



Discover the Future of CORBA

Micro Focus[®] | CORBA[®] Add-on for Cloud, Containers & Virtual Environments 1.1.0

Installation and Configuration Guide

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK

<http://www.microfocus.com>

© Copyright 2019-2020 Micro Focus or one of its affiliates.

MICRO FOCUS, the Micro Focus logo and VisiBroker are trademarks or registered trademarks of Micro Focus or one of its affiliates.

All other marks are the property of their respective owners.

2020-05-28

Contents

Preface	1
In this Guide	1
Contacting Micro Focus	2
Further Information and Product Support	2
Information We Need	2
Contact Information	2
Introduction	5
What is the CORBA Add-on for Cloud, Containers & Virtual Environments	5
Components	5
Prerequisites	6
Uninstalling	6
CORBA in the Cloud or in Virtual Environments	7
Introduction	7
Installation in the Cloud or in Virtual Environments	8
Installation Footprints	8
Installation footprint on Linux.....	9
Installation footprint on Windows.....	9
Deployment Scenario for Cloud and Virtual Environment	10
Installation Prerequisites	11
Installation Steps	11
CORBA for Cloud and Virtual Environments: Installed Components Overview ..	12
Installing with the GUI	13
Silent installer properties.....	18
CORBA for Cloud and Virtual Environments: Upgrading existing installations ..	18
Installing with the GUI	18
Silent installer properties.....	20
CORBA in Containers	21
Introduction	21
Installation in Containers	22
Installation Footprint	22
Deployment Scenario for CORBA for Containers	23
Installation Prerequisites	24
CORBA for Containers: Installed Components Overview	25
Installation	27
Installing on the Docker Development machine	27
Installing with the GUI	27
Silent installer properties.....	35
Upgrading the client side ORB installation	35
Installing with the GUI	36
Silent installer properties.....	38
Silent Installation	39
Installing with the Silent Installer	39
Sample installer properties file	39
Performing a silent installation	41
Installing the SPS Client	43
Installing the SPS Client	43
Installation Overview	43
After Installation	44
Configuring the SPS Client	44

Installing Keys and Certificates	45
Docker Toolbox and IP Addresses	47
Introduction	47
Using the IP Address of the Windows System	47
Configure Oracle VM VirtualBox Port Forwarding	48
Common Docker Images	53
The Docker Images	53
Dockerfiles	54
The Operating System Docker Image	54
The Dockerfile for CentOS	54
Building the CentOS Operating System Docker Image	55
The Dockerfile for Ubuntu	55
Building the Ubuntu Operating System Docker Image	55
The I-DBC Docker Image	56
The Dockerfile for I-DBC	58
User ID	59
I-DBC Environment Variables.....	59
Common Entrypoint Helper Script	59
Install I-DBC	60
Building the I-DBC Docker Image.....	60
The Orbix 3 Docker Image	61
The Orbix 3 Docker Image	61
The Dockerfile for Orbix 3	61
User ID	62
Installing Orbix 3	62
Orbix 3 Entrypoint Helper Script.....	62
Proxified IOR Location	63
Building the Orbix 3 Docker Image	63
The Orbix 6 Docker Image	65
The Orbix 6 Docker Image	65
The Dockerfile for Orbix 6	65
User ID	66
Installing Orbix 6	66
Orbix 6 Entrypoint Helper Script.....	66
Proxified IOR Location	67
Orbix 6 Domain Name	67
Build Script	67
Building the Orbix 6 Docker Image	67
Creating Orbix 6 Deployment Descriptors	69
Introduction	69
The Basic Log Demo	70
Deployment inside a Docker Container	71
Creating the Deployment Descriptor	72
Modifying the Deployment Descriptor for use with Docker.....	73
Creating a Deployment Descriptor for your Orbix 6-based Application	74
The VisiBroker Docker Image	75
The VisiBroker Docker Image	75
The Dockerfile for VisiBroker	75
User ID	76
Installing VisiBroker	76

Install HotFixes	76
VisiBroker Entrypoint Helper Script	76
Proxified IOR Location	77
Building the VisiBroker Docker Image	77
The VisiBroker Smart Agent Relay	79
The Smart Agent in Containerized Environments	79
Topology of the Smart Agent Relay	80
Configuring I-DBC for Use with the Smart Agent Relay Within a Container	81
I-DBC Proxification using visiOSAgentPerPOA	82
Configuring the Smart Agent Relay	82
Command Line Switches	82
Logging	83
Ports	83
Properties	83
Initializing the SmartAgent Relay	84
Satisfying Smart Agent Requests	86
Successful Request/Response Cycle	86
Transient Error Mitigation	86
No response from the Internal Smart Agent Relay	87
No response from the Internal Smart Agent	88
Property Reference	88
Updating SPS Configuration Items	91
Introduction	91
Prerequisites	91
Build the Base OS and I-DBC Docker Images	92
Build the Base Docker Image	92
Build the I-DBC Docker Image	92
Run the I-DBC Docker Image	92
Save the Current SPS Configuration	93
Start I-DBC inside Docker	93
Change the Server SSL Version	93
Diff the config file	95
Structure of the SPS Configuration File	96
Determine the Full Name of the Server SSL Version Configuration Item	97
Using the Configuration Item Name and Value	99
Published Ports with Docker	100
Using I-DBC to Proxy Transient and Persistent IORs	100
Using CORBA in a Kubernetes Environment	103
Introduction	103
Example Configuration	103
External Load Balancer	104
Pods	104
Replicas	105
Kubernetes Cluster	105
Kubernetes Deployments	105
Kubernetes Services	106
Kubernetes Secrets	107
YAML Files	107
CORBA Application Considerations for Kubernetes	108
Client Considerations	108
Server Considerations	110
Servers and State	110
Server Proxified IOR	111
Special ORB Facilities	111

Consider a Single Replica	111
The Greeter Example	112
Introduction	112
Prerequisites	112
IDL	113
Build the Server	113
Build the base Docker images	113
Build the Docker image for the server	114
Build the Client	117
Tag and Push the Server Docker Image to a Registry	117
Create a Kubernetes Secret Object to Hold Licensing Information	118
Create the Kubernetes Service for the Greeter Example	119
YAML File Template	119
Updating the YAML File Template.....	120
Create the Service	121
Check the Service	121
Create the Kubernetes Deployment for the Greeter Example	122
YAML File Template	122
Updating the YAML File Template.....	124
Create the Deployment.....	125
Check the Deployment.....	125
Check the Replica Set.....	125
Check the Pods.....	126
Display the Pod Logs	126
Extract the IOR from a Pod	126
Run the Client Outside Kubernetes	127
Troubleshooting	128
Pods will not start	128
The Service has no External IP Address or DNS Name	129
OBJECT_NOT_EXIST Exceptions	129
I-DBC will not start in the pod	129
The CORBA Application will not start in the pod	129
The IOR does not proxify correctly in the pod	129

Index	131
--------------------	------------

Preface

This Guide describes the Micro Focus® | CORBA® Add-on for Cloud, Containers & Virtual Environments. It describes how to install and set up the product.

In this Guide

This manual contains the following chapters:

- [Introduction](#) describes some of the concepts of the CORBA Add-on for Cloud, Containers & Virtual Environments.
- [CORBA in the Cloud or in Virtual Environments](#) describes how the CORBA Add-on for Cloud, Containers & Virtual Environments operates in Cloud environments and in Virtual Environments and gives installation instructions.
- [CORBA in Containers](#) describes how the CORBA Add-on for Cloud, Containers & Virtual Environments operates in the Docker container and gives installation instructions.
- [Silent Installation](#) gives information on using the silent installer.
- [Installing the SPS Client](#) gives instructions for installing the SPS Client, which is a command line interface to the Security Policy Server (SPS) included in the I-DBC.
- [Docker Toolbox and IP Addresses](#) describes how to use the IP address of your Windows system with the Docker Toolbox.
- [Common Docker Images](#) describes the Docker images required for CORBA-based applications.
- [The Orbix 3 Docker Image](#) describes the Orbix 3 Docker image.
- [The Orbix 6 Docker Image](#) describes the Orbix 6 Docker image.
- [Creating Orbix 6 Deployment Descriptors](#) how to create deployment descriptors for Orbix 6-based applications.
- [The VisiBroker Docker Image](#) describes the VisiBroker Docker image.
- [The VisiBroker Smart Agent Relay](#) describes the Smart Agent Relay (osarelay) which enables you to use the Smart Agent (osagent) in containerized environments.
- [Updating SPS Configuration Items](#) describes how to configure the SPS to allow your applications to run correctly.
- [Using CORBA in a Kubernetes Environment](#) describes how to run your CORBA applications using the Kubernetes container orchestration system.

Contacting Micro Focus

Our Web site gives up-to-date details of contact numbers and addresses.

Further Information and Product Support

Additional technical information or advice is available from several sources.

The product support pages contain a considerable amount of additional information, such as:

- The WebSync service, where you can download fixes and documentation updates.
- The Knowledge Base, a large collection of product tips and workarounds.
- Examples and Utilities, including demos and additional product documentation.

To connect, enter <http://www.microfocus.com> in your browser to go to the Micro Focus home page.

Note:

Some information may be available only to customers who have maintenance agreements.

If you obtained this product directly from Micro Focus, contact us as described on the Micro Focus Web site, <http://www.microfocus.com>. If you obtained the product from another source, such as an authorized distributor, contact them for help first. If they are unable to help, contact us.

Information We Need

However you contact us, please try to include the information below, if you have it. The more information you can give, the better Micro Focus SupportLine can help you. But if you don't know all the answers, or you think some are irrelevant to your problem, please give whatever information you have.

- The name and version number of all products that you think might be causing a problem.
- Your computer make and model.
- Your operating system version number and details of any networking software you are using.
- The amount of memory in your computer.
- The relevant page reference or section in the documentation.
- Your serial number. To find out these numbers, look in the subject line and body of your Electronic Product Delivery Notice email that you received from Micro Focus.

Contact Information

Our Web site gives up-to-date details of contact numbers and addresses.

Additional technical information or advice is available from several sources.

The product support pages contain considerable additional information, including the WebSync service, where you can download fixes and documentation updates. To connect, enter <http://www.microfocus.com> in your browser to go to the Micro Focus home page.

If you are a Micro Focus SupportLine customer, please see your SupportLine Handbook for contact information. You can download it from our Web site or order it in printed form from your sales representative. Support from Micro Focus may be available only to customers who have maintenance agreements.

Introduction

This chapter introduces the Micro Focus® | CORBA® Add-on for Cloud, Containers & Virtual Environments (the CORBA Add-on for Cloud, Containers & Virtual Environments).

What is the CORBA Add-on for Cloud, Containers & Virtual Environments

The CORBA Add-on for Cloud, Containers & Virtual Environments enables you to extend your CORBA applications to operate with Micro Focus CORBA products (VisiBroker, Orbix 3 and Orbix 6) in the Cloud, in virtual environments, and in container-based platforms such as Docker.

It solves the main problem of network isolation and enables you to expose your enclosed and unreachable CORBA service to the internet or to other services outside of containers.

This product solves issues arising from two main deployment scenarios:

- The use of [CORBA in the Cloud or in Virtual Environments](#),
- The use of [CORBA in Containers](#).

Components

The CORBA Add-on for Cloud, Containers & Virtual Environments contains the following components:

- **Micro Focus IIOP Domain Boundary Controller (I-DBC):** Allows CORBA-based clients and servers to communicate easily across a network boundary where Network Address Translation (NAT) is occurring. A NAT layer can map private internal container addresses to public external host addresses. Connecting CORBA clients to services running either side of a NAT requires use of a proxy server, such as I-DBC, to manage the address translation within the CORBA object references. This is often challenging for CORBA-based applications. The I-DBC is described in the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Deployment Guide**.
- **Administration Console for I-DBC:** A graphical interface for administering the I-DBC. Using the Administration Console is described in the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Administrator's Guide**.
- **Support for the following CORBA ORBs:**
 - Orbix 3
 - Orbix 6
 - VisiBroker
- **CORBA product HotFixes:** To enable the CORBA products to work correctly with I-DBC, some HotFixes are required.

Prerequisites

Each component has its own requirements.

- **Installer:** The installer requires JDK 1.7 (or later) to be installed. The installer is available for both Windows and Linux operating systems.
- **I-DBC:** In order to run I-DBC, you will need a valid license file.
Note: You must have the license available when you perform the installation; you cannot specify a license code subsequently.
- **Administration Console:** The machine on which the administration console is installed must have JDK 1.8 (or more recent) installed.
- **VisiBroker 8.5.6 (or later):** During installation, you will need to provide the 64-bit VisiBroker 8.5 installer file and suitable license files. You will also need to provide the installer files for all HotFixes required.
- **Orbix 6.3.11 (or later):** During installation, you will need to provide the 64-bit Orbix 6.3 installer and a suitable license file.
- **Orbix 3.3.15 (or later):** During installation, you will need to provide the 64-bit Orbix 3.3 installer, the 64-bit Orbix 3.3 SSL installer and Java and C++ authentication codes. Orbix 3.3 can be installed without the authentication codes being specified, but in this case the codes must be added to the Orbix 3.3 installation before the ORB can be used.
- An **installation directory**. For a Cloud, Virtual Environment or Container installation, an installation directory must be specified. The installer uses the following default directories:

On Windows systems:

```
<System Program Files>\Micro Focus\corba_addon_cloud_container
```

On Linux systems:

```
/opt/microfocus/corba_addon_cloud_container
```

Uninstalling

An `uninstall` file is included in the installation for Cloud, Virtual Environment and Container installations, but not for ORB upgrades. Running this file guides you through removing the CORBA Add-on for Cloud, Containers & Virtual Environments.

Note that the `uninstall` removes all files created during the installation of CORBA Add-on for Cloud, Containers & Virtual Environments.

CORBA in the Cloud or in Virtual Environments

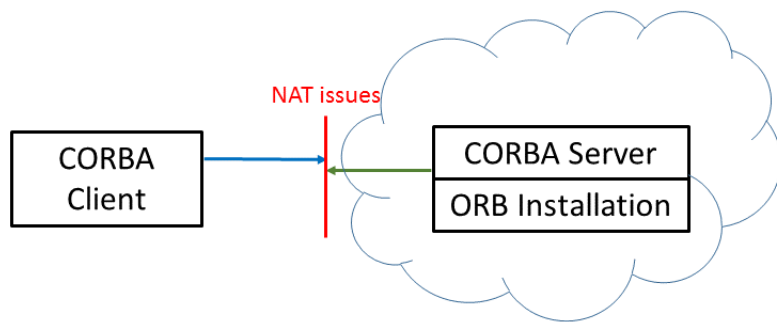
This chapter describes how the CORBA Add-on for Cloud, Containers & Virtual Environments can be used to extend CORBA functionality into the Cloud and in virtual environments.

