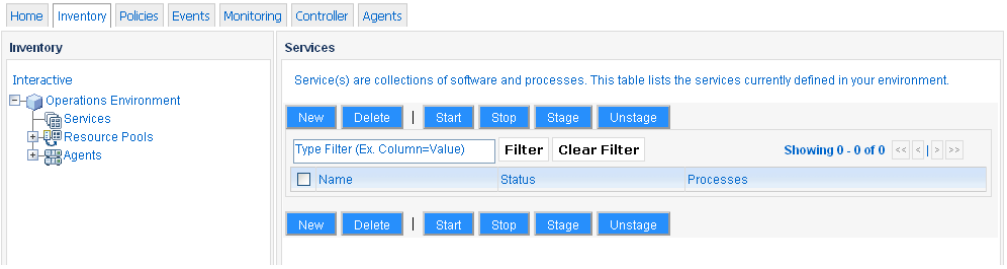
Our service will have two process groups, one that consists of a WLS Administration Server instance and one that consists of two Managed Server instances. In the WLS domain, these instances are named AdminServer, MS_1, and MS_2. You need to ensure that your WebLogic domain is configured correctly and that the servers can start successfully.

Note: Process groups are referred to as process types in the service metadata XML file.

To define the service and process groups:

1. Click the **Inventory** tab in the WLOC navigation bar and click **Services** in the tree.

Because we have not defined any services yet, there are no services listed in the table.

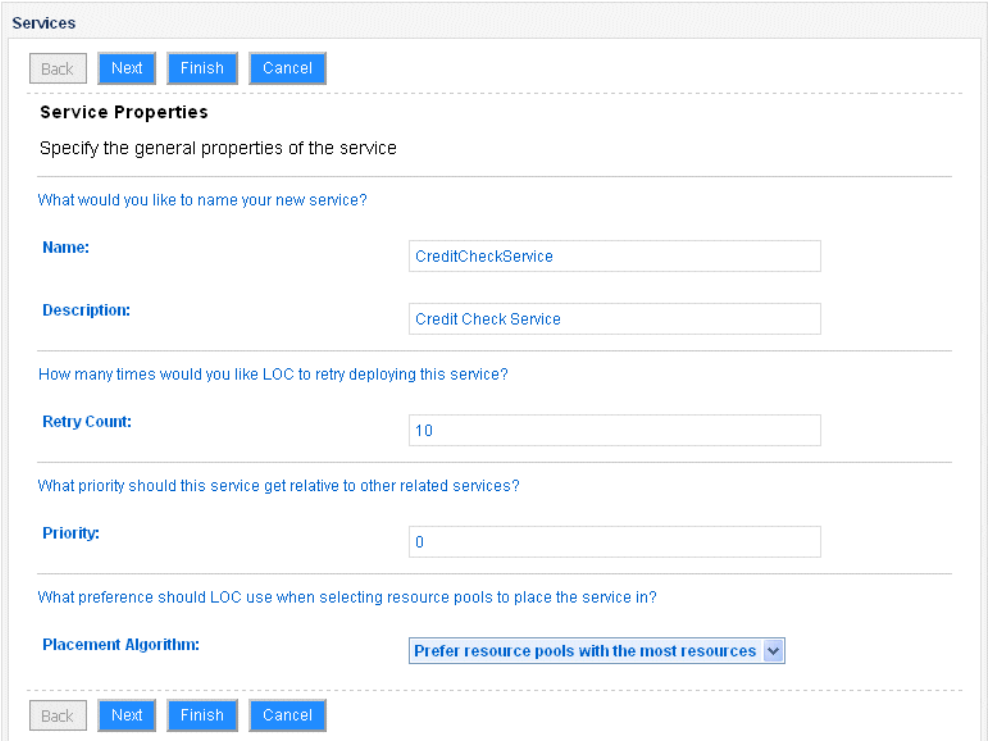


Note: You can resize the portlets by dragging the portlet resizing icon in the lower right corner of the Inventory pane.

2. Click **New** to display the **Service Properties** page.
3. Enter the values shown in the following table for the general service properties.

Table 4-1 General Service Properties

In this field . . .	Specify the following information . . .
Name	CreditCheckService
Description:	Credit Check Service
Retry Count:	10 (the default)
Priority:	0 (the default)
Placement Algorithm:	Prefer resource pools with the most resources



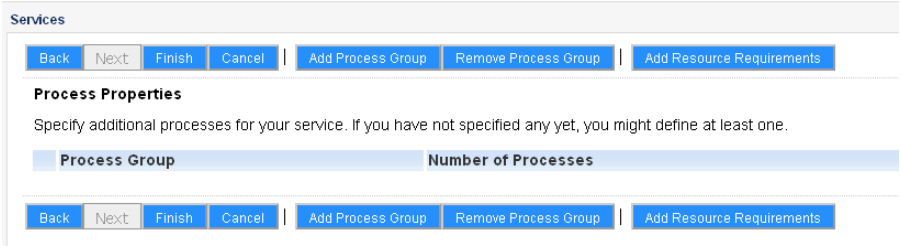
4. Click **Next**.

   The **Process Properties** page is displayed. We use this page as the starting point to add the process groups in the service.

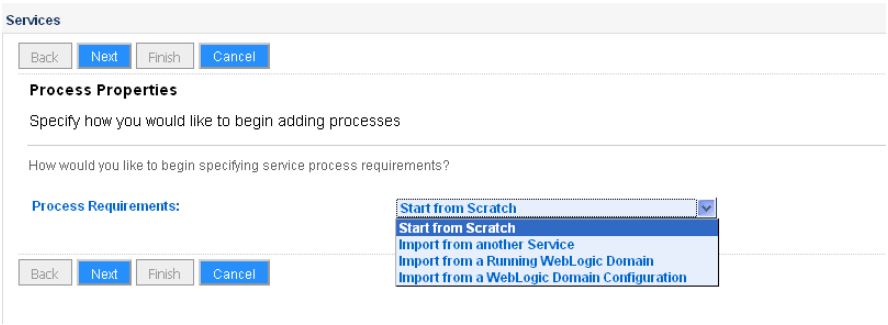
Define the Administration Server Process Group

The following steps describe how to specify the properties and define a ready metric for the Administration Server process group.

1. In the **Process Properties** page, click **Add Process Group** to add the first process group for this service.

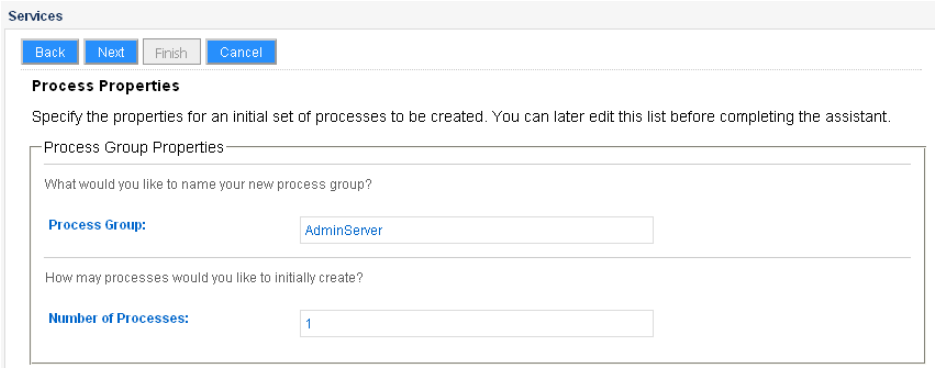


2. Select **Start from Scratch** from the **Process Requirements** drop down menu and click **Next**.



By selecting the **Start from Scratch** option, we are prompted on subsequent pages to supply properties for the process group, JVM parameters that are used when adding instances to the process group, and ready metrics that specify when an instance is available as a resource.

3. In the **Process Properties** page, we will first define the AdminServer process group. Enter the following values for the **Process Group Properties**:
   - **Process Group:** AdminServer
   - **Number of processes:** 1



4. In the **Process Group Template Properties** portion of the window, enter the values shown in Table 4-2 for the AdminServer process group instance.

Table 4-2 AdminServer Process Group Template Properties

In this field . . .	Specify the following information . . .
Name	AdminServer
Description	AdminServer JVM
Main Class	Leave this option selected. Because our service is deployed on WLS, we need to invoke the `weblogic.Server` main class.
Main JAR	Leave this option unselected.
Host:	The name of the host machine. In this example, we specify `localhost`. The host and port number are used to determine the address the Agent uses to collect JMX metric information from the endpoint.
Starting Port#	7001
JMX Service URL	In this example, we leave this field blank because we are using a WLS endpoint and we specified values in the Host and Starting Port# fields. If you are using a non-WLS endpoint, you can specify a JMX Service URL in this field as an alternative to providing values in the Host and Starting Port# fields.
Classpath	The CLASSPATH for the domain.

Table 4-2 AdminServer Process Group Template Properties (Continued)

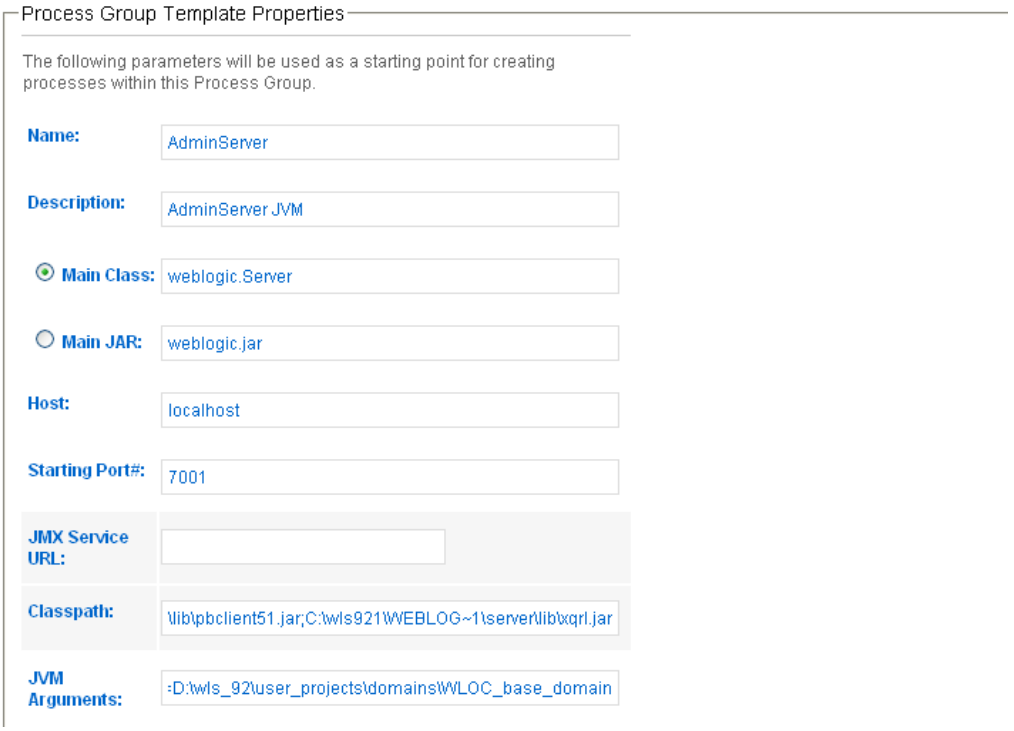
In this field . . .	Specify the following information . . .