Introduction

The CORBA Add-on for Cloud, Containers & Virtual Environments adds support for:

- The following Cloud environments:
 - Amazon AWS
 - Microsoft Azure
 - Google Cloud
- The following Virtual Environments:
 - VMWare vSphere
 - VMWare vCloud

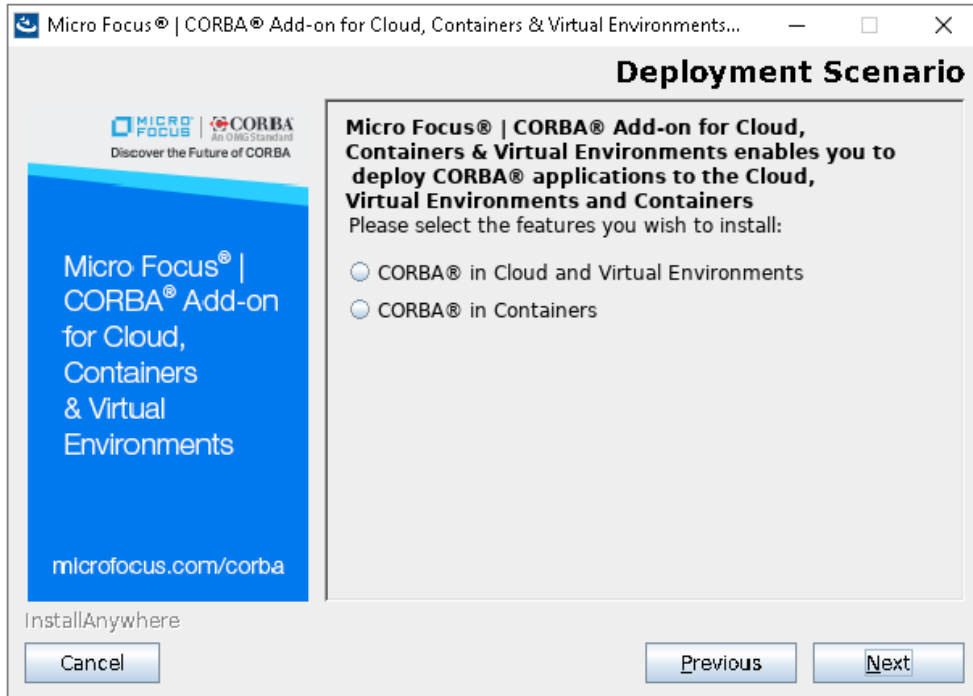
In these circumstances, the CORBA server is hidden inside an isolated network hosted on either the Virtual Environment or the Cloud provider infrastructure, and is therefore unreachable from external CORBA clients. This is shown in the following diagram.



The CORBA Add-on for Cloud, Containers & Virtual Environments product provides a solution to the issues that can arise when trying to connect to CORBA server applications in a cloud or virtual environment with Network Address Translation (NAT) or any other network bridging issues and the challenge when trying to connect to CORBA clients that are running outside of this environment.

Installation in the Cloud or in Virtual Environments

In order to deploy the CORBA Add-on for Cloud, Containers & Virtual Environments in a Cloud or Virtual Environment, run the installer and select the **CORBA in Cloud and Virtual Environments** option at the **Deployment Scenario** window.



Installation Footprints

The CORBA Add-on for Cloud, Containers & Virtual Environments product provides the following components for CORBA in Cloud and Virtual Environments deployments:

- **Micro Focus IIOP Domain Boundary Controller (I-DBC):** Allows CORBA-based clients and servers to easily communicate across a network boundary where network isolation is occurring. Requires a Linux host.
- **Administration Console for I-DBC:** A graphical interface for administering I-DBC. Using the Administration Console is described in the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Administrator's Guide**.
- The capability to upgrade an existing CORBA ORB installation with features and capabilities to communicate across a network boundary with the help of I-DBC.
- The I-DBC install packages and scripts to be used on the Linux host.
- Administration Console install package to be extracted and installed on the host.

Installation footprint on Linux

Installing the product for a Cloud environment or a virtual environment, on a Linux system, installs the following files (assuming you are installing on the Domain Boundary Controller (I-DBC) machine):

<code>idbc/</code>	Directory containing the full I-DBC installation
<code>adminconsole/</code>	Directory containing the extracted Administration Console GUI to administer an I-DBC installation
<code>doc/license_agreement.txt</code>	
<code>doc/notices.txt</code>	
<code>resources/mf_idbc_install.sh</code>	I-DBC installer script
<code>resources/mf_idbc_services.sh</code>	I-DBC service installer script to be run after I-DBC installation
<code>resources/microfocus_CLI-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources/microfocus_IDBC-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources/microfocus_SPS-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources/microfocus_AdminConsole.tar.gz</code>	AdminConsole package for manual extraction
<code>uninstall/</code>	Contains the files for uninstallation

Installation footprint on Windows

Installing the product for a Cloud environment or a virtual environment, on a Windows system, installs the following files (assuming you are installing on the Domain Boundary Controller (I-DBC) machine):

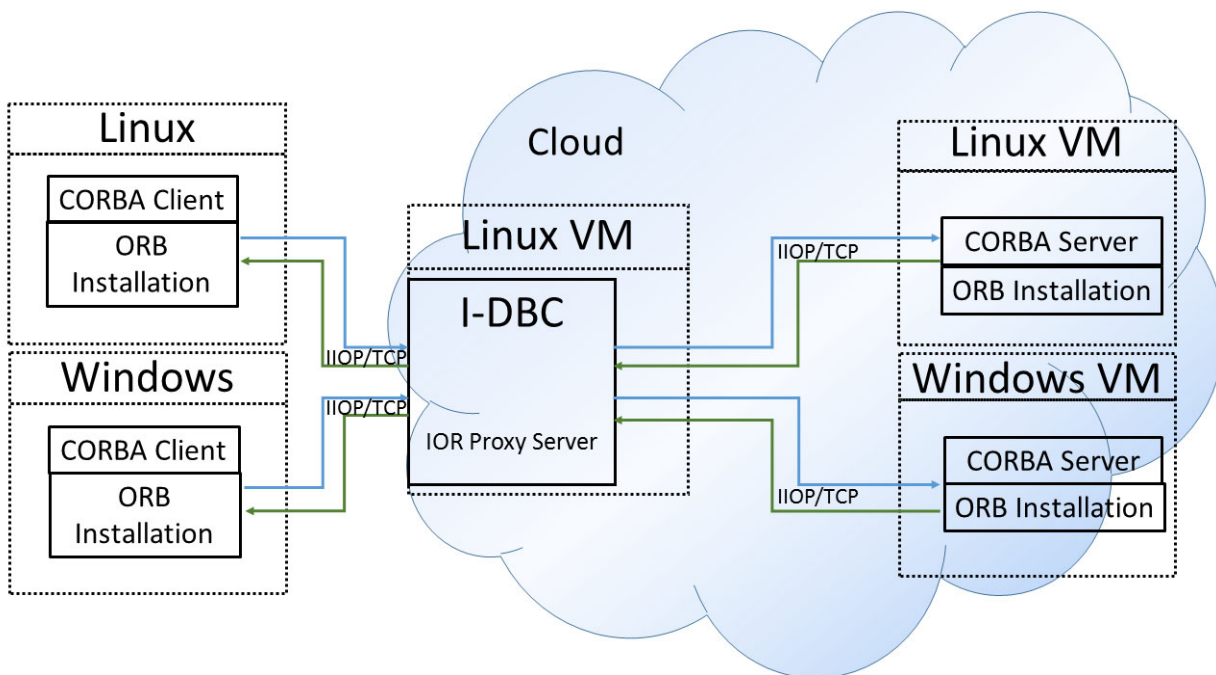
<code>adminconsole\</code>	Directory containing the extracted Administration Console GUI to administer an I-DBC installation on a Linux host
<code>doc\license_agreement.txt</code>	
<code>doc\notices.txt</code>	
<code>resources\mf_idbc_install.sh</code>	I-DBC installer script to be installed on a Linux host
<code>resources\mf_idbc_services.sh</code>	I-DBC service installer script to be run after I-DBC installation
<code>resources\microfocus_CLI-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources\microfocus_IDBC-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources\microfocus_SPS-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources\microfocus_AdminConsole.tar.gz</code>	AdminConsole package for manual extraction
<code>uninstall\</code>	Contains the files for uninstallation

Deployment Scenario for Cloud and Virtual Environment

This section describes the virtual machines that make up a typical Cloud or Virtual Environment deployment:

- **CORBA client machine:** The CORBA client application runs on this machine. This machine is usually located outside the Cloud or Virtual Environment.
- **CORBA server machine:** The CORBA server application runs on this machine. This machine is located within the Cloud or Virtual Environment and it is not directly reachable from the outside.
- **Domain Boundary Controller machine:** This machine acts as the CORBA gateway between the outer CORBA Client machines and the inner CORBA server machines.
- **Optional development machine:** This machine hosts the CORBA development environment and the CORBA Add-on for Cloud, Containers & Virtual Environments components. This can be a generic development machine or it can be dedicated to running the Administration Console component to administer the I-DBC component on the Domain Boundary Controller machine.

For a further overview and full information on the capabilities of the I-DBC and Administration Console components, see the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Deployment Guide** and the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Administrator's Guide**.



Installation Prerequisites

To deploy the CORBA Add-on for Cloud, Containers & Virtual Environments, you need the following components:

- The CORBA Add-on for Cloud, Containers & Virtual Environments Linux installer, for the installation of I-DBC.
- Optionally the CORBA Add-on for Cloud, Containers & Virtual Environments Windows installer for manual I-DBC deployment and for the Administration Console.
- A Linux VM designated as the boundary controller machine to host I-DBC.
- A license for I-DBC, if you are installing on the Domain Boundary Controller machine.
- An ORB installation. You will need to install an ORB, upgrade and existing installation to work with a Cloud deployment scenario. The current supported ORB installations are:
 - VisiBroker 8.5.6 or higher
 - Orbix 6.3.11 or higher
 - Orbix 3.3.15 or higher
- The CORBA Add-on for Cloud, Containers & Virtual Environments ORB HotFixes for your existing ORB client and server installation machines (to be downloaded from [Micro Focus Supportline](#)). The CORBA Add-on for Cloud, Containers & Virtual Environments ORB HotFixes must match the platforms (operating system, compiler version, and bitness) that your ORB installations are deployed on.

Installation Steps

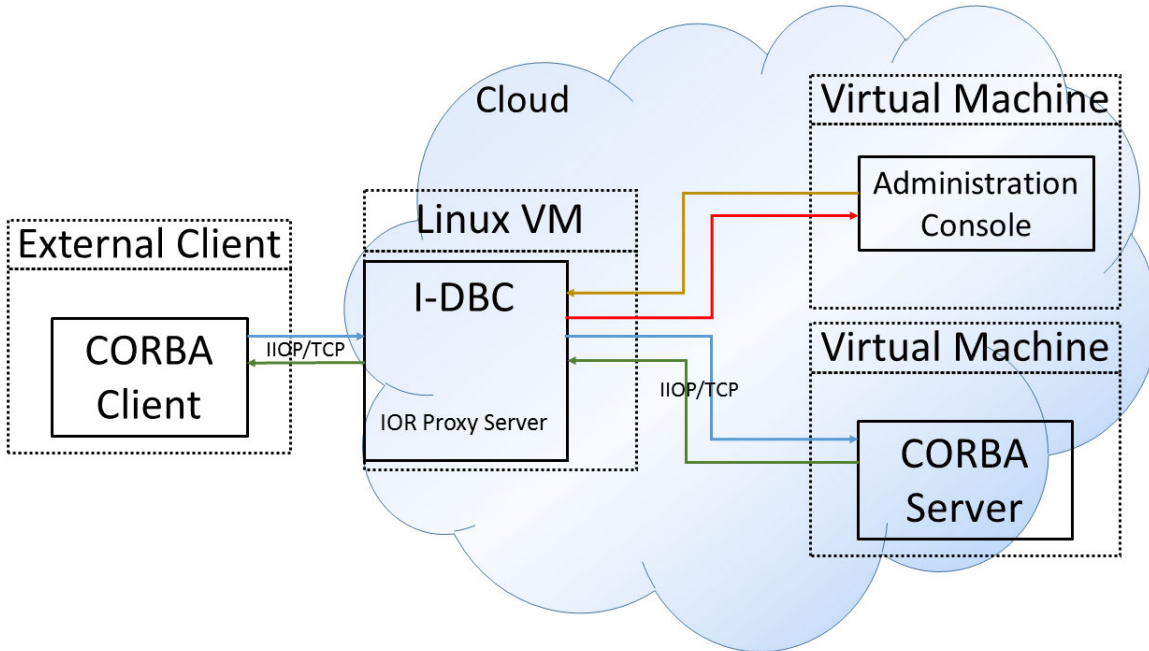
To install the CORBA Add-on for Cloud, Containers & Virtual Environments in a VCloud or virtual environment, follow these steps:

- 1 On the I-DBC Linux host machine, run the CORBA Add-on for Cloud, Containers & Virtual Environments installer selecting the **CORBA in Cloud and Virtual Environments** deployment scenario, and choosing the option to install on a **Domain Boundary Controller host**.
- 2 On the CORBA server machine, run the CORBA Add-on for Cloud, Containers & Virtual Environments installer selecting the **CORBA in Cloud and Virtual Environments** deployment scenario, and choosing the **Upgrade an existing ORB installation on this host** option.
- 3 On the CORBA client machine, run the CORBA Add-on for Cloud, Containers & Virtual Environments installer selecting the **CORBA in Cloud and Virtual Environments** deployment scenario, and choosing the **Upgrade an existing ORB installation** option.

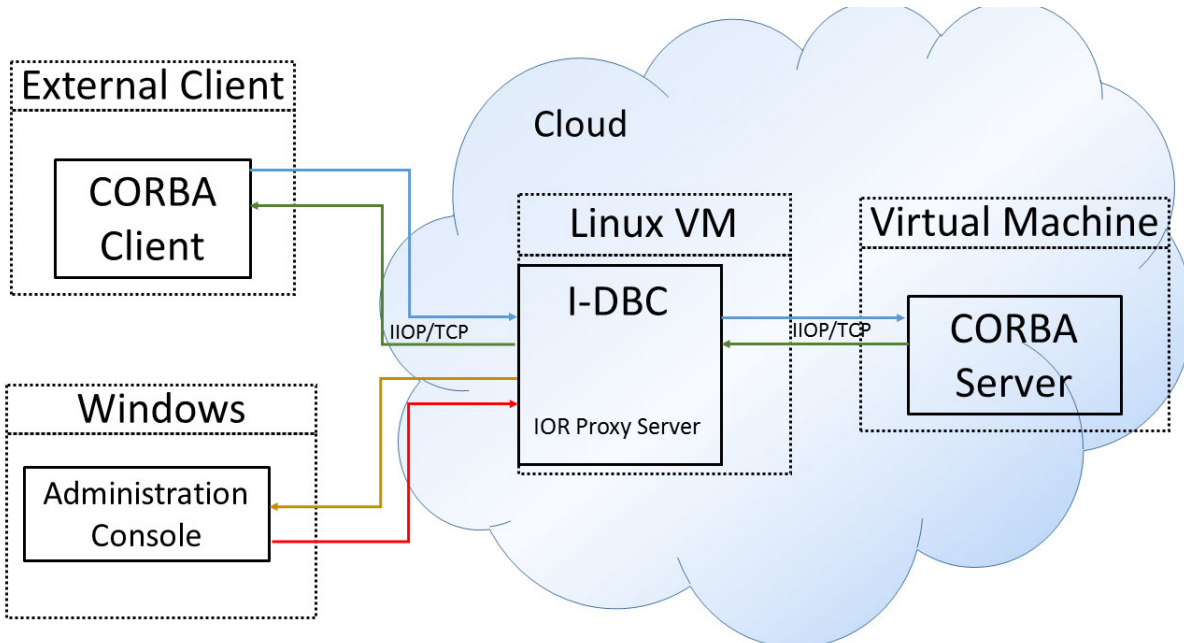
Once the installation is complete and I-DBC is properly configured, the connectivity challenges can be overcome.

CORBA for Cloud and Virtual Environments: Installed Components Overview

After you have installed the CORBA Add-on for Cloud, Containers & Virtual Environments on the appropriate machines, the overall deployment looks like the following illustration.



Suppose the example above chose to install the Administration Console on a Windows system, the result would look like:



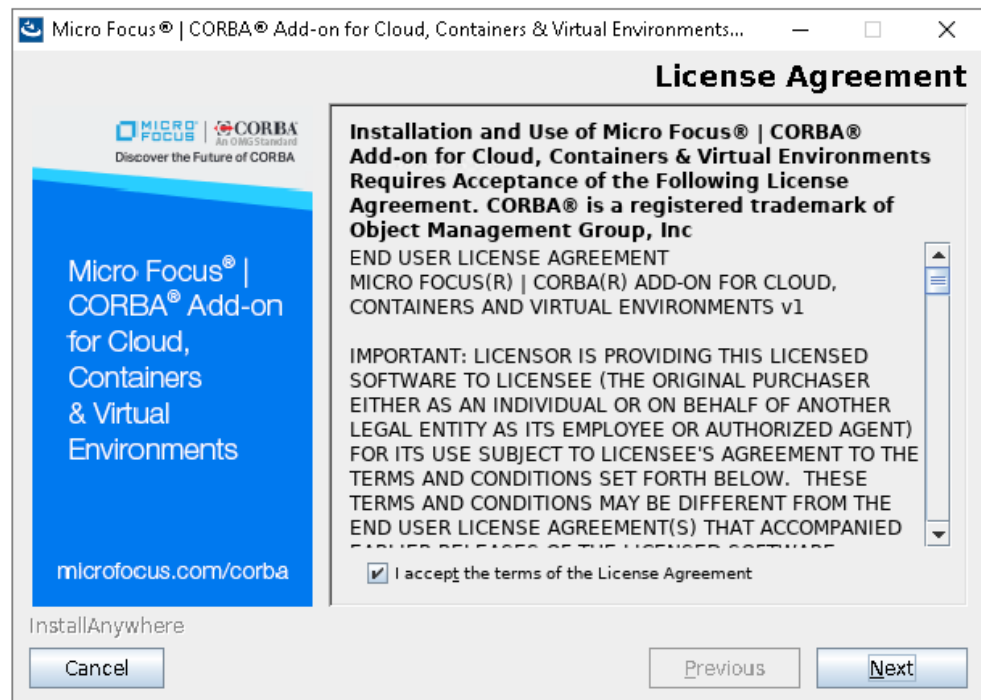
To install the CORBA Add-on for Cloud, Containers & Virtual Environments to the Domain Boundary Controller host, run the installer as follows:

- 1 Download the installer into a temporary directory (for example, `\temp` on Windows, or `/tmp` on UNIX).
- 2 Run the installer to launch InstallAnywhere.
 - On Windows, `mf_ccve_corba_addon_<version>_win_x64.exe`
 - On UNIX, `mf_ccve_corba_addon_<version>_lnx_x64.bin`

Installing with the GUI

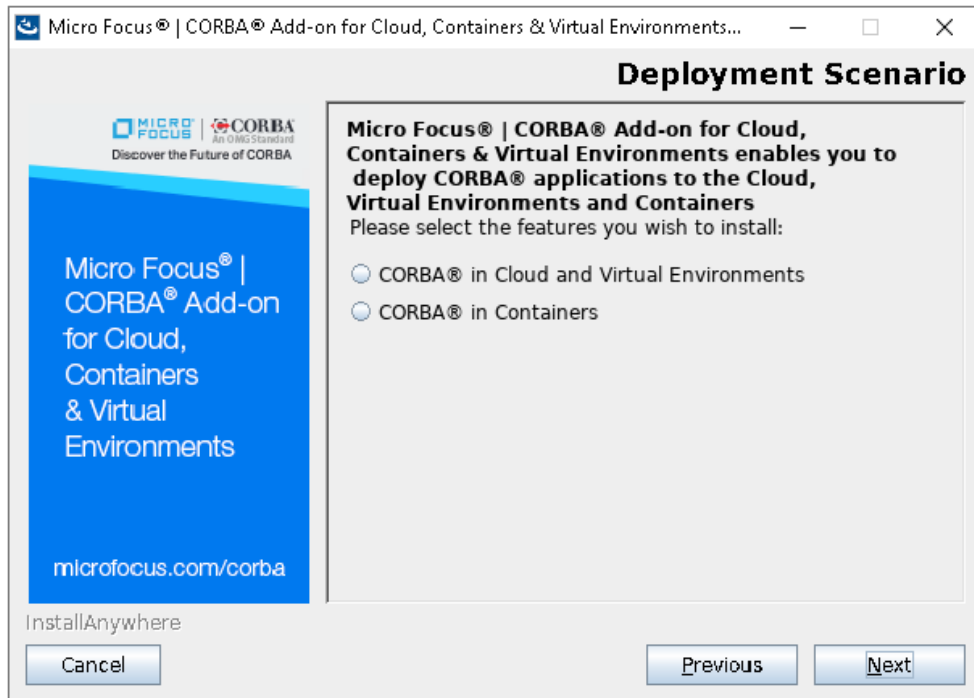
To install via the GUI, run the installer as described above. The installer will run through the following screens.

- 1 The **License Agreement** screen displays.

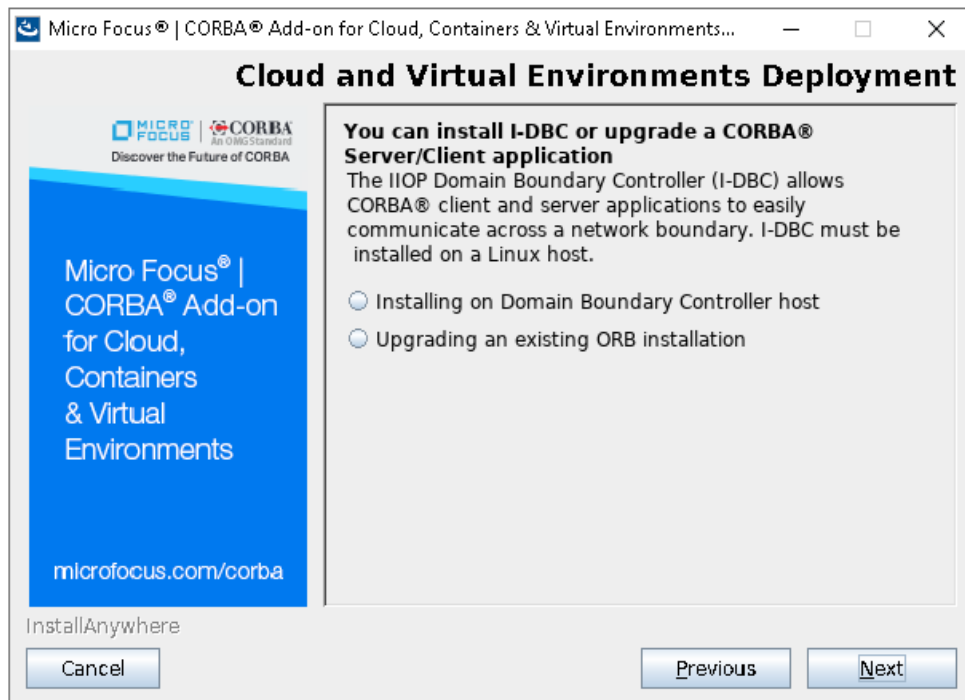


Read and agree the terms of the license agreement. Check **I accept the terms of the License Agreement** and click **Next**. If you do not accept the license, you cannot proceed further.

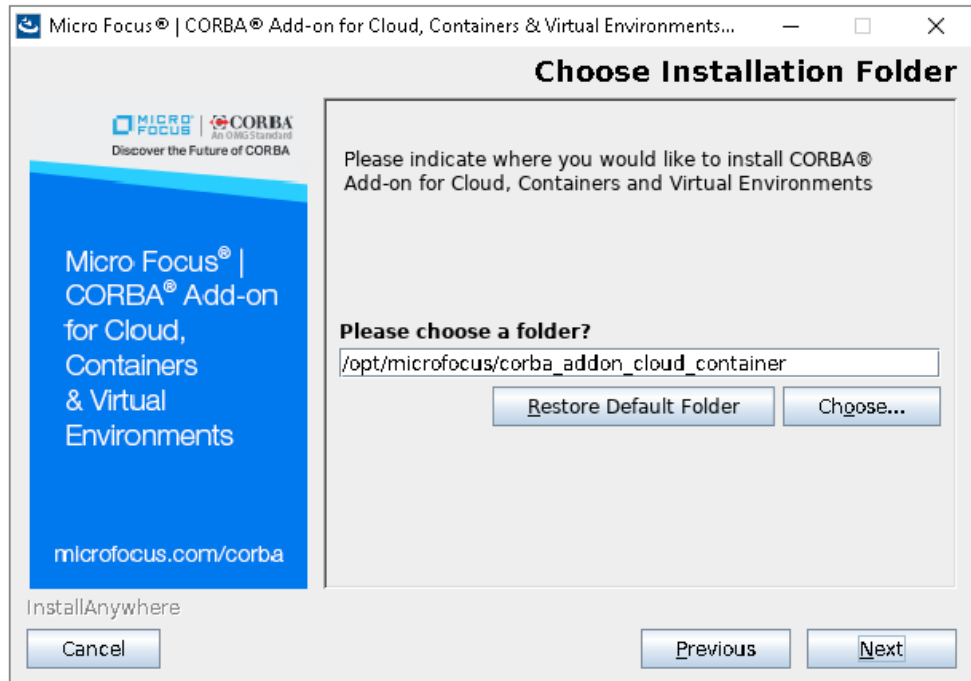
- 2 The **Deployment Scenario** window displays. Select **CORBA in Cloud and Virtual Environments** and click **Next**.



- 3 The **Cloud and Virtual Environments Deployment** window displays. Select **Installing on Domain Boundary Controller host**.

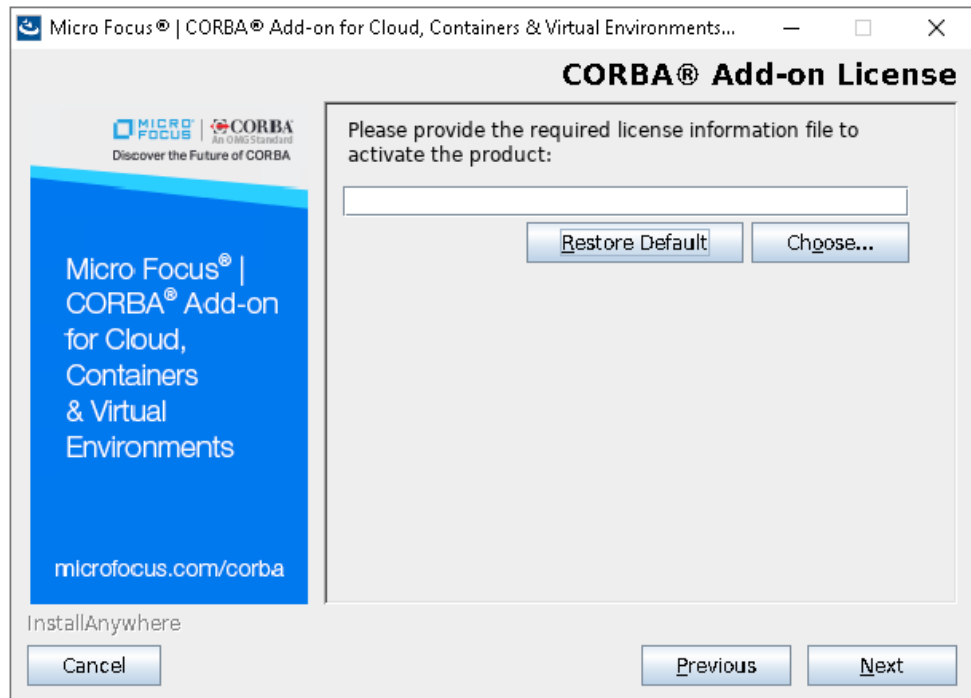


4 The **Choose Installation Folder** screen displays.



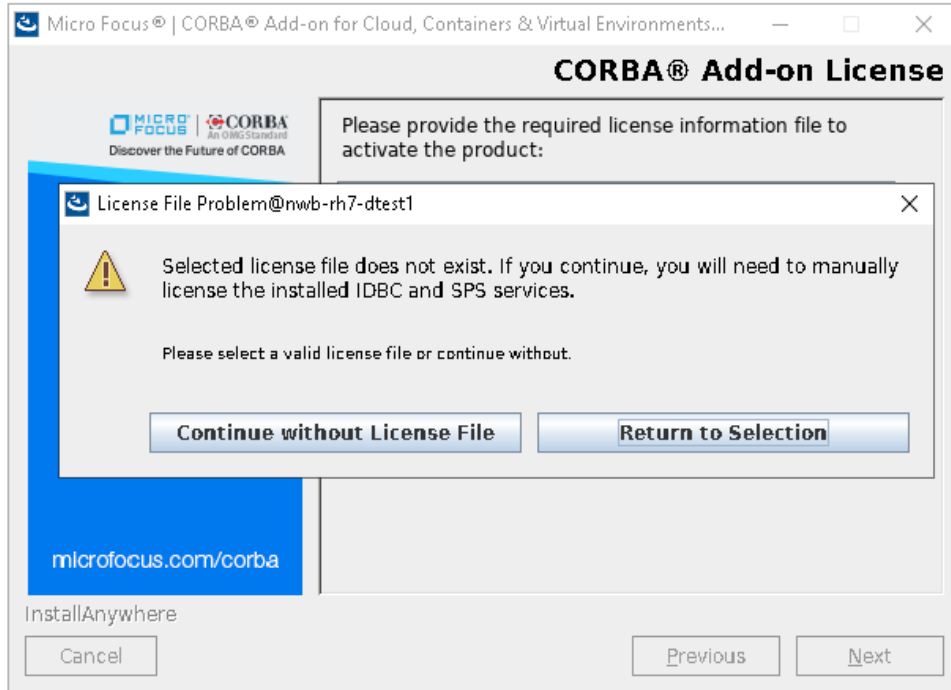
Specify your desired installation directory either by typing the folder name into the text box or by clicking **Choose** to browse for it. Click **Next** to proceed.