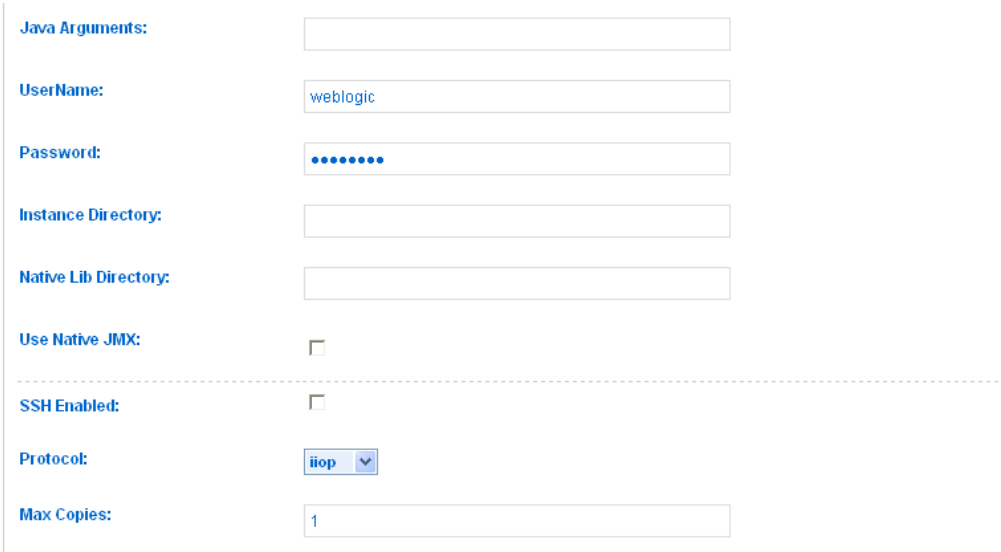
JVM Arguments	The JVM arguments to be used when the process starts. In this example, we enter the arguments that are appended to the `weblogic.Server` start command to specify minimum and maximum heap sizes, and to set the WebLogic Server instance name, security credentials, security policy, home and domain directories:
	`-Xmx128m -Xms64m -da`
	`-Dwls.home=C:\wls_92\weblogic92\server`
	`-Dweblogic.management.discover=true`
	`-Dweblogic.Name=AdminServer`
	`-Dweblogic.management.username=weblogic`
	`-Dweblogic.management.password=weblogic`
	`-Djava.security.policy=C:\wls_92\weblogic92\server\lib\weblogic.policy`
	`-Dweblogic.RootDirectory=C:\wls_92\user_projects\domains\LOC_base_domain`
	Note: All JVM arguments must be specified on one line, separated by spaces.

Java Arguments	In this example, no Java arguments are required.
UserName	The username required for authenticating JMX connections. In this example, we specify `weblogic` as both the user name and the password because those are the values required to authenticate to the Administration Server.
Password	The password required for authenticating JMX connections. Enter `weblogic`.
Instance Directory	Leave this field blank.
Native Lib Directory	Leave this field blank.
Use Native JMX	Leave this unchecked.
SSH Enabled	Leave this unchecked.

Table 4-2 AdminServer Process Group Template Properties (Continued)

In this field . . .	Specify the following information . . .
Protocol	Leave `iiop` selected.
Max Copies	Accept the default, 1. This field is used in conjunction with JVM variables to create multiple copies of a process without having to physically configure duplicates in the metadata configuration. We are not using JVM variables in this use case so we do not need to specify a different value in this field.



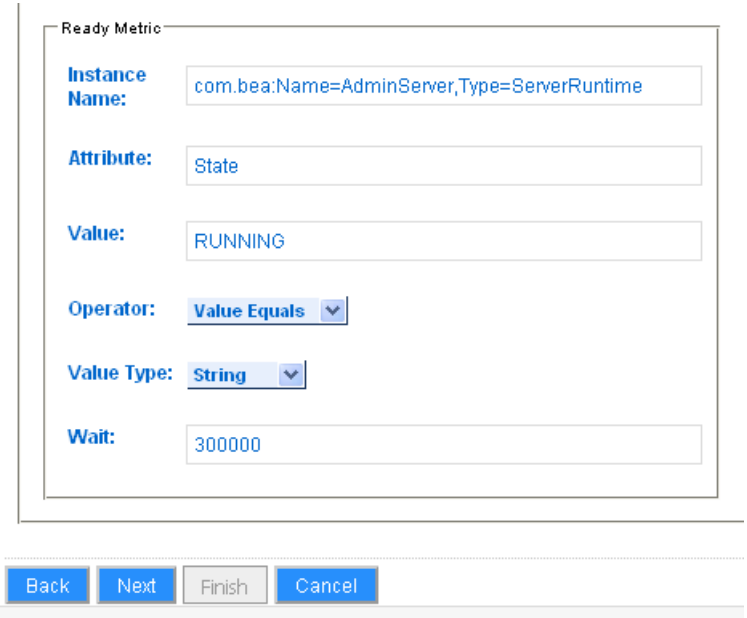


5. Specify a ready metric for the AdminServer instance by entering the values shown in Table 4-3.

   A ready metric indicates when a process has been started and is ready for work. For a WebLogic Server instance, the `ServerRuntime` MBean has a `State` attribute. When `ServerRuntimeMbean.State=RUNNING`, the WebLogic Server instance is ready.

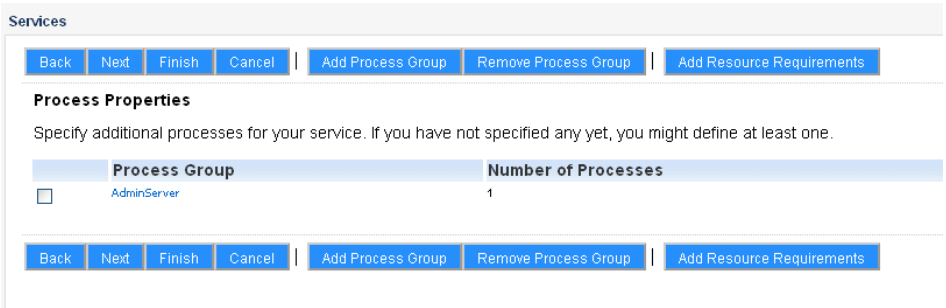
Table 4-3 AdminServer Ready Metrics

In this field . . .	Specify the following information . . .
Instance Name	com.bea:Name=AdminServer,Type=ServerRuntime
Attribute	State
Value	RUNNING
Operator	Value Equals (the default)
Value Type	String
Wait	300000 (This value ensures the WLS Admin Server instance has time to complete its startup.)



6. Click **Next**.

   Note that the AdminServer process group is listed in the **Process Group** table.



   In the next steps we will define the Managed Server process group.

Define the Managed Servers Process Group

The following steps describe how to specify the properties and ready metrics for both Managed Server instances.

1. In the **Process Properties** page, click **Add Process Group** to add the Managed Server process group.
2. Select **Start from Scratch** from the **Process Requirements** drop down menu and click **Next**.
3. In the **Start from Scratch** page, we will first define the Managed Servers process group.
   a. Name the process group **ManagedServers**
   b. Because we have two Managed Servers in this group, enter **2** in the **Number of Processes** field.
4. Define the Managed Server processes by entering the values shown in Table 4-4 for the **Process Group Template** properties.

   Note that the values that are populated in the template are obtained from the values we provided for the AdminServer process group. We modify them as appropriate for the ManagedServers group.

   The values that we specify on this page are duplicated for all the instances in the group. Therefore, in this example, both of the Managed Server instances will initially contain the same values.

Table 4-4 ManagedServers Process Group Template Properties

In this field . . .	Specify the following information . . .
Name	MS_1. WLOC automatically appends 0 to the first process instance name. For each additional process instance that is configured, a numeric suffix is added to the name starting with 1 and incrementing by 1 for each additional process instance. Because we are defining two ManagedServers, the first instance is automatically named MS_10 and the second instance is named MS_11.
Description	MS_1 JVM
Main Class	Leave this option selected. Because our service is managing WLS Admin and Managed Servers, we need to invoke the `weblogic.Server` main class.
Main JAR	Leave this option unselected.
Host:	localhost The host and port number are used to determine the address the Agent uses to collect JMX metric information from the endpoint.

Table 4-4 ManagedServers Process Group Template Properties (Continued)

In this field . . .	Specify the following information . . .
Starting Port#	7002
JMX Service URL	Leave this field blank.
Classpath	The CLASSPATH for the domain. This field is prepopulated with the CLASSPATH that we entered for the AdminServer process group.

JVM Arguments	The JVM arguments to be used when the process starts. This field is prepopulated with the arguments that we entered for the AdminServer process group. We need to modify them as follows for the MS_1 Managed Server:
	1. Add the following argument required to start Managed Servers:
	`-Dweblogic.management.server=http://localhost:7001`
	2. Change the name of the server:
	`-Dweblogic.Name=MS_1`
	The remaining values apply to both the AdminServer and the Managed Servers.
	Note: All JVM arguments must be specified on one line, separated by spaces.

Java Arguments	In this example, we do not need to specify any Java arguments.
UserName	The username required for authenticating JMX connections. We use `weblogic` in this example.
Password	The password required for authenticating JMX connections. We use `weblogic` in this example.
Instance Directory	Leave this field blank.
Native Lib Directory	Leave this field blank.
Use Native JMX	Leave this unchecked.
SSH Enabled	Leave this unchecked.
Protocol	Leave `iiop` selected.
Max Copies	Leave this value set to `1`.

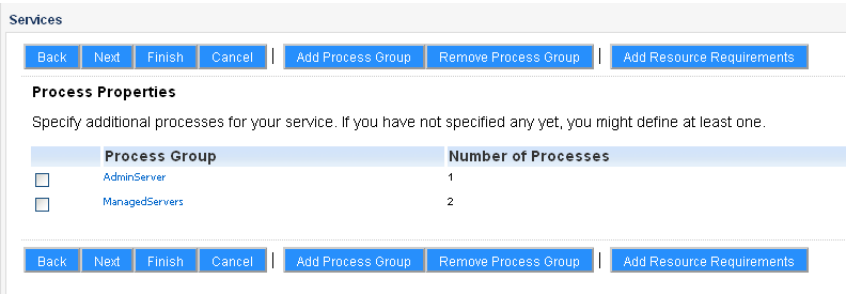
5. Define the ready metric for the MS_1 process instance as follows.