5 The **CORBA® Add-on License** screen displays.



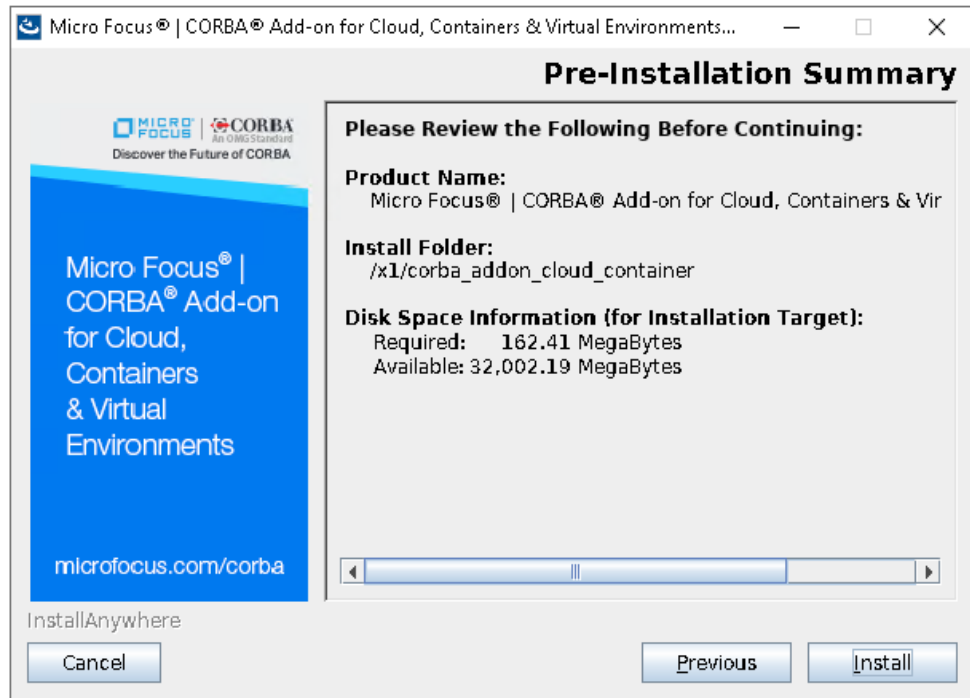
Specify your CORBA® Add-on license.slip file by typing the filename into the textbox or clicking **Choose** to browse for it. Click **Next** to continue.

- 6 If the `license.slip` file you specify does not exist or cannot be found, the following screen is displayed.



If you do not have a `license.slip` file, then you can click the **Continue without License File** button to continue with installing I-DBC. However you will need to manually license the installed I-DBC and SPS services before you can use it. To do this, once you have obtained a `license.slip` file, you need to place it into the license directory for I-DBC.

7 The **Pre-Installation Summary** screen displays.



This screen is shown once all the required installation parameters have been specified. Click **Previous** to move back through earlier screens if any changes need to be made. Click **Install** to proceed with the installation.

8 If more than one network interface is detected, you will see the following screen.



Select the correct IP address for the I-DBC that you wish to configure.

Silent installer properties

As an alternative to the GUI installation, you can use the silent installer as described in [“Installing with the Silent Installer”](#).

The silent installation properties that you need to specify in this case are:

```
USER_INSTALL_DIR=<install location>
INSTALLER_UI=SILENT
INSTALL_CLOUD=1
INSTALL_IDBC=1
CCVE_LICENSE=<license.slip location>
UPGRADE_ORB=0
INSTALL_CONTAINER=0
INSTALL_DOCKER=0
```

If the I-DBC host has multiple network interfaces, use the following properties to configure which IP address will be used with the I-DBC and SPS services.

```
CONFIG_IDBC_HOST=<Network Interface IP>
CONFIG_SPS_HOST=<Network Interface IP>
```

CORBA for Cloud and Virtual Environments: Upgrading existing installations

In order to upgrade an existing VisiBroker, Orbix 3 or Orbix 6 installation to operate with the CORBA Add-on for Cloud, Containers & Virtual Environments, run the installer as follows:

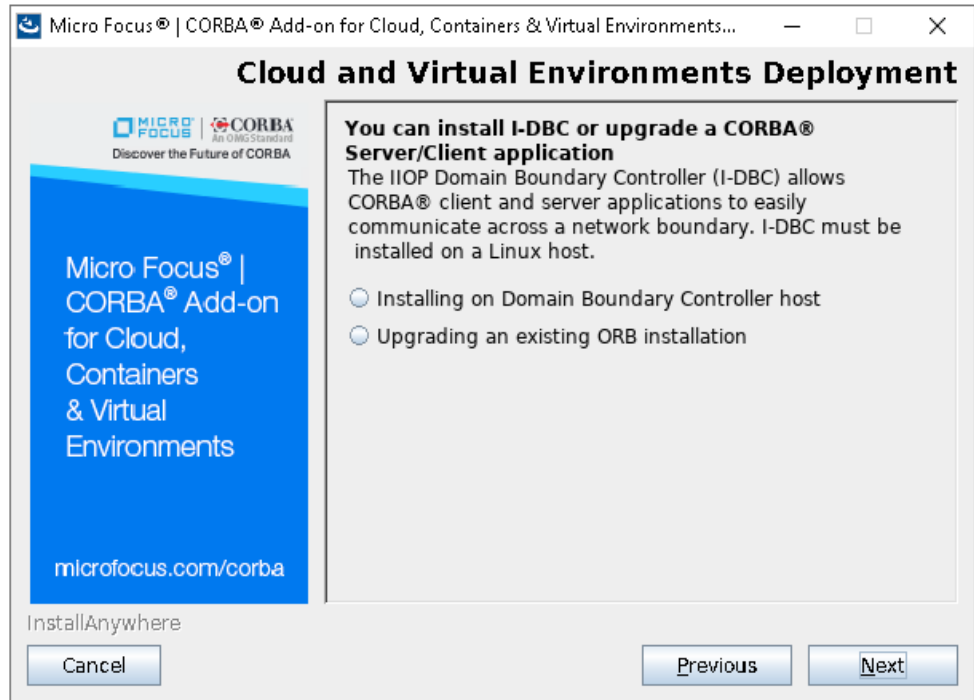
- 1 Download the installer into a temporary directory (for example, `\temp` on Windows, or `/tmp` on UNIX).
- 2 Run the installer.
 - On Windows, `mf_ccve_corba_addon_<version>_win_x64.exe`
 - On UNIX, `mf_ccve_corba_addon_<version>_lnx_x64.bin`

Installing with the GUI

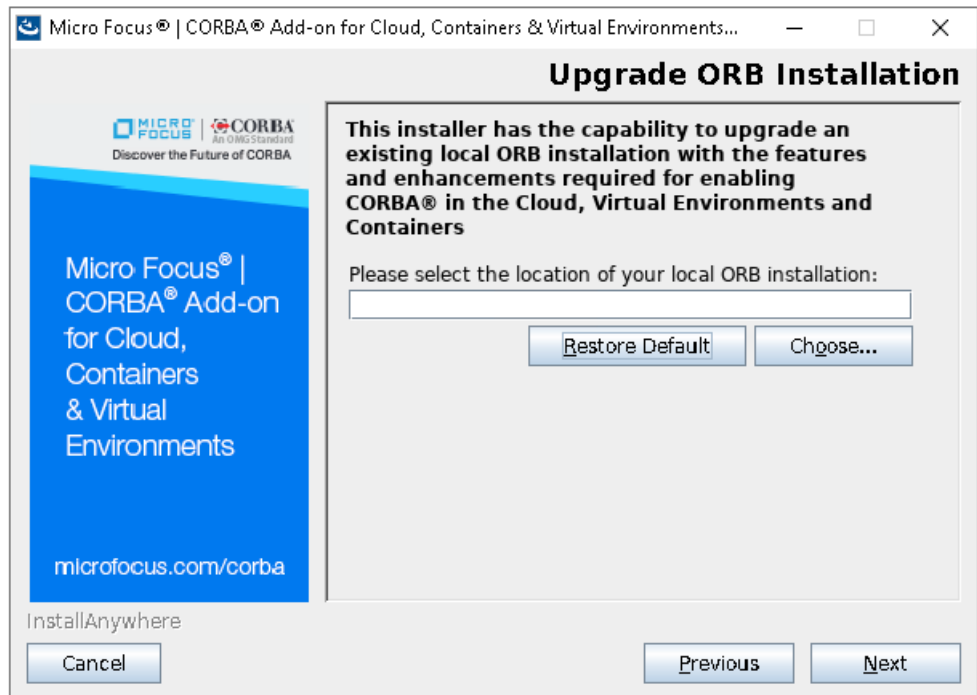
To install via the GUI, run the installer as described above. The installer will run through the following screens.

- 1 The **License Agreement** screen displays, as described in the previous procedure.
- 2 The **Deployment Scenario** window displays. As in the previous procedure, select **CORBA in Cloud and Virtual Environments** and click **Next**.

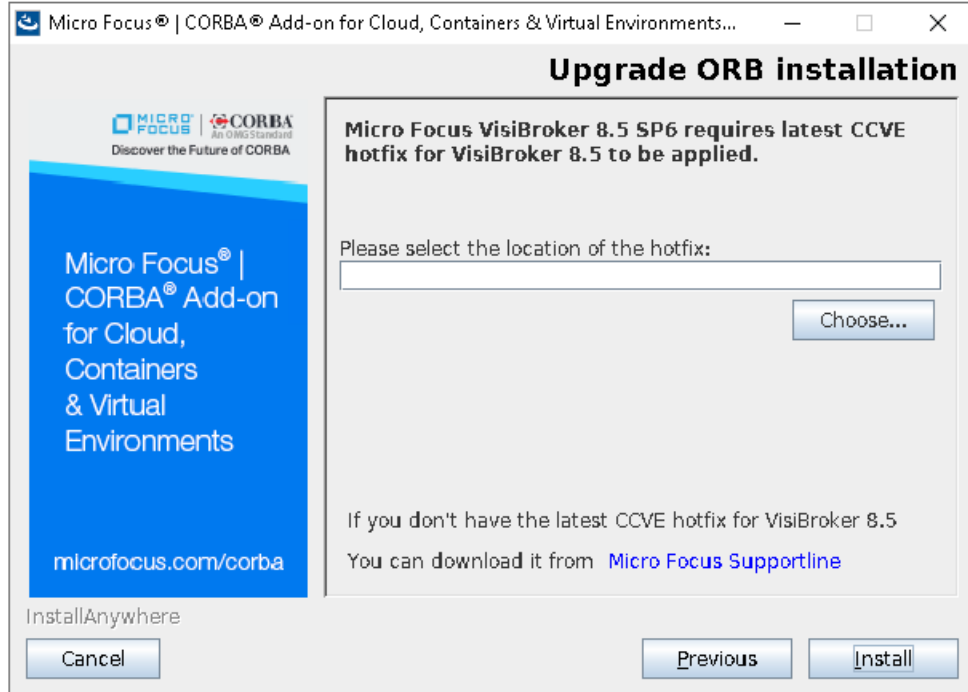
- 3 The **Cloud and Virtual Environments** window displays. This time, select **Upgrading an existing ORB installation**.



- 4 The **Upgrade ORB Installation** window displays. Select the location of a valid ORB to upgrade and click **Next**.



- 5 At the next screen, specify the location of the HotFix to apply in order to upgrade your ORB.



If you have not yet downloaded the necessary HotFix, click the **Micro Focus Supportline** link shown on the screen, and download from there to your local machine. You can now select the HotFix and proceed with the installation.

Silent installer properties

As an alternative to the GUI installation, you can use the silent installer as described in “[Installing with the Silent Installer](#)”.

The silent installation properties that you need to specify in this case are:

```
INSTALLER_UI=SILENT
INSTALL_CLOUD=1
INSTALL_IDBC=0
INSTALL_CONTAINER=0
INSTALL_DOCKER=0
UPGRADE_ORB=1
ORB_INSTALLATION=<orb install location>
CCVE_ADDON_HOTFIX=<hotfix files location>
```

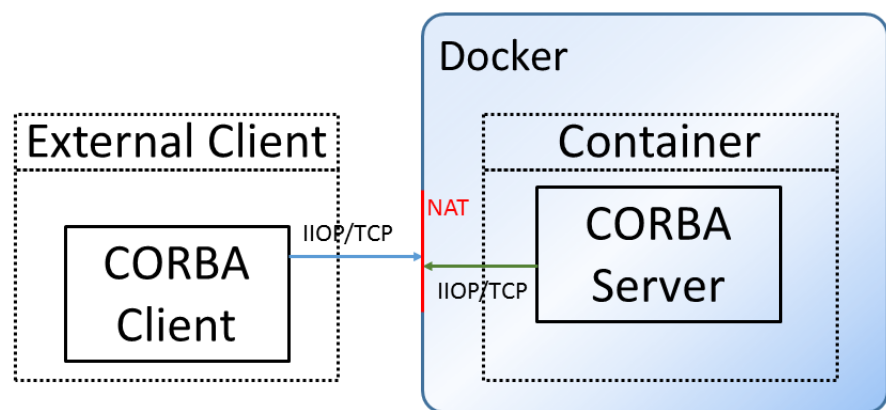
CORBA in Containers

This chapter describes how the CORBA Add-on for Cloud, Containers & Virtual Environments can be used to extend CORBA functionality into the Docker container.

Introduction

The Docker containers can be hosted on either Linux or Windows systems.

In these circumstances, the CORBA server is isolated within a Docker container and is therefore by default unreachable from external CORBA clients.



The CORBA Add-on for Cloud, Containers & Virtual Environments provides a solution to the issues arising when trying to connect CORBA clients running outside of containers to CORBA servers running within containers.