Table 4-5 ManagedServers Ready Metrics

In this field . . .	Specify the following information . . .
Instance Name	com.bea:Name=MS_1,Type=ServerRuntime
Attribute	State
Value	RUNNING
Operator	Value Equals (the default)
Value Type	String
Wait	300000

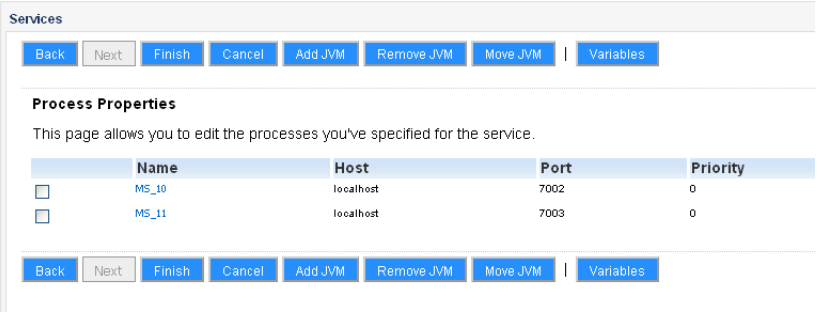
6. Click **Next**.

   The **Process Properties** page is displayed listing both the AdminServer and ManagedServers process groups in the table.

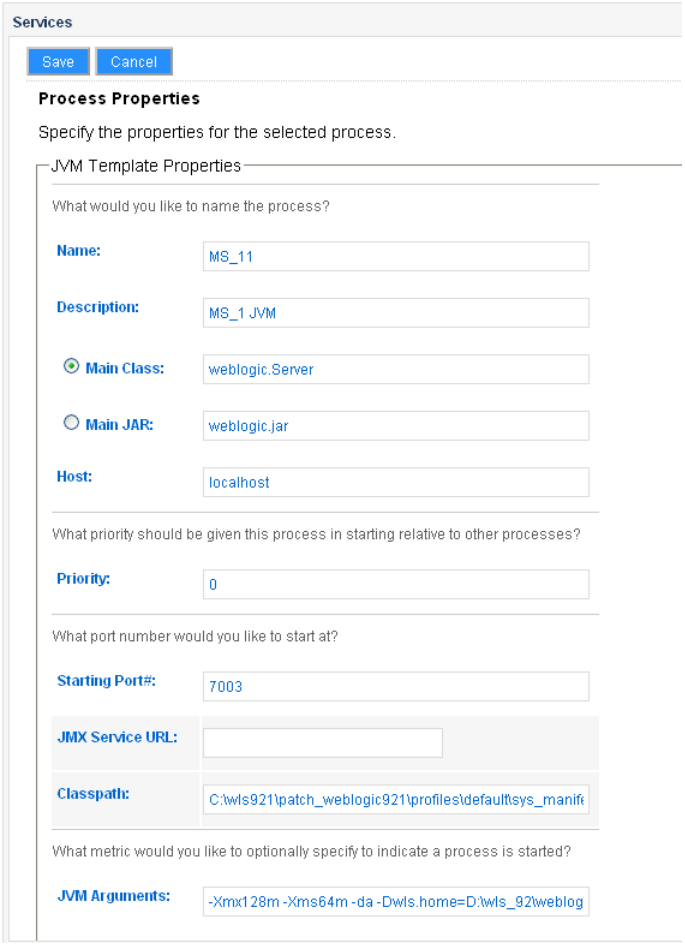


   We now need to modify the properties for the second Managed Server instance, MS_2.

7. Select the name **ManagedServers** in the **Process Group** table. (Select the name itself, not the check box.) The list of defined processes in the process group is displayed. Note that the second Managed Server instance created is named MS_11 and the port is automatically incremented to 7003.

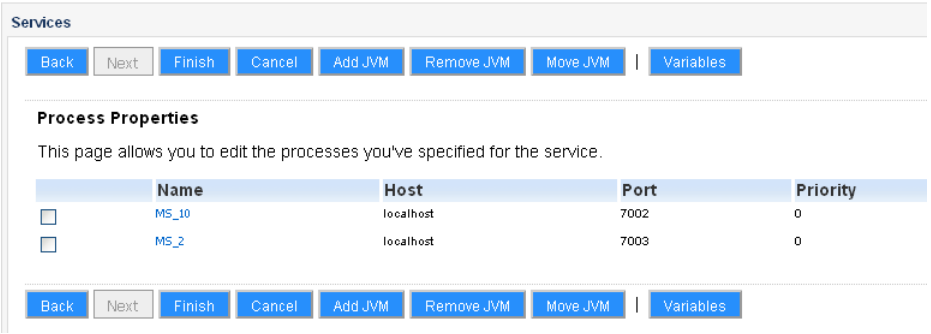


8. Select MS_11 in the **Name** column. The properties for the process are displayed.



9. In the **Name** field, change MS_11 to MS_2.
10. In the **Description** field, change MS_1 JVM to MS_2 JVM.
11. In the **JVM Args** field, change the argument `-Dweblogic.Name=MS_1` to `-Dweblogic.Name=MS_2`.
12. In the **Ready Metric** section, change the **Instance Name** field from `com.bea:Name=MS_1,Type=ServerRuntime` to `com.bea:Name=MS_2,Type=ServerRuntime`.

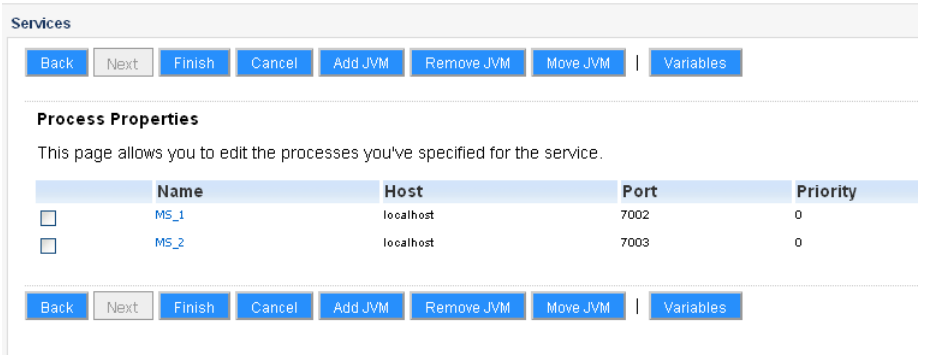
13. Click **Save**. The updated process is shown in the table.



To avoid confusion, we will also change the name of the first Managed Server from MS_10 to MS_1 to match the name of the Managed Server in the WLS domain.

14. Select MS_10 in the **Name** column. The process properties are displayed.

15. In the **Name** field, change MS_10 to MS_1 and click **Save**. We now have two processes in the ManagedServers process group named MS_1 and MS_2.



16. Click **Finish**.

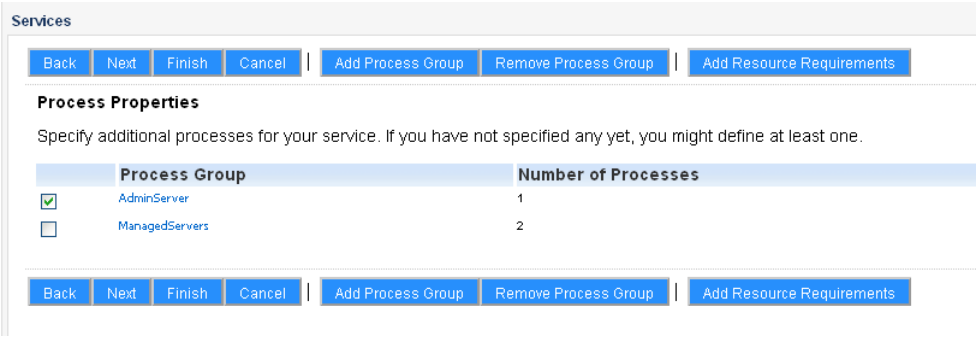
17. Define the resource requirements for the process groups as described in the following section.

Define Resource Requirements for the Service

Before we finish creating the service, we need to define the resource requirements for each process group. When you define a resource requirement, you are creating a resource agreement. *Resource agreements* define requirements that are evaluated before starting a service or instance, which can occur both at deployment and runtime.

To define the resource requirements for the process groups in the CreditCheckService, we complete the following steps.

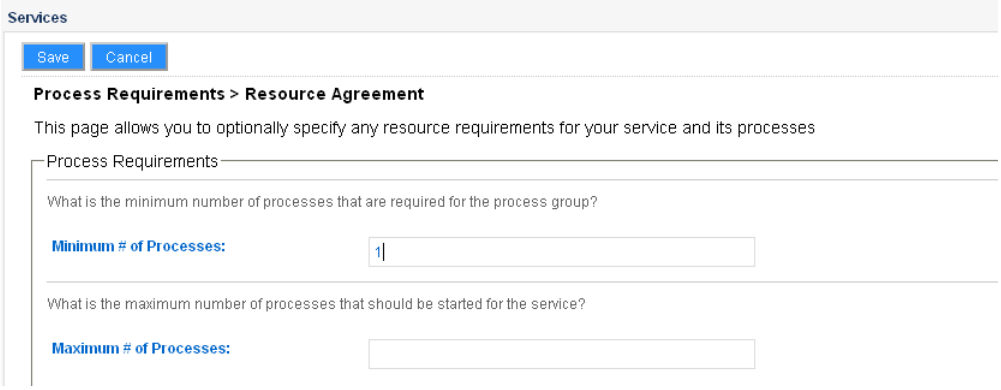
1. Select the **AdminServer** process group check box and click **Add Resource Requirements**.



2. In the **Minimum # of Processes** field, enter 1 and click **Save**.

   This value specifies the number of instances that will be started when the service is deployed. It will also generate a policy that ensures that the minimum number specified is maintained while the service is running.

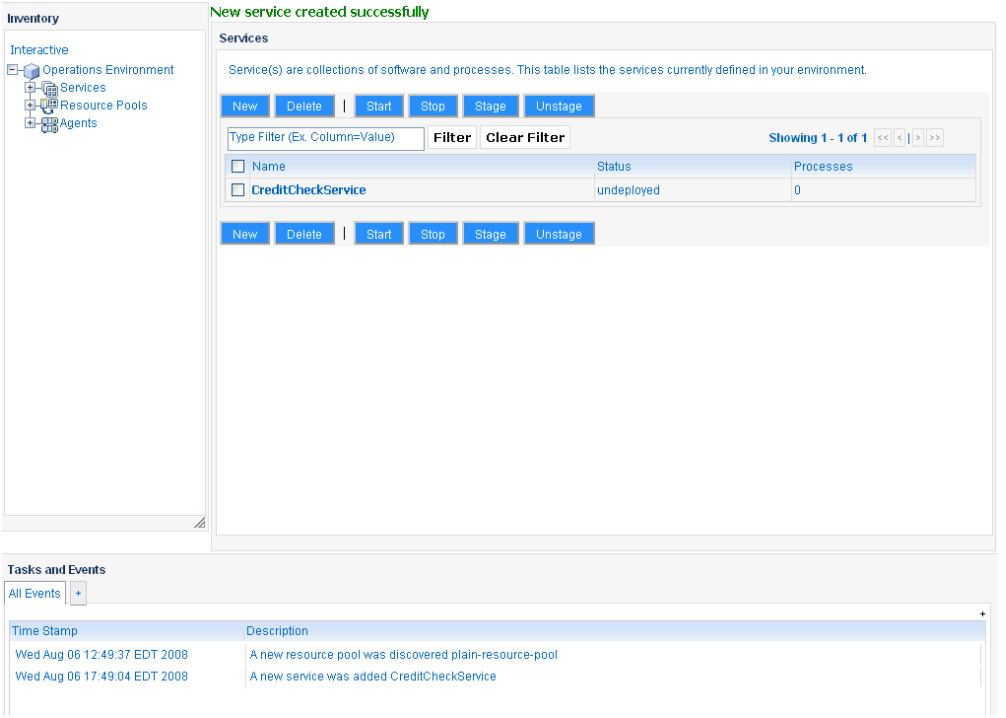
   We do not need to specify any other requirements for the AdminServer process group for this example.



3. Repeat steps 1 and 2 for the **ManagedServers** process group.

4. Click **Finish** to create the service.

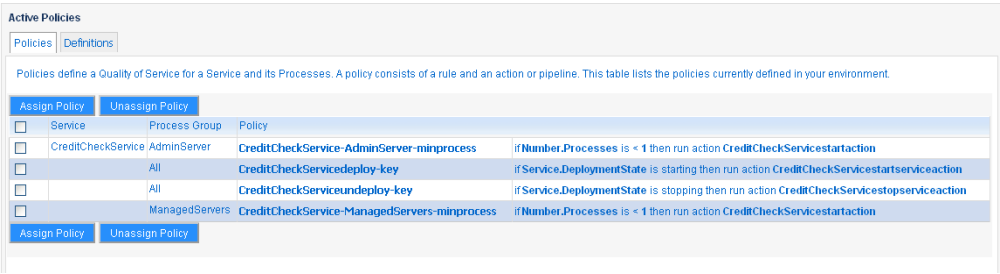
   The CreditCheckService is now shown in the **Services** table and the **Task and Events** Viewer at the bottom of the Console indicates that the CreditCheckService was added.



View Deployment Policy

When you specify the resource requirements for the service, the deployment policy for the service is created automatically. A policy consists of a constraint and an action to take when the constraint is violated. In this example, we set the minimum number of processes to 1 which indicates that there must be one running instance in both the AdminServer and ManagedServers process groups in order to deploy the service. This constraint is automatically bound to the StartJavaInstanceAction.

To view the deployment policy, select the **Policies** tab.



Create Services Using Helper Methods

In this example, we created the service using the **Start from Scratch** option to demonstrate the complete method for defining process requirements. However, the WLOC Administration Console provides efficient helper methods that simplify this process by importing the information from the WLS domain. These helper methods include:

- Import from another Service
- Import from a Running WebLogic Domain
- Import from a WebLogic Domain Configuration

If you had selected one of these options in step 2 in Define the Administration Server Process Group, much of the information that we provided manually could have been captured from the `config.xml` file for the domain or by pointing to a running Administration Server.

CreditCheckService Service Metadata Configuration

When you create the service, the associated constraints, actions, and bindings are created automatically. Listing 4-1 shows how the CreditCheckService service configuration is represented in the `metadata-config.xml` file.

Listing 4-1 CreditCheckService Metadata Configuration

```
<ns2:services>
        <ns2:service>
            <ns2:name>CreditCheckService</ns2:name>
            <ns2:description>Credit Check Service</ns2:description>
            <ns2:state>undeployed</ns2:state>
            <ns2:priority>0</ns2:priority>
            <ns2:constraint-bindings>
```

```
                <ns2:constraint-binding>

<ns2:constraint-key>CreditCheckServicedeploy-key</ns2:constraint-key>

<ns2:action-key>CreditCheckServicestartserviceaction</ns2:action-key>
                </ns2:constraint-binding>
                <ns2:constraint-binding>

<ns2:constraint-key>CreditCheckServiceundeploy-key</ns2:constraint-key>

<ns2:action-key>CreditCheckServicestopserviceaction</ns2:action-key>
                </ns2:constraint-binding>
            </ns2:constraint-bindings>
            <ns2:process-types>
                <ns2:process-type>
                    <ns2:constraint-bindings>
                        <ns2:constraint-binding>

<ns2:constraint-key>CreditCheckService-ManagedServers-minprocess</ns2:constrai
nt-key>

<ns2:action-key>CreditCheckServicestartaction</ns2:action-key>
                        </ns2:constraint-binding>
                    </ns2:constraint-bindings>
                    <ns2:name>ManagedServers</ns2:name>
                  <ns2:description>ManagedServers-description</ns2:description>

<ns2:metadata-key>CreditCheckService-ManagedServersmetakey</ns2:metadata-key>
                </ns2:process-type>
                <ns2:process-type>
                    <ns2:constraint-bindings>
                        <ns2:constraint-binding>

<ns2:constraint-key>CreditCheckService-AdminServer-minprocess</ns2:constraint-
key>

<ns2:action-key>CreditCheckServicestartaction</ns2:action-key>
                        </ns2:constraint-binding>
                    </ns2:constraint-bindings>
                    <ns2:name>AdminServer</ns2:name>
                    <ns2:description>AdminServer-description</ns2:description>

<ns2:metadata-key>CreditCheckService-AdminServermetakey</ns2:metadata-key>
                </ns2:process-type>
            </ns2:process-types>

<ns2:max-failed-event-retry-count>10</ns2:max-failed-event-retry-count>
            <ns2:placement-algorithm>PreferLarger</ns2:placement-algorithm>
        </ns2:service>
```

```
    </ns2:services>
    <ns2:connection-factories/>
    <ns2:connection-infos/>
    <ns2:constraints>
        <ns2:deployment-state-constraint>
            <ns2:name>CreditCheckService-service-deploy</ns2:name>
            <ns2:key>CreditCheckServicedeploy-key</ns2:key>
            <ns2:priority>0</ns2:priority>
            <ns2:state>starting</ns2:state>
            <ns2:evaluation-period>0</ns2:evaluation-period>
        </ns2:deployment-state-constraint>
        <ns2:deployment-state-constraint>
            <ns2:name>CreditCheckService-service-undeploy</ns2:name>
            <ns2:key>CreditCheckServiceundeploy-key</ns2:key>
            <ns2:priority>0</ns2:priority>
            <ns2:state>stopping</ns2:state>
            <ns2:evaluation-period>0</ns2:evaluation-period>
        </ns2:deployment-state-constraint>
        <ns2:min-process-constraint>
            <ns2:name>CreditCheckService-AdminServer-minprocess</ns2:name>
            <ns2:key>CreditCheckService-AdminServer-minprocess</ns2:key>
            <ns2:priority>0</ns2:priority>
            <ns2:state>deployed</ns2:state>
            <ns2:evaluation-period>0</ns2:evaluation-period>
            <ns2:value>1</ns2:value>
        </ns2:min-process-constraint>
        <ns2:min-process-constraint>
            <ns2:name>CreditCheckService-ManagedServers-minprocess</ns2:name>
            <ns2:key>CreditCheckService-ManagedServers-minprocess</ns2:key>
            <ns2:priority>0</ns2:priority>
            <ns2:state>deployed</ns2:state>
            <ns2:evaluation-period>0</ns2:evaluation-period>
            <ns2:value>1</ns2:value>
        </ns2:min-process-constraint>
    </ns2:constraints>
    <ns2:notifications/>
    <ns2:pipelines/>
    <ns2:actions>
        <ns2:action>
            <ns2:name>CreditCheckServicestartserviceaction</ns2:name>
            <ns2:key>CreditCheckServicestartserviceaction</ns2:key>

<ns2:impl-class>com.bea.adaptive.actions.internal.StartServiceAction</ns2:impl
-class>
            <ns2:adjudicate>false</ns2:adjudicate>
            <ns2:properties/>
        </ns2:action>
        <ns2:action>
            <ns2:name>CreditCheckServicestopserviceaction</ns2:name>
```

```
            <ns2:key>CreditCheckServicestopserviceaction</ns2:key>

<ns2:impl-class>com.bea.adaptive.actions.internal.StopServiceAction</ns2:impl-
class>
            <ns2:adjudicate>false</ns2:adjudicate>
            <ns2:properties/>
        </ns2:action>
        <ns2:action>
            <ns2:name>CreditCheckServicestartaction</ns2:name>
            <ns2:key>CreditCheckServicestartaction</ns2:key>

<ns2:impl-class>com.bea.adaptive.actions.internal.StartJavaInstanceAction</ns2
:impl-class>
            <ns2:adjudicate>false</ns2:adjudicate>
            <ns2:properties/>
        </ns2:action>
        <ns2:action>
           <ns2:name>CreditCheckService-ManagedServers-defaultaction</ns2:name>
            <ns2:key>CreditCheckService-ManagedServers-defaultaction</ns2:key>

<ns2:impl-class>com.bea.arc.ui.actions.ConsoleNotificationAction</ns2:impl-cla
ss>
            <ns2:adjudicate>false</ns2:adjudicate>
            <ns2:properties/>
        </ns2:action>
        <ns2:action>
            <ns2:name>CreditCheckService-AdminServer-defaultaction</ns2:name>
            <ns2:key>CreditCheckService-AdminServer-defaultaction</ns2:key>

<ns2:impl-class>com.bea.arc.ui.actions.ConsoleNotificationAction</ns2:impl-cla
ss>
            <ns2:adjudicate>false</ns2:adjudicate>
            <ns2:properties/>
        </ns2:action>
    </ns2:actions>
```

AdminServer Process Group Metadata Configuration

Listing 4-2 shows how the AdminServer process group configuration is represented in the `metadata-config.xml` file. The ready metric configuration is shown in bold type.