The Docker container will run one of the following operating systems:

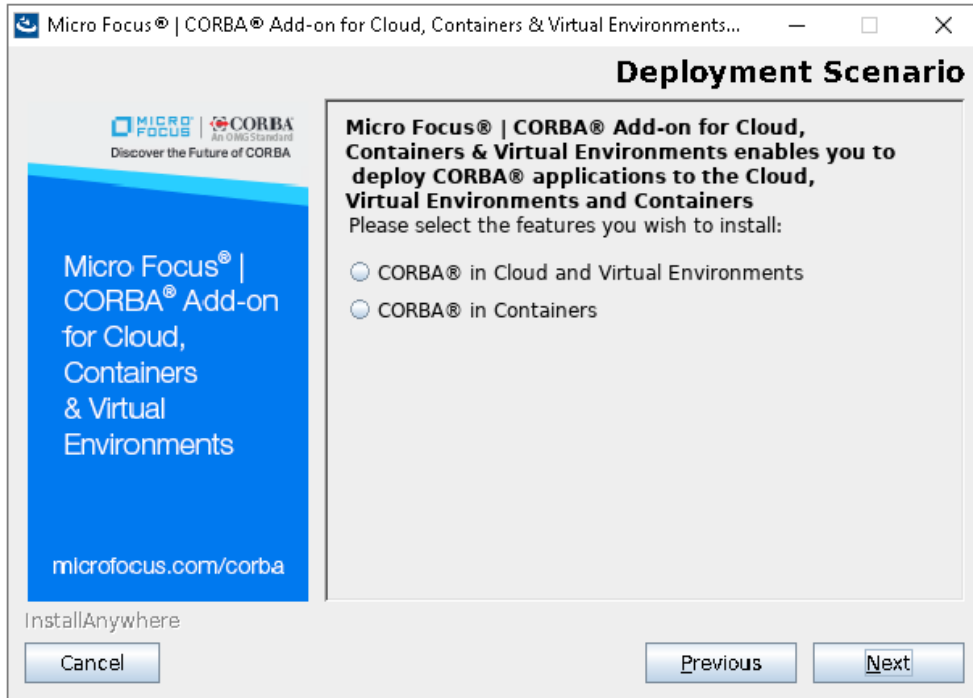
- CentOS
- Ubuntu

The Docker container can be hosted on a system running one of the following operating systems:

- Linux
- Windows 7
- Windows 10

Installation in Containers

In order to deploy the CORBA Add-on for Cloud, Containers & Virtual Environments in a container, run the installer and select the **CORBA in Containers** option at the **Deployment Scenario** window.



Installation Footprint

The CORBA Add-on for Cloud, Containers & Virtual Environments product provides the following components for the CORBA in Containers deployment.

- **Dockerfiles** to enable you to build Docker image layers, from the base operating system layer image all the way to the CORBA application samples layer.
- **Administration Console for I-DBC:** A graphical interface for administering I-DBC. Using the Administration Console is described in the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Administrator's Guide**.
- The capability to upgrade an existing CORBA ORB installation with features and capabilities to communicate with a CORBA application deployed within a container.
- The I-DBC install packages and scripts to be used within a container.
- Administration Console install package to be extracted and installed on the host.

The footprint of the CORBA Add-on for Cloud, Containers & Virtual Environments when installed in a container differs from the Cloud installation, as it installs the Docker folder. This folder contains dockerized

samples, and scripts demonstrating how to dockerize ORB applications. Installing the product installs the following files:

<code>adminconsole/</code>	The extracted Administration Console GUI to administer a I-DBC installation running within a container.
<code>doc/license_agreement.txt</code>	
<code>doc/notices.txt</code>	
<code>docker/</code>	Directory containing all Docker-related assets. See “CORBA for Containers: Installed Components Overview” for the contents of the directory.
<code>docker/common/centos_layer</code>	An operating system image for a CentOS base layer used by the <code>idbc_layer</code> .
<code>docker/common/ubuntu_layer</code>	An operating system image for an ubuntu base layer used by the <code>idbc_layer</code> .
<code>docker/common/idbc_layer_layer</code>	An I-DBC image built on top of the operating system image. I-DBC overcomes the NAT issues. Used by the <code>orb_base_layer</code> .
<code>docker/<orbix3,orbix6,visibroker>/orb_base_layer</code>	A CORBA product image built on top of the I-DBC image. The CORBA product can be one of Orbix 6, Orbix3, or VisiBroker. Used by the <code>application_layer</code> .
<code>docker/<orbix3,orbix6,visibroker>/application_layer/</code>	An application image built on top of the CORBA product image. Several demonstration examples are provided to illustrate how to build your own CORBA-based application inside a Docker container.
<code>kubernetes/</code>	Directory containing demos and samples related to Kubernetes container orchestration. See “Using CORBA in a Kubernetes Environment” .
<code>resources/</code>	The resources sub-directory contains all the components necessary and used within and outside a container. That can be used when creating your own containerized CORBA application.
<code>resources/mf_idbc_install.sh</code>	I-DBC installer script
<code>resources/mf_idbc_services.sh</code>	I-DBC service installer script to be run after I-DBC installation
<code>resources/microfocus_CLI-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources/microfocus_IDBC-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources/microfocus_SPS-4.0.0.tar.gz</code>	I-DBC installer package used by <code>mf_idbc_install.sh</code>
<code>resources/microfocus_AdminConsole.tar.gz</code>	AdminConsole package for manual extraction
<code>uninstall/</code>	Contains the files for uninstallation

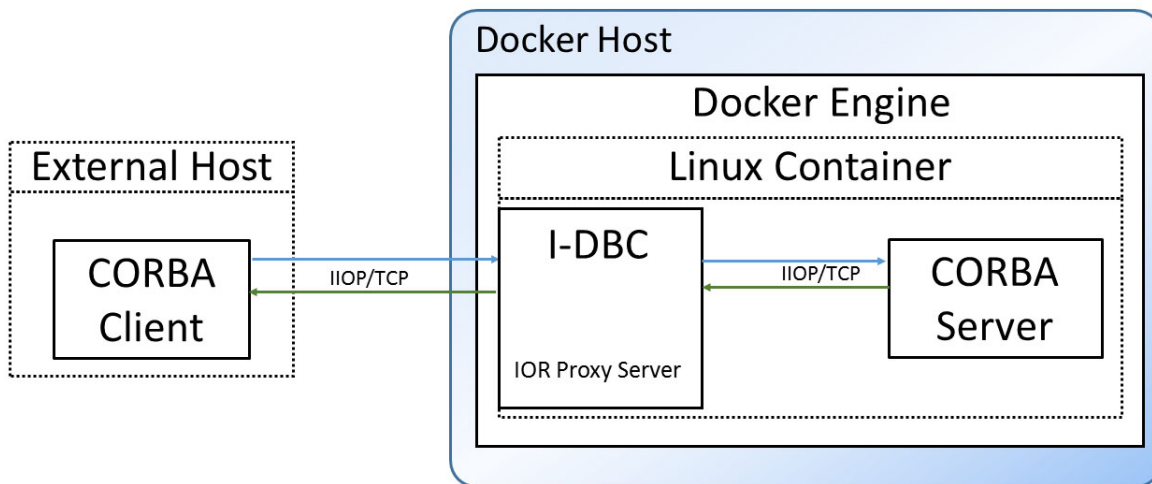
Deployment Scenario for CORBA for Containers

This section describes the infrastructure that makes up a typical Docker deployment:

- **CORBA client machine:** the CORBA client application runs on this machine.

- **CORBA server machine:** the CORBA server application runs within a Docker container hosted in the Docker engine running on this machine. This CORBA server application is isolated by Docker and is not directly reachable to the outside. The I-DBC component deployed within each Docker container makes the CORBA server application available to the outside.
- **Development machine:** this machine hosts the CORBA development environment and the Docker development environment. This machine is used to build Docker container images that can then be deployed to the CORBA server machine.

For a further overview and full information on the capabilities of the I-DBC and Administration Console components, see the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Deployment Guide** and the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Administrator's Guide**, and the chapter "[Common Docker Images](#)".



Installation Prerequisites

To deploy the CORBA Add-on for Cloud, Containers & Virtual Environments, you need the following components:

- The CORBA Add-on for Cloud, Containers & Virtual Environments Linux installer.
- Optionally the CORBA Add-on for Cloud, Containers & Virtual Environments Windows installer.
- A license for the CORBA Add-on for Cloud, Containers & Virtual Environments.
- A Docker development environment.
- An ORB installation. You will need to install an ORB, or upgrade an existing installation to work with a Cloud deployment scenario. The current supported ORB installations are:
 - VisiBroker 8.5.6 or higher for Linux 64-bit installer, or VisiBroker 8.5 for Linux 64-bit GA installer plus VisiBroker 8.5 service pack 6.
 - Orbix 6.3.11 or higher for Linux installer

- Orbix 3.3.15 or higher for Linux 64-bit installer (both Orbix and OrbixSSL installers required).
- The CORBA Add-on for Cloud, Containers & Virtual Environments ORB HotFixes for Linux 64-bit for the ORB runtime deployed within Docker containers (to be downloaded from [Micro Focus Supportline](#)).
- The CORBA Add-on for Cloud, Containers & Virtual Environments ORB HotFixes for the existing ORB client machines (to be downloaded from Micro Focus Supportline). The ORB HotFixes must match the platforms (operating system, compiler, bitness) that your ORB installations are deployed on.

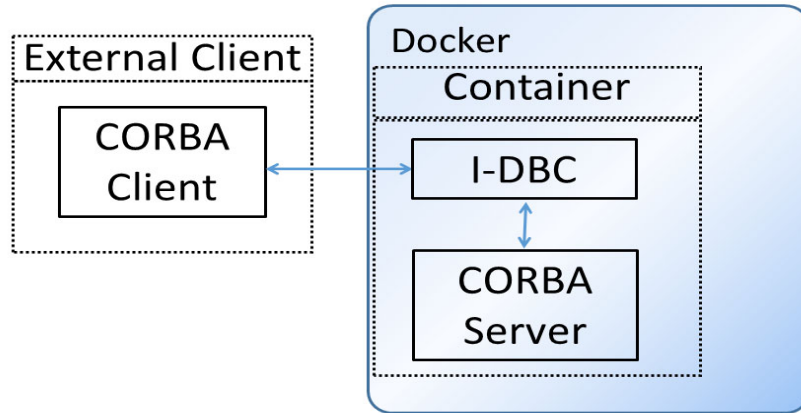
CORBA for Containers: Installed Components Overview

The CORBA Add-on for Cloud, Containers & Virtual Environments for Containers needs to be installed on the Docker development machine and also on the CORBA client machines. See “[CORBA for Containers: Installed Components Overview](#)” for a description of the process.

After you have installed the CORBA Add-on for Cloud, Containers & Virtual Environments, the following components will have been installed in the `docker/` directory (see “[Installation Footprint](#)”):

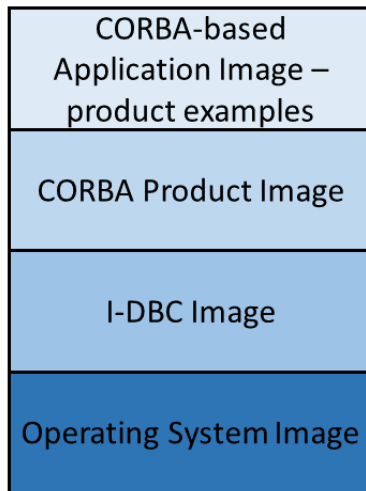
- Dockerfiles to enable you to build Docker images:
 - An operating system image base layer - the operating system base layer is either CentOS or Ubuntu by default.
 - An I-DBC image layer built on top of the operating system image. I-DBC enables connectivity between the Docker isolated network within the container and the outside.
 - An ORB product image layer built on top of the I-DBC image layer. By default, the ORB product can be either:
 - Orbix 3
 - Orbix 6
 - VisiBroker
 - An application image layer built on top of the ORB product image. Several demonstration examples are provided to illustrate how to build your own CORBA server application inside a Docker container.
- Scripts to support configuration and running of I-DBC, the ORB, and CORBA server applications inside a Docker container.
- Enhancements and fixes to ORB products to allow them to work with I-DBC inside a Docker container, such as the VisiBroker OSAgent relay component.
- The I-DBC and SPS components to enable cross-containers connectivity. These allow CORBA based clients and servers to easily communicate across a network boundary where Network Address Translation (NAT) is occurring.
- The Administration Console for I-DBC. This is a graphical interface for administering I-DBC.

Docker application image layers built with these tools can be deployed and run on any Docker host machine. CORBA server applications run inside these containers can now be transparently accessed from CORBA clients.



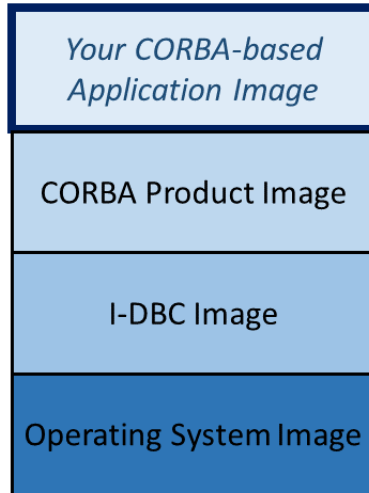
The Docker product examples provide several README files that explain how to build Docker images. Follow through the examples to gain an understanding of how to build the Docker images and run Docker containers using those images.

The final Docker image is composed of multiple Docker images:



Once you get a basic understanding of how to build Docker images that can run CORBA product examples, you can follow the examples as a guide to building a Docker image that runs your particular CORBA-based application.

You replace the “product examples” image with an image that runs your CORBA-based application.



Installation

The CORBA Add-on for Cloud, Containers & Virtual Environments for Containers needs to be installed on the Docker development machine and also on the CORBA client machines.

- 1 On the Docker development machine, run the installer selecting the **CORBA in Containers** deployment scenario.
- 2 On the CORBA client machines, run the installer selecting the **CORBA in Containers** deployment scenario to upgrade an existing ORB installation.

Installing on the Docker Development machine

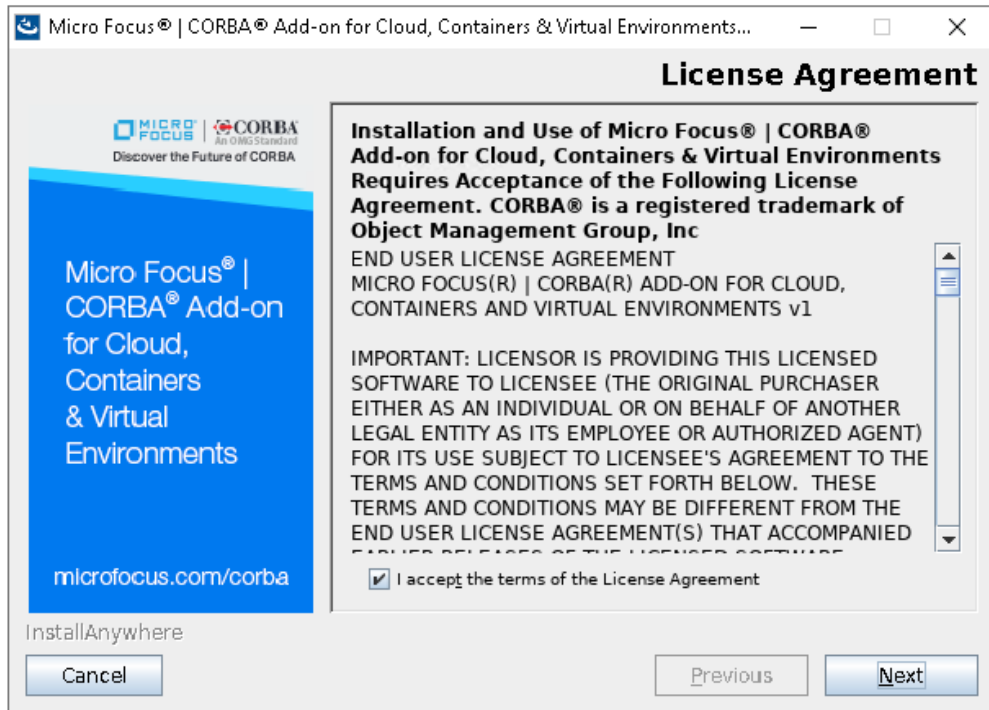
To install the CORBA Add-on for Cloud, Containers & Virtual Environments to the Docker development machine, run the installer as follows:

- 1 Download the installer into a temporary directory (for example, `\temp` on Windows, or `/tmp` on UNIX).
- 2 Run the installer to launch InstallAnywhere.
 - On Windows, `mf_ccve_corba_addon_<version>_win_x64.exe`
 - On UNIX, `mf_ccve_corba_addon_<version>_lnx_x64.bin`

Installing with the GUI

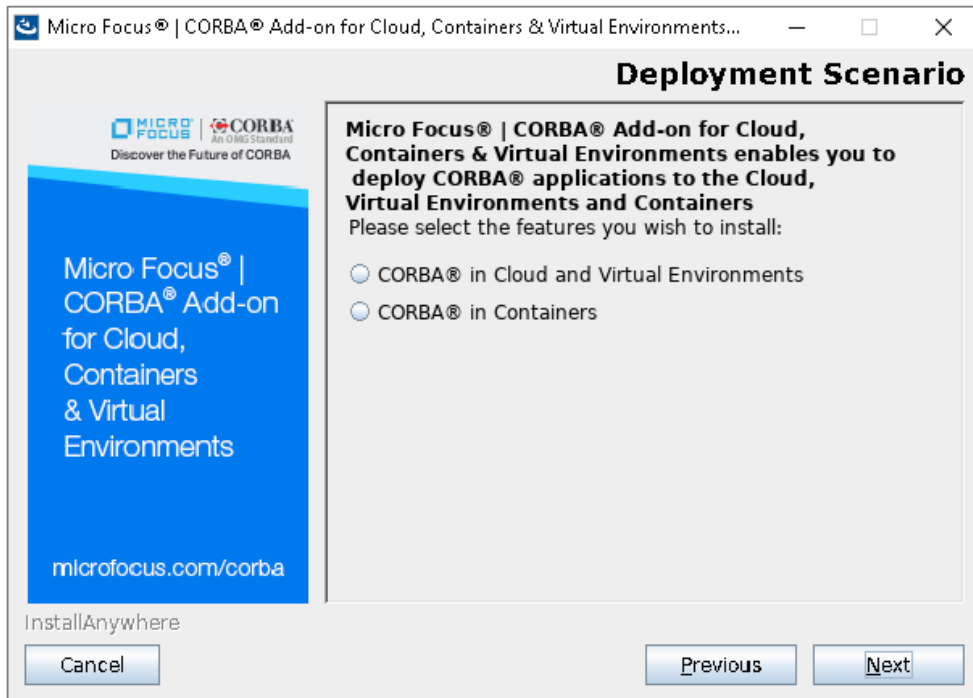
To install via the GUI, run the installer as described above. The installer will run through the following screens.

1 The **License Agreement** screen displays.

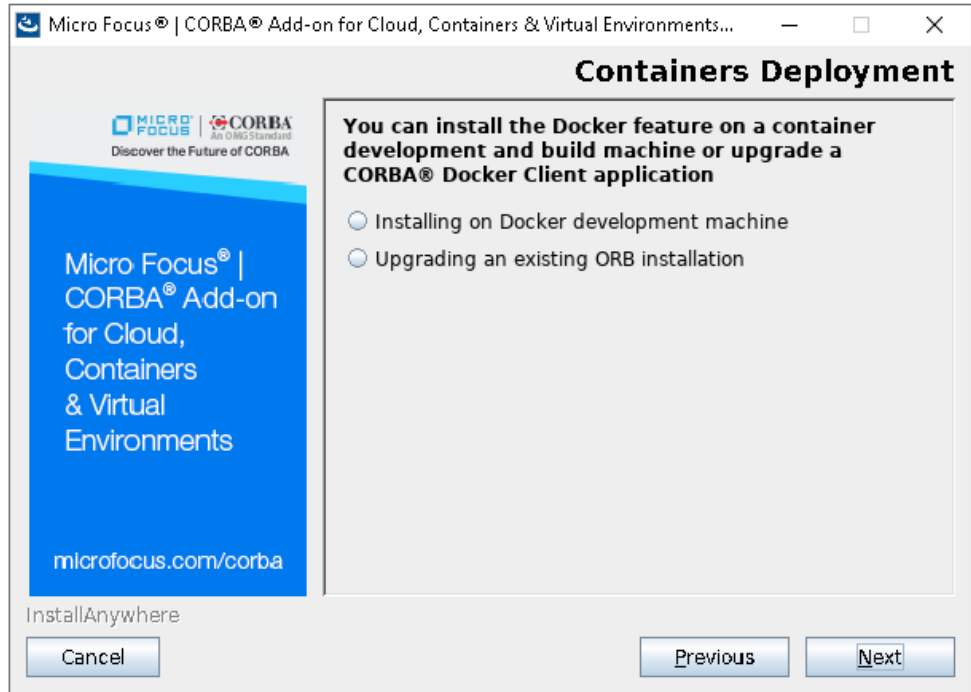


Read and agree the terms of the license agreement. Check **I accept the terms of the License Agreement** and click **Next**. If you do not accept the license, you cannot proceed further.

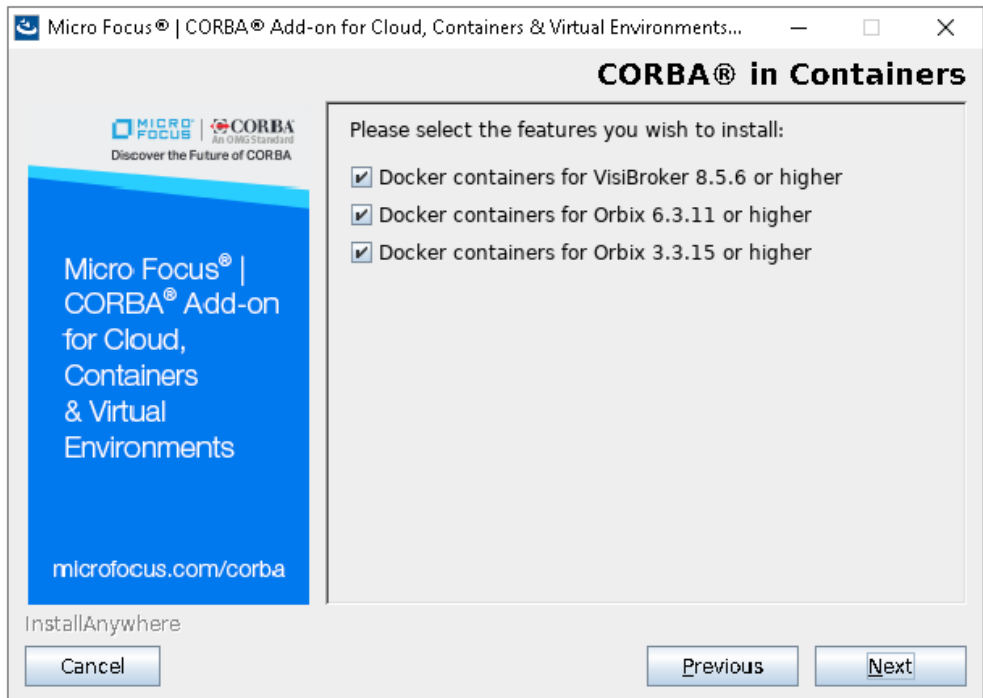
2 The **Deployment Scenario** window displays. Select **CORBA® in Containers** and click **Next**.



3 The **Containers Deployment** screen is displayed. Select **Installing on Docker development machine** and click **Next**.



4 The **CORBA® in Containers** screen displays.

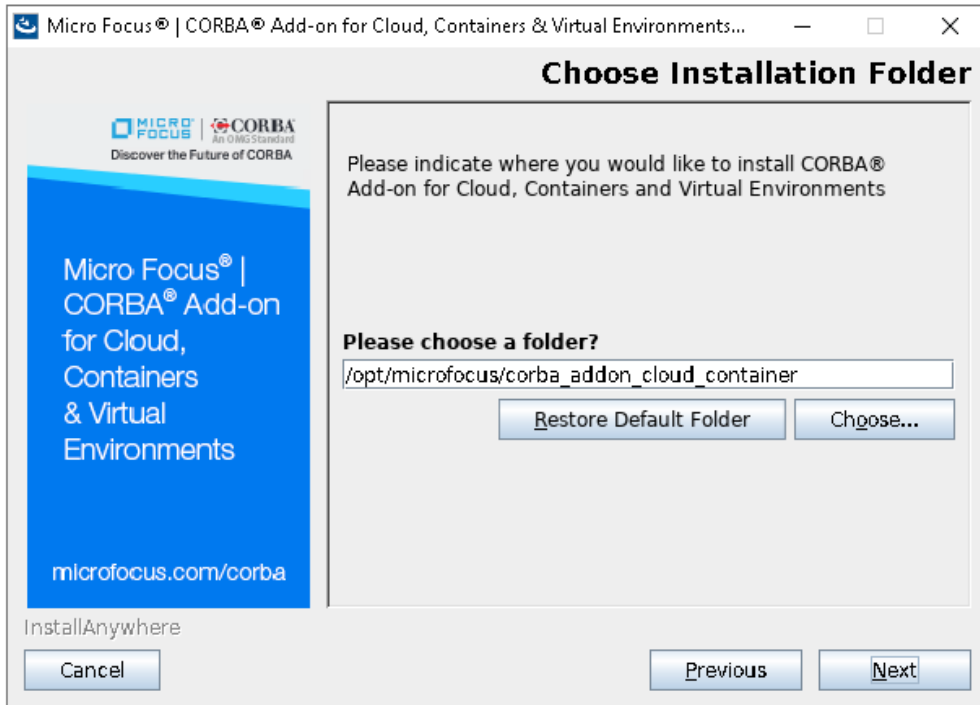


Here you can tick all, some, or none of the options presented. Select the option for the ORB or ORBs you wish to use.

Note:

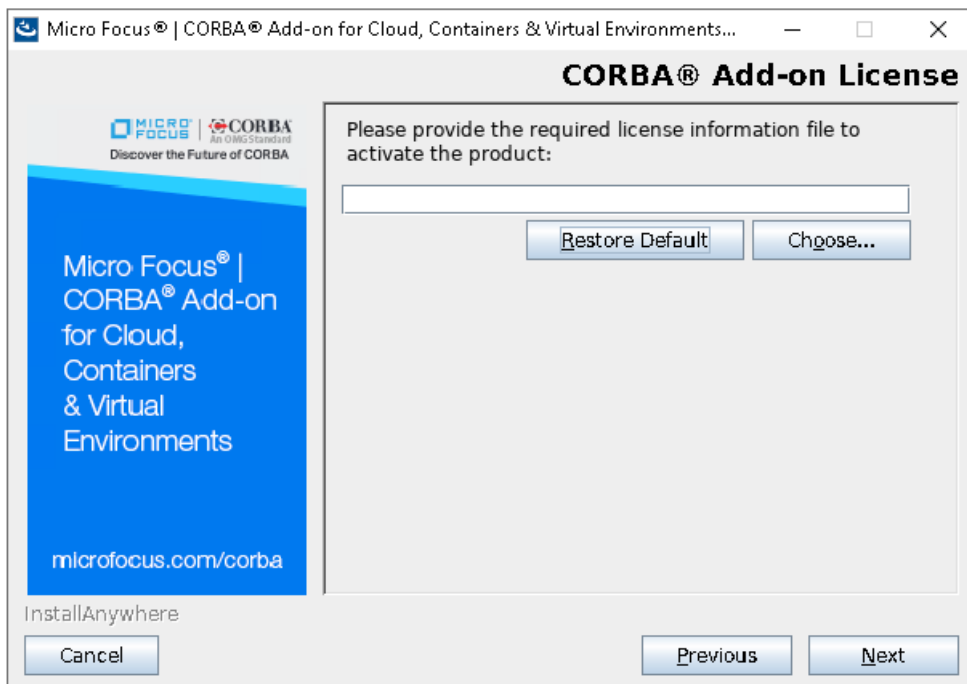
If you select no options, Docker common layers (OS layer and I-DBC layer) are installed, giving you a starting point to develop an ORB layer that could be built on Orbacus, JacORB or TAO or any other ORB runtime.

5 The Choose Installation Folder screen displays.



Specify your desired installation directory either by typing into the text field or by clicking **Choose** to browse for it. Click **Next** to proceed.

6 The CORBA® Add-on License screen displays.



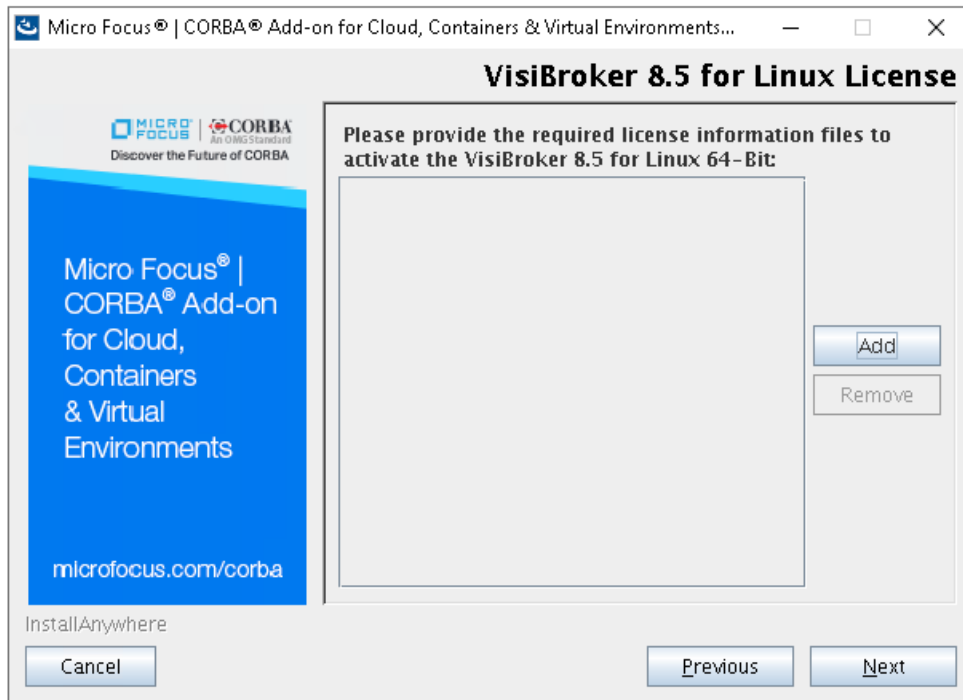
Specify your CORBA® Add-on license file by typing the filename into the textbox or clicking **Choose** to browse for it. Click **Next** to continue.

- 7 If you selected VisiBroker in step 4, the **VisiBroker Installer** screen displays.