Listing 4-2 AdminServer Process Group Metadata Configuration

```
<ns2:metadata-group>
            <ns2:name>CreditCheckService-AdminServermeta1</ns2:name>
            <ns2:key>CreditCheckService-AdminServermetakey</ns2:key>
            <ns2:instances>
                <ns2:jvm-instance>
                    <ns2:name>AdminServer</ns2:name>
                    <ns2:description>AdminServer JVM</ns2:description>
                    <ns2:max-copies>1</ns2:max-copies>
                    <ns2:variables/>
                    <ns2:main-class>weblogic.Server</ns2:main-class>
                    <ns2:ready-information>
                        <ns2:check-type>ValueEquals</ns2:check-type>
                        <ns2:max-wait-period>300000</ns2:max-wait-period>

<ns2:instance>com.bea:Name=AdminServer,Type=ServerRuntime</ns2:instance>
                        <ns2:attribute>State</ns2:attribute>
                        <ns2:value>RUNNING</ns2:value>
                        <ns2:value-type>java.lang.String</ns2:value-type>
                    </ns2:ready-information>
                    <ns2:jvm-args>
                        <ns2:arg>-Xmx128m</ns2:arg>
                        <ns2:arg>-Xms64m</ns2:arg>
                        <ns2:arg>-da</ns2:arg>
                      <ns2:arg>-Dwls.home=C:\wls_92\weblogic92\server</ns2:arg>
                        <ns2:arg>-Dweblogic.management.discover=true</ns2:arg>
                        <ns2:arg>-Dweblogic.Name=AdminServer</ns2:arg>
                     <ns2:arg>-Dweblogic.management.username=weblogic</ns2:arg>
                     <ns2:arg>-Dweblogic.management.password=weblogic</ns2:arg>

<ns2:arg>-Djava.security.policy=C:\wls_92\weblogic92\server\lib\weblogic.polic
y</ns2:arg>

<ns2:arg>-Dweblogic.RootDirectory=C:\wls_92\user_projects\domains\WLOC_base_do
main</ns2:arg>
                        <ns2:arg>-cp</ns2:arg>

<ns2:arg>C:\wls921\patch_weblogic921\profiles\default\sys_manifest_classpath\w
eblogic_patch.jar;
C:\wls921\JDK150~1\lib\tools.jar;C:\wls921\WEBLOG~1\server\lib\weblogic_sp.jar
;C:\wls921\WEBLOG~1\server\lib\weblogic.jar;
C:\wls921\WEBLOG~1\server\lib\webservices.jar;C:\wls921\patch_weblogic921\prof
iles\default\sys_manifest_classpath\weblogic_patch.jar;
C:\wls921\JDK150~1\lib\tools.jar;C:\wls921\WEBLOG~1\server\lib\weblogic_sp.jar
;C:\wls921\WEBLOG~1\server\lib\weblogic.jar;C:\wls921\WEBLOG~1\server\lib\webs
ervices.jar;
;C:\wls921\patch_weblogic921\profiles\default\sys_manifest_classpath\weblogic_
```

```
patch.jar;
C:\wls921\JDK150~1\lib\tools.jar;C:\wls921\WEBLOG~1\server\lib\weblogic_sp.jar
;C:\wls921\WEBLOG~1\server\lib\weblogic.jar;C:\wls921\WEBLOG~1\server\lib\webs
ervices.jar;
;C:\wls921\WEBLOG~1\common\eval\pointbase\lib\pbclient51.jar;C:\wls921\WEBLOG~
1\server\lib\xqrl.jar;;
;C:\wls921\WEBLOG~1\common\eval\pointbase\lib\pbclient51.jar;C:\wls921\WEBLOG~
1\server\lib\xqrl.jar;
;C:\wls921\WEBLOG~1\common\eval\pointbase\lib\pbclient51.jar;C:\wls921\WEBLOG~
1\server\lib\xqrl.jar</ns2:arg>
                    </ns2:jvm-args>
                    <ns2:java-args/>
                    <ns2:native-lib-dir></ns2:native-lib-dir>
                    <ns2:instance-dir></ns2:instance-dir>
                    <ns2:native-jmx>false</ns2:native-jmx>
                    <ns2:protocol>iiop</ns2:protocol>
                    <ns2:host>localhost</ns2:host>
                    <ns2:port>7001</ns2:port>
                    <ns2:username>weblogic</ns2:username>

<ns2:password>{Salted-3DES}tkGybm3FYNQvTYnIKW3B7Q==</ns2:password>
                    <ns2:ssh-enabled>false</ns2:ssh-enabled>
                    <ns2:wait-for-ssh>false</ns2:wait-for-ssh>
                    <ns2:priority>0</ns2:priority>
                    <ns2:copies-at-create/>
                    <ns2:copies-at-shutdown/>
                </ns2:jvm-instance>
            </ns2:instances>
        </ns2:metadata-group>
    </ns2:metadata>
```

Step 2: Define the Adaptive Runtime Policies

Now that we have defined the service and the initial deployment policies, we need to define the runtime policies.

WLOC policies specify runtime requirements (constraints or rules) for a service and actions to take when the service operates outside of the constraints. These policies define service level agreements (SLAs) for your services. For example, you can define a policy that starts another JVM if the constraint is violated.

WLOC contains a set of predefined constraints, called Smart Packs, that you can use to place requirements on some common measurements of service health and performance. In this

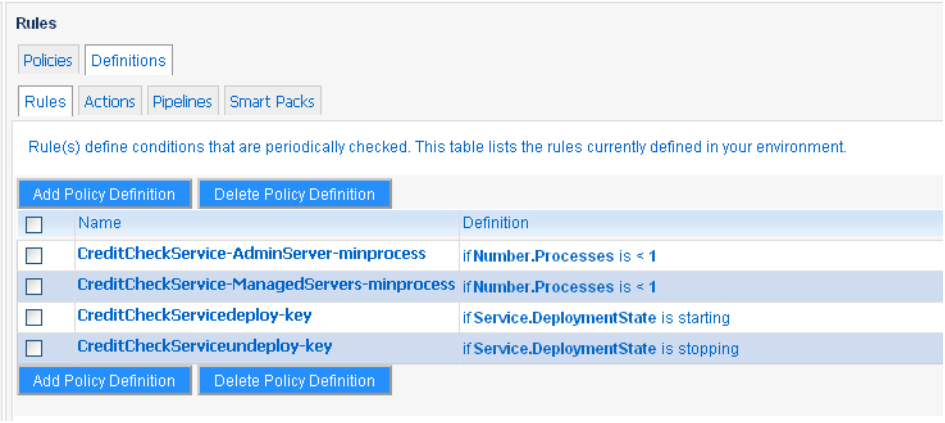
example, we will define a runtime policy for the ManagedServers process group using the MaxAverageJVMProcessorLoad constraint that is provided as part of the Smart Pack constraints.

For the runtime policy, we will define a rule (constraint) in which we specify a value of 0.8 for the MaxAverageJVMProcessorLoad constraint. This value indicates that when the average JVM processor load exceeds 80%, an action must occur. In this example, a second Managed Server instance will be started.

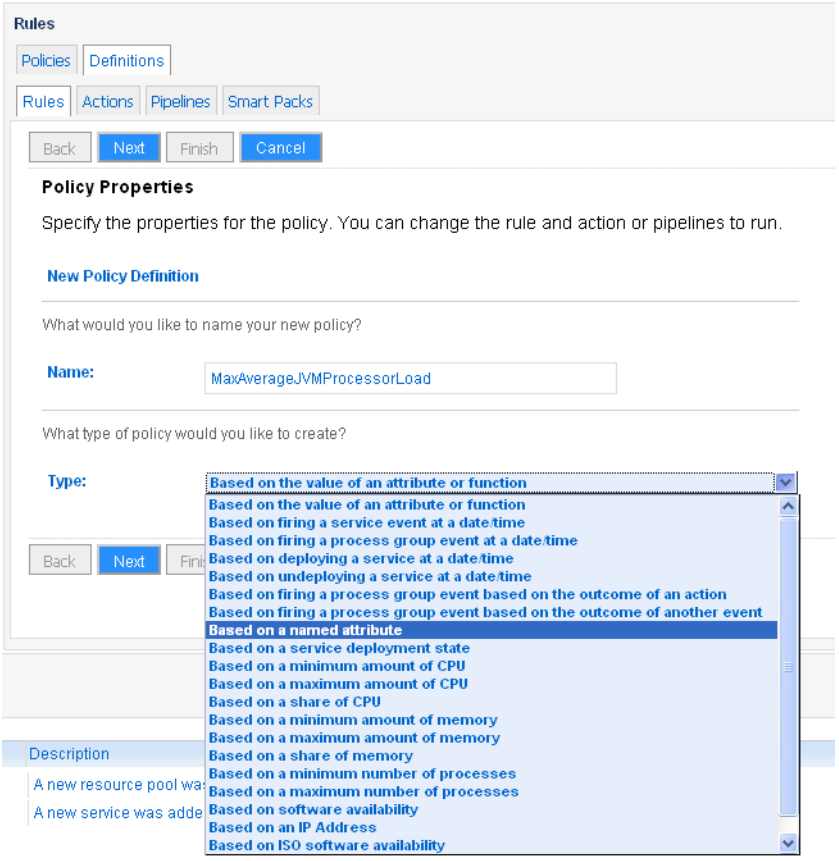
To create the runtime policy and assign it to the ManagedServers process group:

1. Select the **Policies** tab in the WLOC navigation bar, select the **Definitions** tab, and then select the **Rules** tab.

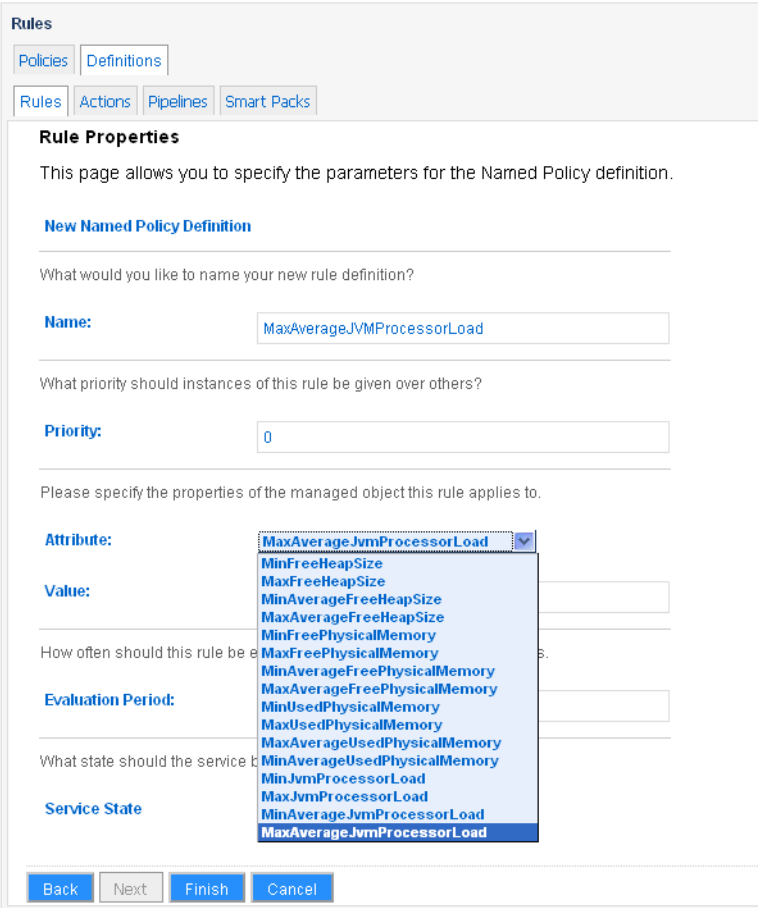
   The list of the process constraint deployment polices are shown in the table.



2. Click **Add Policy Definition**.
3. In the **Policy Properties** page, enter **MaxAverageJVMProcessorLoad** in the **Name** field.
4. Select **Based on a named attribute** from the **Type** drop-down menu and click **Next**.

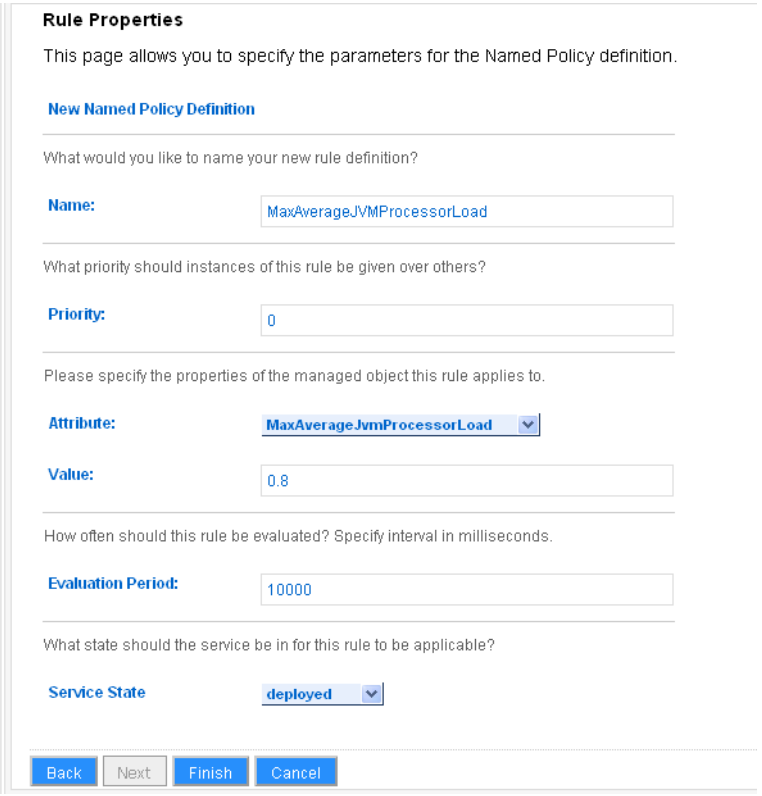


5. In the **Rule Properties** page, accept the default for the **Name** and **Priority** fields.
6. In the **Attribute** field, select **MaxAverageJVMProcessorLoad** from the drop-down menu.

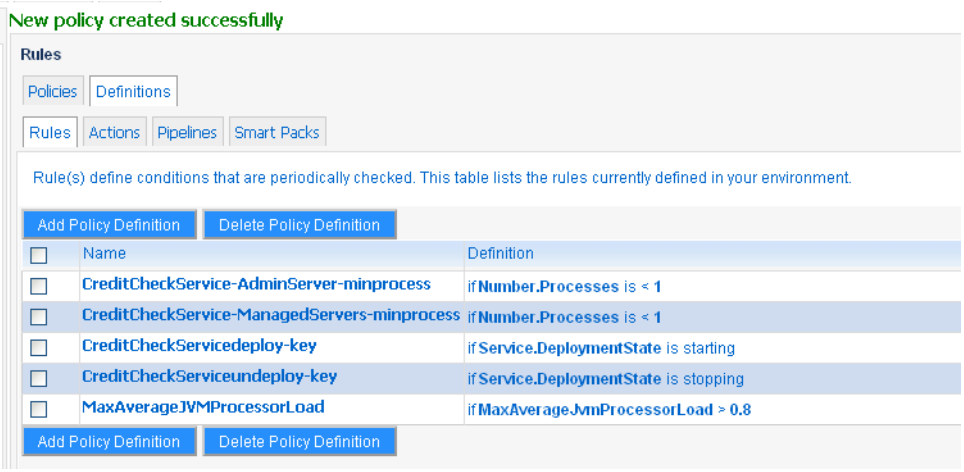


7. Specify the remaining values as shown in the following table and click **Finish**.

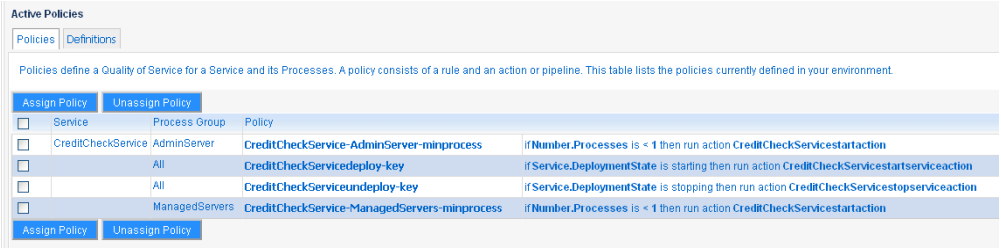
In this field . . .	Do the following . . .
Value	Enter 0.8.
Evaluation Period	Enter 10000. This is the amount of time, in milliseconds, to wait before reevaluating the constraint.
Service State	Select **deployed** from the drop-down menu. This indicates that the service must be deployed before the constraint will be evaluated.



The new policy rule is added to the **Rules** table and the confirmation message `New policy created successfully` is displayed.



8. Select the **Policies** subtab to return to the **Active Policies** page.



9. Click **Assign Policy** to assign the policy (the rule and the action to take) to the ManagedServers process group.

10. In the **Policy Properties** page, specify the properties of the policy as follows:

    a. In the **Service** drop-down menu, select **CreditCheckService** to assign the policy to the service.

    b. In the **Process** drop-down menu, select **ManagedServers** to assign the policy to the ManagedServers process group.

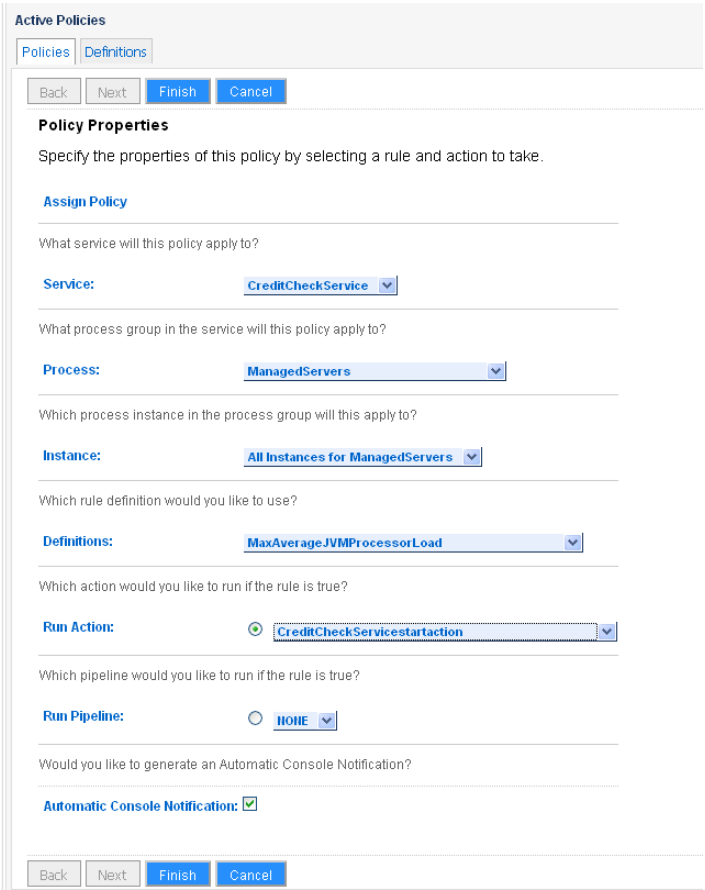
    c. In the **Instance** drop-down menu, select **All Instances for ManagedServers** to assign the policy to all the Managed Server instances in the process group.

    d. In the **Definitions** drop-down menu, select **MaxAverageJVMProcessorLoad** to assign the rule definition to the policy.

e. In the **Run Action** drop-down menu, select **CreditCheckServicestartaction** to specify the action to take when the constraint (rule) is violated.

f. Because we did not define an action pipeline in this example, leave **None** selected in the **Run Pipeline** drop-down menu.

g. Leave the **Automatic Console Notification** checkbox selected. This setting automatically generates a console notification when the specified constraint for the SLA is violated. A notification is also automatically generated when the action taken to rectify the SLA violation has completed.



11. Click **Finish** to finish assigning the policy to the process group.

The policy assignment is created and the `Binding created successfully` confirmation message is displayed.

Note that the **MaxAverageJVMProcessorLoad** policy is now assigned to the ManagedServers process group in the **Active Policies** table.

Runtime Policy Metadata Configuration

Listing 4-3 shows how the MaxAverageJVMProcessorLoad configuration for the constraint binding, custom constraint, and associated action is represented in the `metadata-config.xml` file.

Listing 4-3 Runtime Policy Metadata Configuration

```
.
.
.
<ns2:process-types>
                <ns2:process-type>
                    <ns2:constraint-bindings>
                        <ns2:constraint-binding>

<ns2:constraint-key>CreditCheckService-ManagedServers-minprocess</ns2:constrai
nt-key>

<ns2:action-key>CreditCheckServicestartaction</ns2:action-key>
                        </ns2:constraint-binding>
                        <ns2:constraint-binding>

<ns2:constraint-key>MaxAverageJVMProcessorLoad</ns2:constraint-key>

<ns2:action-key>CreditCheckServicestartaction</ns2:action-key>
                        </ns2:constraint-binding>
```

```
                    </ns2:constraint-bindings>
                    <ns2:name>ManagedServers</ns2:name>
                  <ns2:description>ManagedServers-description</ns2:description>

<ns2:metadata-key>CreditCheckService-ManagedServersmetakey</ns2:metadata-key>
                </ns2:process-type>
.
.
.
<ns2:custom-constraint>
            <ns2:name>MaxAverageJVMProcessorLoad</ns2:name>
            <ns2:key>MaxAverageJVMProcessorLoad</ns2:key>
            <ns2:priority>0</ns2:priority>
            <ns2:state>deployed</ns2:state>
            <ns2:evaluation-period>10000</ns2:evaluation-period>
            <ns2:constraint>MaxAverageJvmProcessorLoad</ns2:constraint>
            <ns2:value>0.8</ns2:value>
        </ns2:custom-constraint>
.
.
.
ns2:action>
            <ns2:name>CreditCheckServicestartaction</ns2:name>
            <ns2:key>CreditCheckServicestartaction</ns2:key>

<ns2:impl-class>com.bea.adaptive.actions.internal.StartJavaInstanceAction</ns2
:impl-class>
            <ns2:adjudicate>false</ns2:adjudicate>
            <ns2:properties/>
        </ns2:action>
```

What's Next?

Now that we have defined the service to be managed and assigned deployment and runtime policies, we need to deploy the service as described in Chapter 5, "Deploy the Service Against Available Resources."

CHAPTER 5

Deploy the Service Against Available Resources

The next task in the use case example is to deploy the service to the resource pools on which it will run. To start a WLOC service, you deploy it using the WLOC Administration Console, or configure it for auto-deployment. WLOC chooses one or more resource pools for the initial deployment.