Select the location of the VisiBroker installer and the location of the HotFix to apply in order to upgrade the VisiBroker ORB. If you have not yet downloaded the necessary HotFix, click the **Micro Focus Supportline** link provided on the screen, and download from there to your local machine. You can now select the hotfix and proceed with the installation. Click **Next**.

8 The **VisiBroker License** screen displays.



Enter the location of your VisiBroker license file, or the locations of multiple license files for different VisiBroker components. You can add the location of the required license files by clicking **Add** and remove an entry by clicking **Remove**.

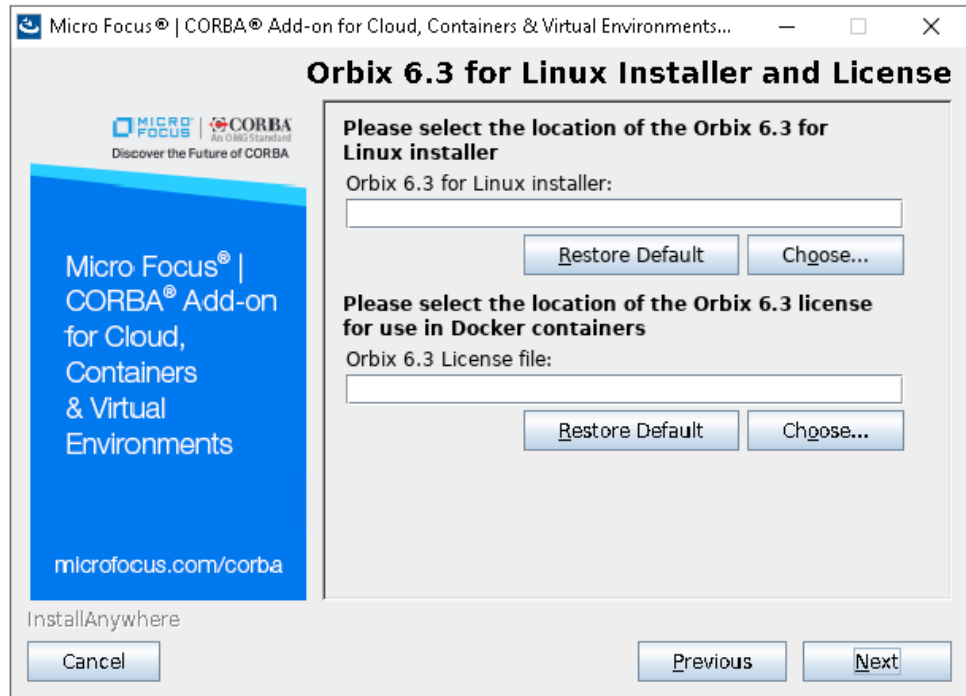
Click **Next** to continue.

9 The **Additional VisiBroker container HotFixes** screen displays.



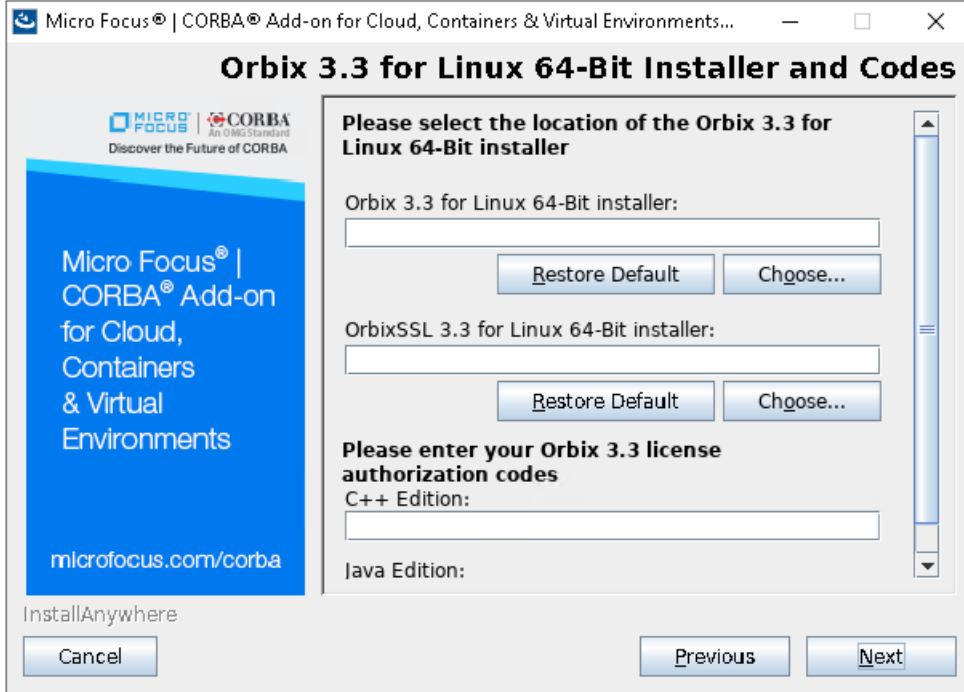
If you have HotFixes that need to be applied to the VisiBroker installation within Docker containers, you can specify them here. Select the HotFix and click **Add** if there are any more to apply. The HotFixes will be applied in the order they are listed in the text box (top to bottom). Use the **Up** and **Down** buttons to adjust the sequence. To specify HotFix(es), click the **Add** button. When you have listed all of them in the correct order, click **Next**.

- 10 If you selected Orbix 6 in step 4, the **Orbix 6.3 Installer and License** screen displays.



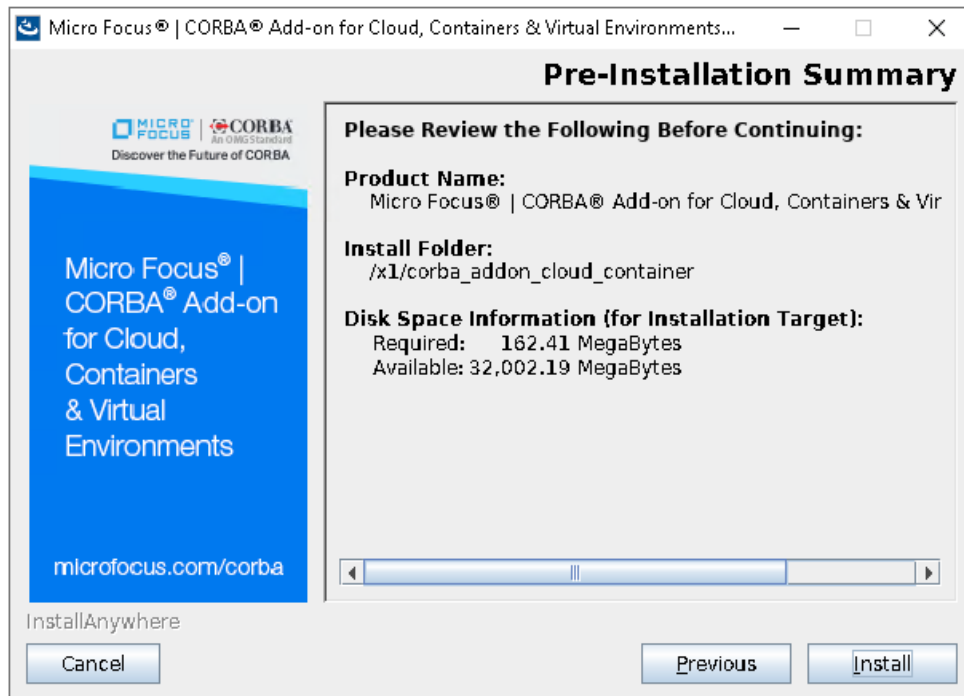
Enter the locations both of the Linux 64-bit installer and of the license file in this screen. You can type the filenames directly into the textboxes, or browse for them by clicking **Choose**. Once both files have been specified, click **Next** to continue.

11 If you selected Orbix 3 in step 4, the **Orbix 3.3 Installer and Codes** screen displays.



Enter the locations both of the Orbix 3.3 and the OrbixSSL 3.3 Linux 64-bit installers and of the C++ and Java license codes in this screen. You can type the filenames directly into the textboxes, or browse for them by clicking **Choose**. Once both files have been specified, click **Next** to continue.

12 The **Pre-Installation Summary** screen displays.



This screen is shown once all the required installation parameters have been specified. Click **Previous** to move back through earlier panels if any changes need to be made. Click **Install** to proceed with the installation.

Silent installer properties

As an alternative to the GUI installation, you can use the silent installer as described in [“Installing with the Silent Installer”](#).

The silent installation properties that you need to specify in this case are:

```
USER_INSTALL_DIR=<install location>
INSTALLER_UI=SILENT
INSTALL_CLOUD=0
INSTALL_IDBC=0
UPGRADE_ORB=0
INSTALL_CONTAINER=1
INSTALL_DOCKER=1
CCVE_LICENSE=<license.slip location>
INSTALL_VB_DOCKER=1
VISI_INSTALLER=<VB 8.5.6 Linux 64 Bit installer location>
VISI_LICENSE=<Comma separated license files location list>
VISI_CCVE_ADDON_HOTFIX=<ccve hotfix for VB 8.5.6 Linux 64 Bit location>
VISI_HOTFIX_LIST=<Comma separated customer hotfix files location list>
INSTALL_O6_DOCKER=1
ORBIX6_INSTALLER=<Orbix 6.3.11 Linux installer location>
ORBIX6_LICENSE=<license.slip location>
INSTALL_O3_DOCKER=1
ORBIX3_INSTALLER=<Orbix 3.3.15 Linux 64 Bit installer location>
ORBIXSSL3_INSTALLER=<OrbixSSL 3.3.15 Linux 64 Bit installer location>
ORBIX3_CXX_KEY=<product code>
ORBIX3_JAVA_KEY=<product code>
```

Upgrading the client side ORB installation

The client upgrade installs additional demos and examples for Docker and Kubernetes into the existing ORB installation’s demo examples directory. In addition the ORB installation might require also the latest CCVE HotFix. The installer will guide you through the upgrade process

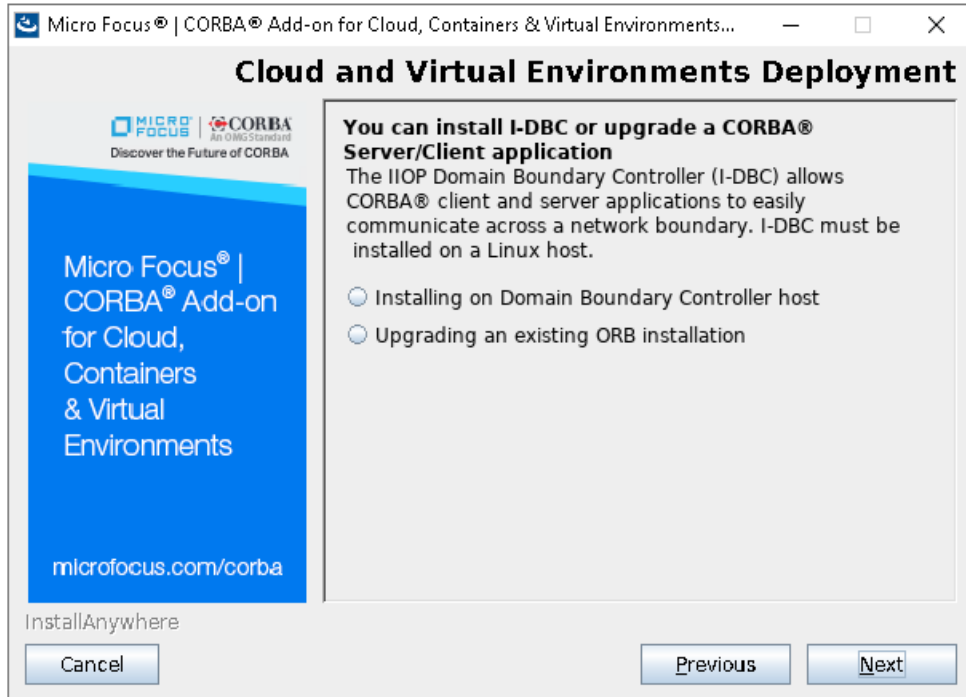
To upgrade the client side ORB installation, run the installer as follows:

- 1 Download the installer into a temporary directory (for example, `\temp` on Windows, or `/tmp` on UNIX).
- 2 Run the installer to launch InstallAnywhere.
 - On Windows, `mf_ccve_corba_addon_<version>_win_x64.exe`
 - On UNIX, `mf_ccve_corba_addon_<version>_lnx_x64.bin`

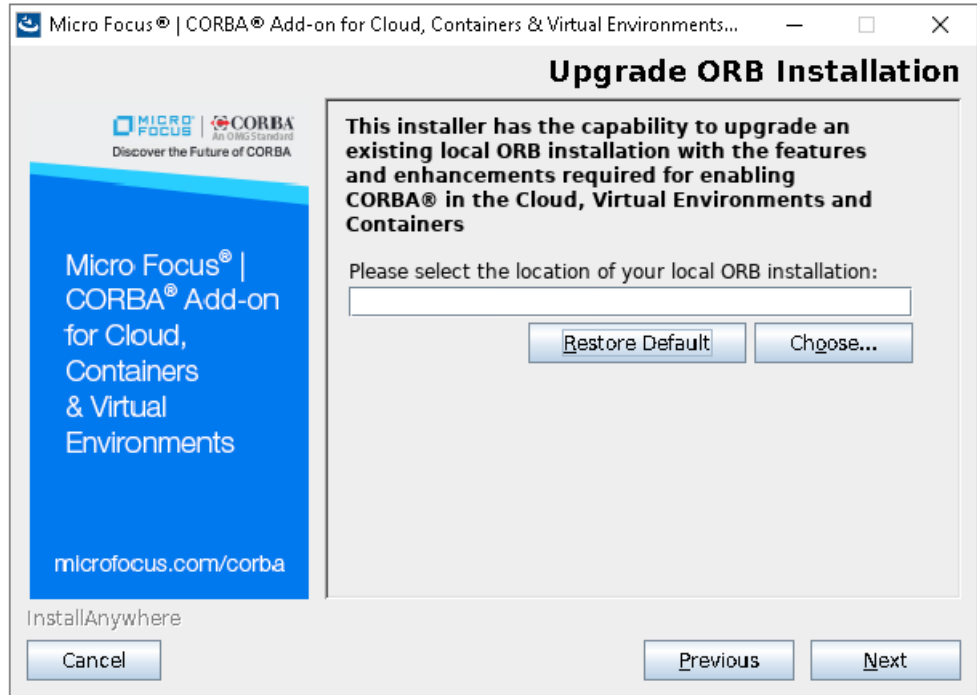
Installing with the GUI

To install via the GUI, run the installer as described above. The installer will run through the following screens.