To choose resources pools for an initial deployment, WLOC follows this procedure:

1. The Controller examines the process requirements that you configured for the service.
2. The Controller examines all resource pools that are currently active—including those that are hosting other services—and uses the following process of elimination to determine which resource pools are candidates for hosting the service:
   - If the service specifies software requirements, resource pools that do not offer access to all of the required software are eliminated as candidates.
   - If the service consists of a single process, resource pools that offer fewer computing resources than the service's minimum resource requirements are eliminated.
   - If the service consists of multiple processes, WLOC may use multiple resource pools to run the service.
3. After this process of elimination, WLOC determines which resource pool or combination of resource pools can be used to host the service. Then, it uses one of the following placement algorithms that you configure when you create the service to choose a resource pool or collection of resource pools:
   - **Prefer resource pools with the most resources**: WLOC selects the resource pool combination that provides the greatest amount of computing resources.

– **Prefer resource pools with fewer resources**: WLOC selects the resource pool that most closely matches the minimum resource requirements of the service. This algorithm ensures the most efficient use of resources in your data center.

Deployment Scenario

In this example, we will deploy the CreditCheckService from the Administration Console. Note that before we start the service, the state of the service is undeployed, which indicates that there are no instances running. When we start the service, the following occurs:

1. The Controller evaluates the process requirements for each process group in the service. When we specified the resource requirements in "Define Resource Requirements for the Service" on page 4-17, we specified that there should be a minimum of 1 process for each process group.

2. The Controller compares the resource pools available for each process group in the service against the resource agreements and eliminates any resource pools that cannot host the service. Because the only requirement that we specified is minimum number of processes, it will randomly choose one of the two Managed Servers for the ManagedServers process group.

3. Use the placement algorithm that we specified when we defined the service properties, **Prefer resource pools with the most resources**, to determine the resource pool on which to place the service.

4. The Controller stages each of the instances individually.

5. The Controller starts each of the instances individually. When the minimum number of processes from each group is started, the service is deployed.

6. The Controller evaluates the runtime policy. When the average load across all the instances in the ManagedServers process group exceeds the defined value, another JVM instance is started.

   In a real world client application, a typical setting for the `MaxAverageJVMProcessorLoad` might be .8, indicating that when the average load across all the instances in the ManagedServers process group exceeds 80%, an additional JVM instance is started. For the purposes of this example, we set the value of `MaxAverageJVMProcessorLoad` to 0 to trigger the required action.

7. The runtime policy is continually evaluated. When the constraint is violated again and there are no other available processes to start, the action fails.

Note: When the service is deployed, two additional directories are created in the Agent directory: `stderr` and `stdout`. The JVM standard output stream is directed to the `stdout` directory and the standard error stream is output to the `stderr` directory. Because we accepted the defaults when we created the Plain Agent using the Configuration Wizard, these directories are as follows:

```
C:\WLOC_UseCase\user_projects\PlainAgent\stderr
```

```
C:\WLOC_UseCase\user_projects\PlainAgent\stdout
```

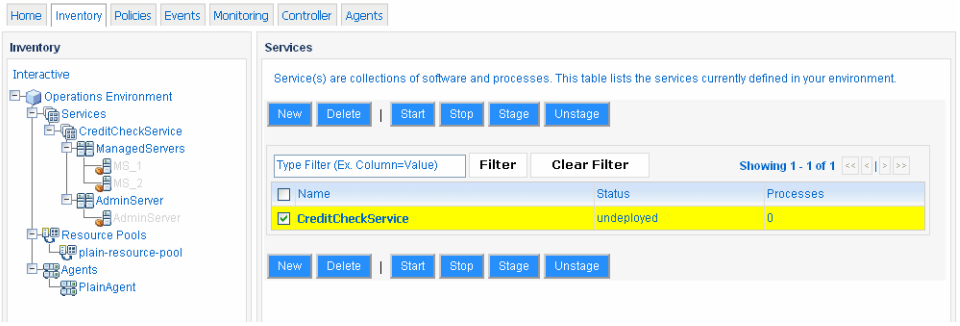
Deploy the Service

To deploy the CreditCheckService:

1. Click the **Inventory** tab in the WLOC navigation bar and click **Services**.

   The **CreditCheckService** is displayed in the **Services** table.

2. Select the **CreditCheckService** check box and click **Start**.



Note: We have expanded the hierarchy in the Inventory pane to illustrate the process groups and the configured process group instances. As the state of a process changes, the changes are reflected in the Inventory pane: when a process is not running, is unstaged, or its state cannot be determined, it appears grayed out. When a process is started, it appears in a solid, blue state. When a process is destroyed, it returns to the grayed-out state.

The message `Request to start service was successfully sent` is displayed, and the **Status** column in the table reflects the transition states of the service.

3. As the service is proceeding through the various transition states of the deployment process, view the following in the console:

– The runtime status changes in the **Status** column to reflect the current or transitional state as follows:

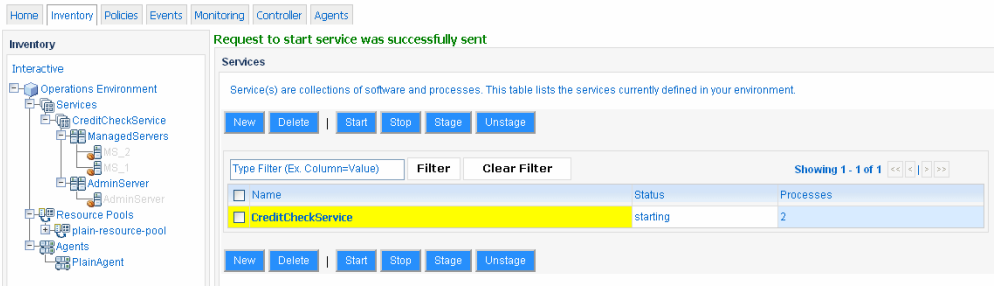
  • undeployed—No JVM instances are running.

  • staging—The service is transitioning from undeployed to staged state.

  • staged—No JVM instances are running (same as undeployed for a Plain Agent.)

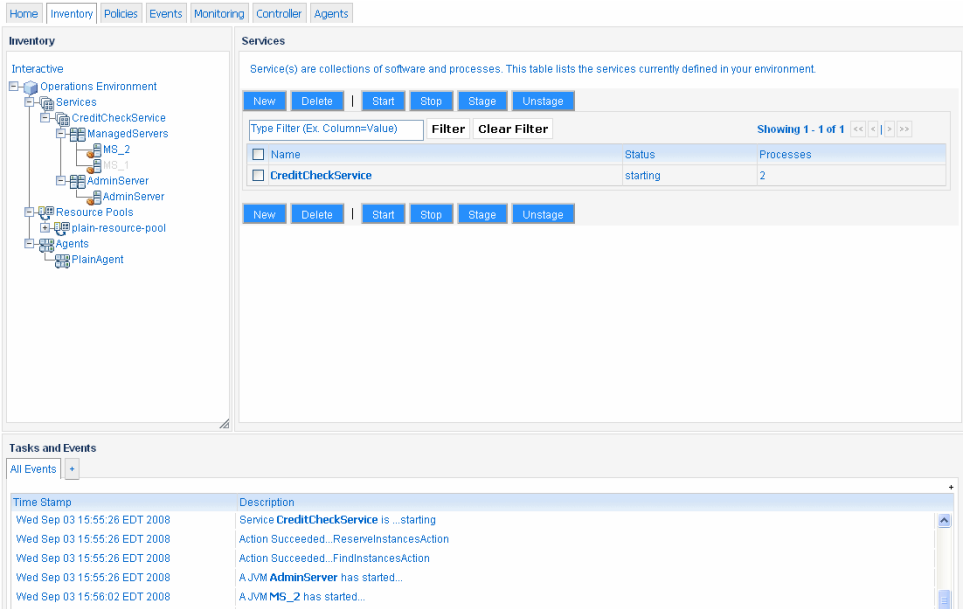
  • starting—The service is transitioning from staged to deployed state.

  • deployed—The minimum number (at least) of JVMs associated with the service are running.

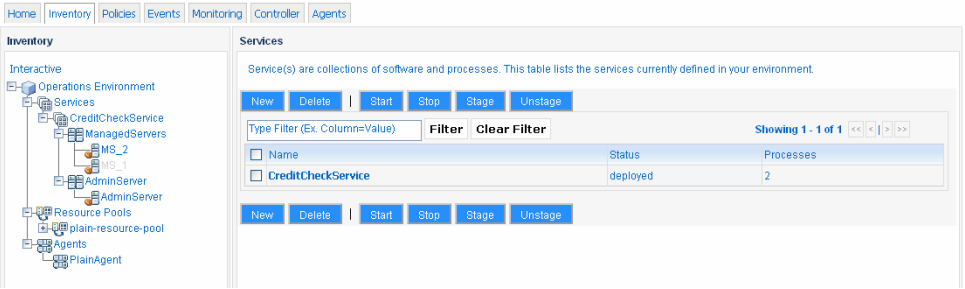


– The state of the process is shown in the Inventory pane. As each JVM process is started, it changes to dark blue in the Inventory pane.

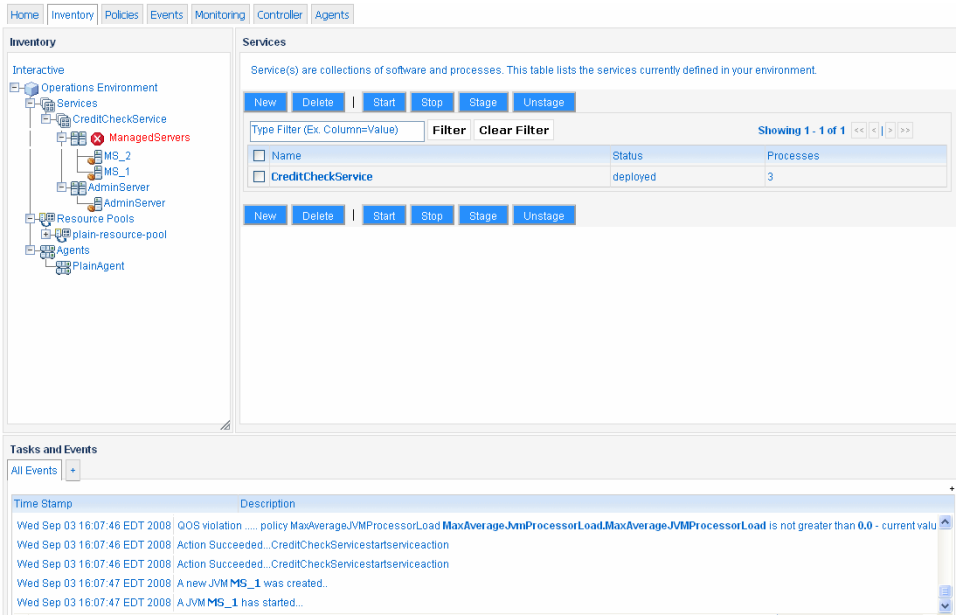
– The **Task and Event Viewer** displays each event as it occurs.



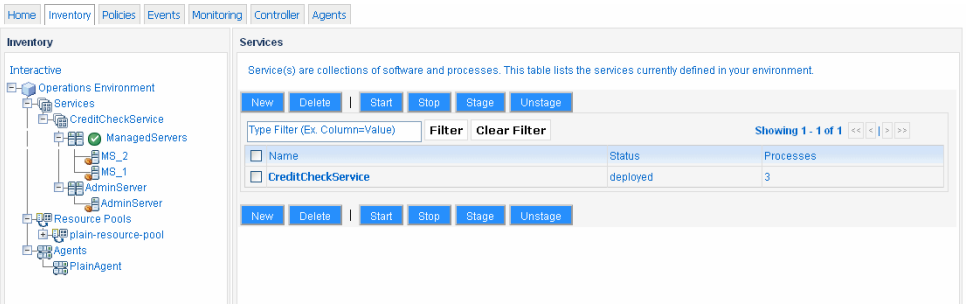
– When the minimum number of instances are started, the service state is changed to `deployed` and the runtime policy is evaluated.



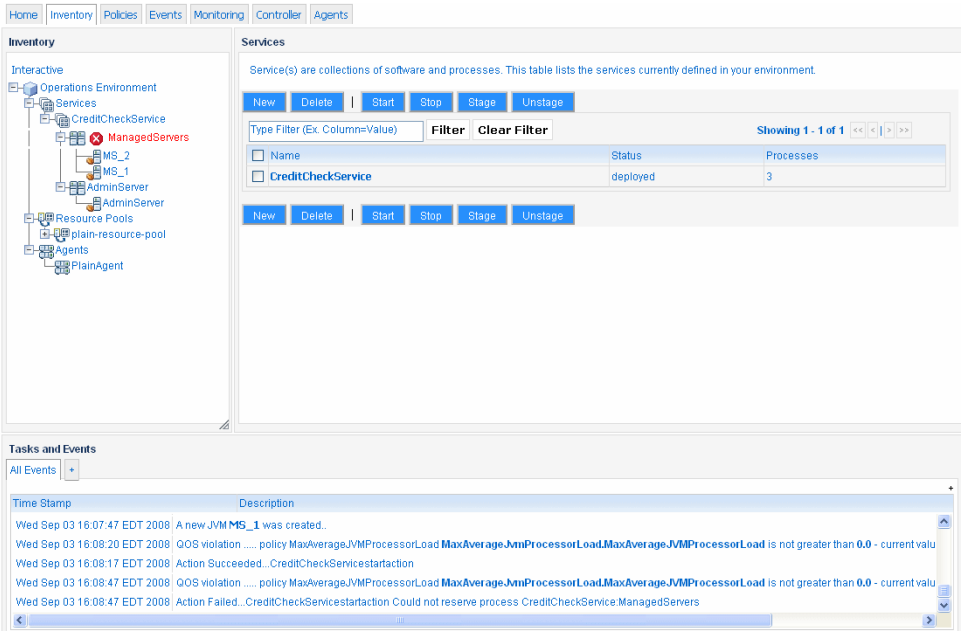
– The **Task and Event Viewer** indicates a Quality of Service (QOS) violation and the second ManagedServers JVM instance is started. Note that the Managed Server process group is red in the Inventory pane indicating that a QOS violation occurred.



– The ManagedServers process group changes to green in the Inventory pane when the second JVM instance is started, indicating that the action to rectify the violation has completed.



– The runtime policy is continually evaluated. Another QOS violation occurs. Because there are no other available processes to start, the action fails, and the ManagedServers process group turns red in the Inventory pane again.



What's Next?

After the service has deployed successfully, we can monitor the available resources as described in Chapter 6, "Monitor WLOC Services and Resources."

CHAPTER 6

Monitor WLOC Services and Resources

Now that the service is deployed, we can use the WLOC Administration Console to monitor how well the service is meeting its service level agreement, and to see which resource pools are hosting services.

To gather monitoring data, the WLOC Controller either actively polls the monitored object or passively listens for changes to a monitored object, depending on the type of data that it is gathering.

The Monitoring tab of the WLOC Administration Console contains a dashboard that enables you to construct graphs to chart metrics of services. Specifically, you can:

- Create views to organize charts according to your needs. For example, you can define one view that contains a set of charts to monitor your environment at a high-level. You can then define additional views to access more specific details, such as CPU usage on a specific set of JVMs.
- Develop custom metrics that are defined as functions of monitorable resource attributes.
- Set preferences to customize charts and graphs as desired.

The tasks described in this topic include:

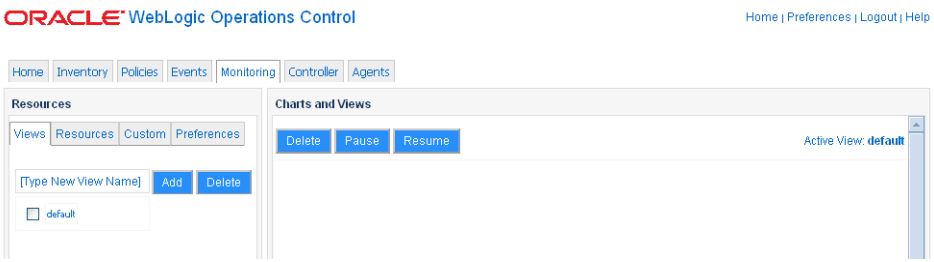
- Create a View
- Browse the Resources Pane

Create a View

To create a view:

1. Click the **Monitoring** tab in the WLOC navigation bar.

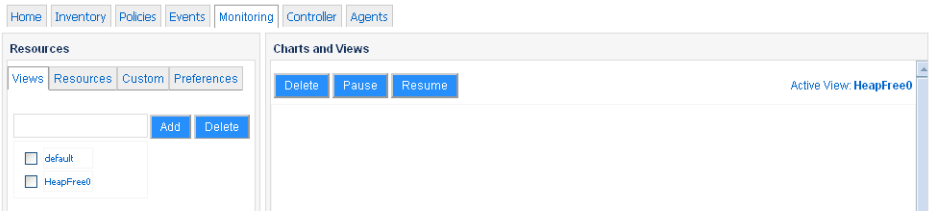
   The **Views** tab is selected by default. Because we have not created any views yet, only the Default view is listed.



2. In the text box, enter the name of the view that you want to create and click **Add**.

   In this example, we will create a view named **HeapFree** which will show the current free heap metrics for the MS_1 JVM. The view name is added to the list.

3. Click the **HeapFree** view name to make it the active view.



4. Add monitoring charts to a view by selecting resources to monitor using standard metrics available from the Resources pane and dragging it to the Charts and Views pane. For instructions on browsing the Resources pane and adding charts to a view, see "Browse the Resources Pane" on page 6-2.

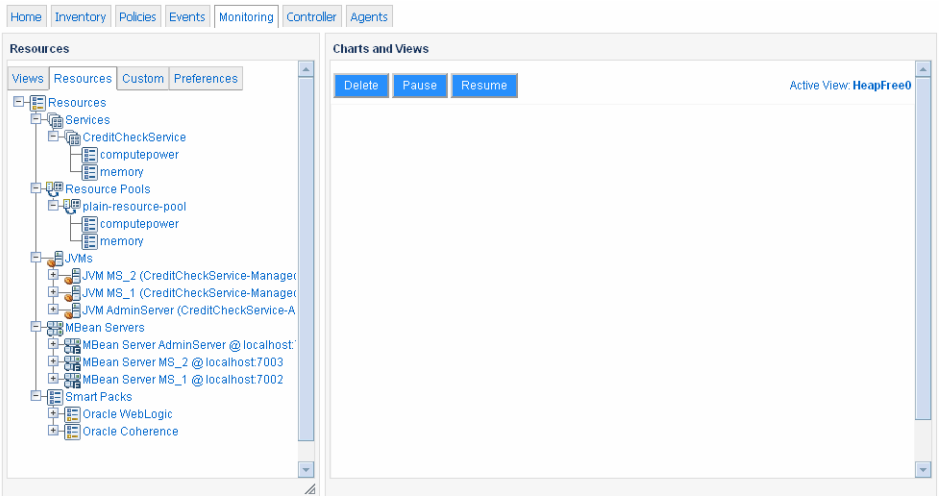
Browse the Resources Pane

When you select the **Resources** tab, it displays a list of the resources that you can monitor. Using this pane, you can monitor the CPU and memory usage for the services, resource pools, and JVMs. You can also monitor the metrics for the installed Smart Packs, as well as values of one or more MBean attributes, over time, for each of the MBean Servers.

You can expand and collapse the list of monitorable resources using the + and - next to each type in the **Resources** pane.

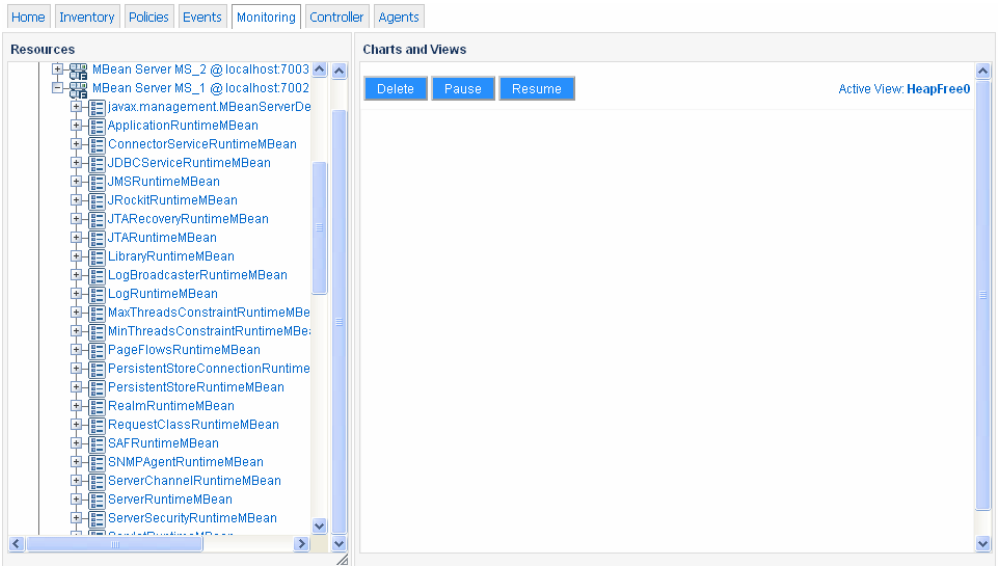
To browse the Resources pane and add a chart to the HeapFree view:

1. Select the **Resources** tab.
2. Expand the list of resources in the **Resources** pane to view the monitorable types for each resource.



3. Scroll down to view the available MBean Servers.and expand **MBean Server MS_1.**

   A list of the available MBeans types for MS_1 is displayed.



4. Expand the **JRockitRuntimeMBean** type in the list.
5. Expand the **MS_1** MBean instance.

   A list of the monitorable MBean attributes for MS_1 is displayed.
6. Scroll down the list of MBean attributes and select the **HeapFreeCurrent** attribute. Drag the attribute to the Charts and Views pane to add it to the current view.The metric chart illustrating the current free heap utilization, over time, is added to the HeapFree view.

The HeapFree view is saved automatically.