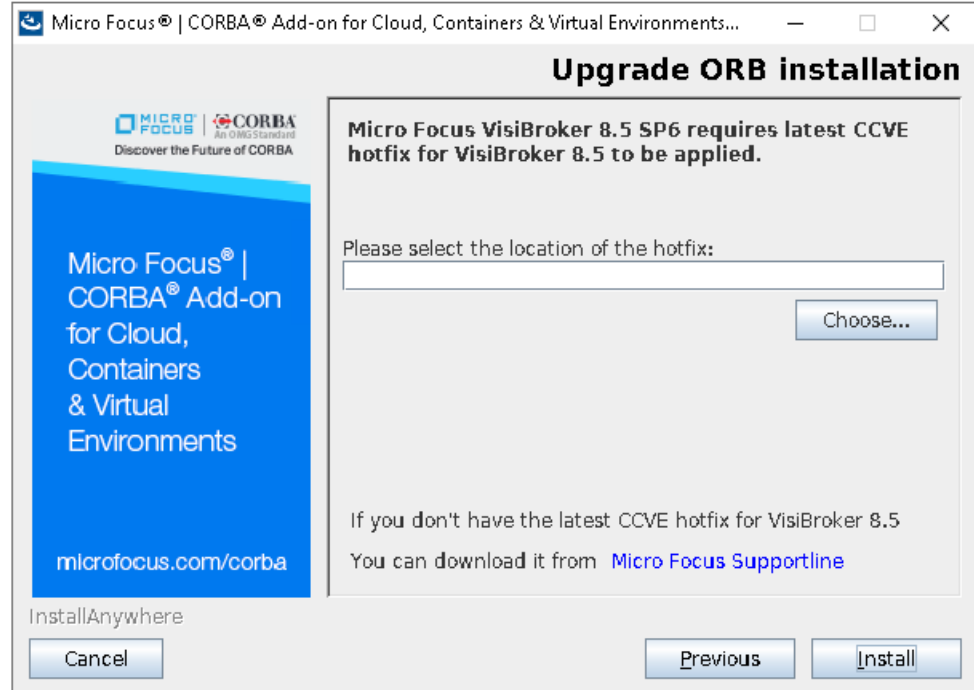
- 1 The **License Agreement** screen displays, as described in the previous procedure.
- 2 The **Deployment Scenario** window displays. As in the previous procedure, select the **CORBA in Containers** option and click **Next**.
- 3 The **Containers Deployment** screen is displayed. Select the **Upgrading an existing ORB Installation** option and click **Next**.



- 4 The **Upgrade ORB Installation** window displays. Select the location of a valid ORB to upgrade and press **Next**.



- 5 At the next screen, specify the location of the HotFix to apply in order to upgrade your ORB.



If you have not yet downloaded the necessary HotFix, click the **Micro Focus Supportline** link provided on the screen, and download from there to your local machine. You can now select the HotFix and proceed with the installation.

Silent installer properties

As an alternative to the GUI installation, you can use the silent installer as described in [“Installing with the Silent Installer”](#).

The silent installation properties that you need to specify in this case are:

```
INSTALLER_UI=SILENT
INSTALL_CLOUD=0
INSTALL_IDBC=0
INSTALL_CONTAINER=1
INSTALL_DOCKER=0
UPGRADE_ORB=1
ORB_INSTALLATION=<orb install location>
CCVE_ADDON_HOTFIX=<hotfix files location>
```

Silent Installation

This chapter describes the silent installer. For further installation information, see [CORBA in the Cloud or in Virtual Environments](#) and [CORBA in Containers](#).

The silent installer is available for both Windows and Linux operating systems.

Installing with the Silent Installer

As an alternative to the GUI described in [CORBA in the Cloud or in Virtual Environments](#) and [CORBA in Containers](#), the installation can be performed in silent mode. A silent installation runs without user interaction, and is typically used to automate installation across multiple machines. Instead of specifying the installation parameters via the interface, the parameters are stored in an installer properties file. The properties and values required will vary greatly depending on what sort of installation you are carrying out. See the [CORBA in the Cloud or in Virtual Environments](#) and [CORBA in Containers](#) chapters for details of those properties.

Sample installer properties file

To perform a silent installation, you must prepare an installer properties file that contains the required information. Each line consists of a property name and a property value, separated by an equals sign. A line can be commented out (or a descriptive comment added) by placing a hash mark (#) at the start of the line. A sample file is below:

```
USER_INSTALL_DIR=<install location>
INSTALLER_UI=SILENT
INSTALL_CLOUD=0
INSTALL_IDBC=0
#CONFIG_IDBC_HOST=<Network Interface IP>
#CONFIG_SPS_HOST=<Network Interface IP>
UPGRADE_ORB=0
#ORB_INSTALLATION=<orb install location>
#CCVE_ADDON_HOTFIX=<hotfix files location>
INSTALL_CONTAINER=1
INSTALL_DOCKER=1
CCVE_LICENSE=<license.slip location>
INSTALL_VB_DOCKER=1
VISI_INSTALLER=<VB 8.5.6 Linux 64 Bit installer location>
VISI_LICENSE=<Comma separated license files location list>
VISI_CCVE_ADDON_HOTFIX=<ccve hotfix for VB 8.5.6 Linux 64 Bit location>
VISI_HOTFIX_LIST=<Comma separated customer hotfix files location list>
INSTALL_O6_DOCKER=1
ORBIX6_INSTALLER=<Orbix 6.3.11 Linux installer location>
ORBIX6_LICENSE=<license.slip location>
INSTALL_O3_DOCKER=1
ORBIX3_INSTALLER=<Orbix 3.3.15 Linux 64 Bit installer location>
ORBIXSSL3_INSTALLER=<OrbixSSL 3.3.15 Linux 64 Bit installer location>
ORBIX3_CXX_KEY=<product code>
ORBIX3_JAVA_KEY=<product code>
```

The meaning of each property is described below. In the chapters [CORBA in the Cloud or in Virtual Environments](#) and [CORBA in Containers](#), the descriptions of each installation scenario set out which properties require which values.

Note that for Windows directory paths, the backslash directory separator must be escaped and specified by using \$ and /. This is required by InstallAnywhere. For example:

```
USER_INSTALL_DIR=C:/$MainDirectory/$SubDirectory/$CCVE
```

Property name	Description
USER_INSTALL_DIR	Specifies the target directory for the installation.
INSTALLER_UI	Specifies the type of installation. For a silent installation, this must have the value <code>silent</code> .
INSTALL_CLOUD	Specifies whether the Cloud and Virtual Environments installer scenario will be used (1) or not (0).
INSTALL_CONTAINER	Specifies whether the Container installer scenario will be used (1) or not (0).
INSTALL_IDBC	Specifies whether IDBC/SPS/CLI services will be installed (1) or not (0). Used only with <code>INSTALL_CLOUD=1</code> .
CONFIG_IDBC_HOST	Specifies the IP address to use for the I-DBC service, if there is more than one possible on the I-DBC host. Used only with <code>INSTALL_IDBC=1</code> on Linux platforms.
CONFIG_SPS_HOST	Specifies the IP address to use for the SPS service, if there is more than one possible on the I-DBC host. Used only with <code>INSTALL_IDBC=1</code> on Linux platforms.
CCVE_LICENSE	Specifies the location of the CCVE license file. Used with <code>INSTALL_IDBC=1</code> on Linux platforms or <code>INSTALL_DOCKER=1</code> .
UPGRADE_ORB	Specifies whether an existing Orbix6, Orbix3 or VisiBroker 8.5 installation will be upgraded (1) or not (0). Only to be used with <code>INSTALL_IDBC=0</code> and <code>INSTALL_DOCKER=0</code> .
ORB_INSTALLATION	Specifies the location of the existing CORBA product to be upgraded. Used only with <code>UPGRADE_ORB=1</code> .
CCVE_ADDON_HOTFIX	Specifies the location of the product-specific CCVE hotfix available from Micro Focus Support. Used only with <code>UPGRADE_ORB=1</code> .
INSTALL_DOCKER	Specifies whether Docker examples will be installed (1) or not (0). Used only with <code>INSTALL_CONTAINER=1</code> .
INSTALL_VB_DOCKER	Specifies whether the Docker examples for VisiBroker will be installed (1) or not (0).
VISI_INSTALLER	Specifies the location of the VisiBroker installer binary. Used only with <code>INSTALL_VB_DOCKER=1</code> .
VISI_LICENSE	Specifies the list of VisiBroker license files, separated by commas. Used only with <code>INSTALL_VB_DOCKER=1</code> .
VISI_CCVE_ADDON_HOTFIX	Specifies the location of the CCVE Linux hotfix for VisiBroker available from Micro Focus Support. Used only with <code>INSTALL_VB_DOCKER=1</code> .
VISI_HOTFIX_LIST	Specifies the list of VisiBroker hotfix files, separated by commas. Used only with <code>INSTALL_VB_DOCKER=1</code> .
INSTALL_O6_DOCKER	Specifies whether the Docker examples for Orbix 6 will be installed (1) or not (0).
ORBIX6_INSTALLER	Specifies the location of the Orbix 6 installer binary. Used only with <code>INSTALL_O6_DOCKER=1</code> .
ORBIX6_LICENSE	Specifies the location of the Orbix 6 license file. Used only with <code>INSTALL_O6_DOCKER=1</code> .
INSTALL_O3_DOCKER=1	Specifies whether the Docker examples for Orbix 3 will be installed (1) or not (0).

Property name	Description
ORBIX3_INSTALLER	Specifies the location of the Orbix 3 installer binary. Used only with <code>INSTALL_O3_DOCKER=1</code> .
ORBIXSSL3_INSTALLER	Specifies the location of the Orbix 3 SSL installer binary. Used only with <code>INSTALL_O3_DOCKER=1</code> .
ORBIX3_CXX_KEY	Specifies the C++ authentication key for Orbix 3. Used only with <code>INSTALL_O3_DOCKER=1</code> .
ORBIX3_JAVA_KEY	Specifies the Java authentication key for Orbix 3. Used only with <code>INSTALL_O3_DOCKER=1</code> .

Performing a silent installation

To perform a silent installation, specify silent mode by using the `-i` switch on the command line.

- **Windows:** `mf_ccve_corba_addon_<version>_win_x64.exe -i silent`
- **Linux:** `mf_ccve_corba_addon_<version>_lnx_x64.bin -i silent`

If the installer properties file is named `installer.properties` and is in the current directory, it will be automatically picked up. To specify a file with a different filename or in a different location, use the `-f` command line switch.

- **Windows:** `mf_ccve_corba_addon_<version>_win_x64.exe -i silent -f c:\corba\installer_win.properties`
- **Linux:** `mf_ccve_corba_addon_<version>_lnx_x64.bin -i silent -f /home/users/corba/installer_linux.properties`

Installing the SPS Client

The SPS Client is a command line interface to the Security Policy Server (SPS). The SPS Client can be used to configure the SPS or to obtain state information about the SPS.

Installing the SPS Client

Note that the SPS Client can only be installed on Linux systems.

All files are placed in the directory `/opt/microfocus/cli`.

Install the package by typing:

```
rpm -ivh /cdrom/linux/resources/Microfocus_CLI-4.0.0-<x>i386.rpm
```

If you want to install into a different directory use the `--prefix` option (not possible using RPM 4.0, for example RedHat 8.0):

```
rpm -ivh --prefix /different_directory ...
```

For more information about the installed package, such as the date of installation, the version number, etc., use the command:

```
rpm -q -i Microfocus_CLI
```

Installation Overview

The SPS Client installation directory contains the following:

Directory	Description
<code>env.sh</code>	Source this script to set the appropriate shell environment (bash and sh) for DBC commands.
<code>env.csh</code>	Source this script to set the appropriate shell environment (csh and tcsh) for DBC commands.
bin/	Contains the binaries.
<code>bin/cliconfig.sh</code>	Shell script to configure the SPS Client.
<code>bin/collectperfddata.sh</code>	Shell script for collecting performance data (see also "Performance Monitoring" in the <i>I-DBC Deployment Guide</i>).
<code>bin/dbcstat</code>	Tool to find out the status of the DBC.
<code>bin/deploydominoior.sh</code>	Shell script to deploy a domino IOR.
<code>bin/der2pem.sh</code>	Shell script to convert key and certificate files from DER to PEM encoding.
<code>bin/generateior</code>	Shell script to generate an IOR.
<code>bin/listconnections.sh</code>	A helper script to view all connections on a single DBC.
<code>bin/openssl</code>	Tool to create keys and certificates
<code>bin/printcert.sh</code>	Tool for checking the validity of certificates.
<code>bin/printior</code>	Tool for printing an IOR in a readable way.
<code>bin/proxifyior.sh</code>	Tool to proxify an IOR.
<code>bin/showciphers.sh</code>	Script to display a list of cipher suite presets offered by the DBC.

Table 1 Contents of the SPS Client installation directory

Directory	Description
bin/spsclient	The SPS Client executable.
bin/spscli.sh	Script to start the SPS Client.
bin/xtradyne.sh	Collection of common settings. This is sourced by all other scripts.
lib/	Dynamic libraries for the SPS Client.
adm/	Contains configuration information and keys.

Table 1 Contents of the SPS Client installation directory

After Installation

Configuring the SPS Client

Use the script `<installdir>/bin/cliconfig.sh` to configure the SPS Client by specifying the host and port of the Security Policy Server. The script can be given the following arguments:

```
./cliconfig.sh [-h][-b] [-i <address>] [-p <port>]
[-s yes|no] [-n <cluster>]
-h      prints a help message
-b      batch mode, do not ask for confirmation
-i <address> This is the IP address of SPS to contact. The default
address is 127.0.0.1
-p <port>  This is the port of SPS to contact. The default port is
15000.
-s yes|no  If you choose "yes" IIOP/SSL will be used to contact the
SPS. If you choose "no" plain IIOP will be used to contact the
SPS. The default is yes.
-n <cluster> name of the DBC cluster. The default name is
iDBCProxyCluster1.
```

If your SPS is for example running on a host with the IP address 192.168.47.11 with the default management port 15000, type:

```
./cliconfig.sh -i 192.168.47.11
```

Installing Keys and Certificates

If SSL is used on the management connection, you need to install the appropriate keys and certificates for the SPS Client installation:

- 1 Copy the file `<installdir>/sps/adm/AdminConsoleKeys.tar` from the SPS host to the directory `<installdir>/cli/adm` on the host where the SPS Client will be running.
- 2 On the SPS Client host change to directory `<installdir>/cli/adm` and unpack the tar file:

```
tar xvfp AdminConsoleKeys.tar
```
- 3 Create symbolic links as follows:

```
ln -sf AdminConsoleKey.der SPSCClientKey.der
ln -sf AdminConsoleCert.der SPSCClientCert.der
```
- 4 Make sure that key files are owned by user `corba`:

```
chown corba *.der
```


Docker Toolbox and IP Addresses

Introduction

Docker for Windows systems can be one of:

- Docker Desktop for Windows
- Docker Toolbox

If your Windows system does not meet the requirements for Docker Desktop for Windows, then you can use the Docker Toolbox.

A component of the Docker Toolbox is the Docker Quickstart terminal. When running this terminal, a message similar to the following may appear on the screen:

```
docker is configured to use the default machine with  
IP 192.168.99.100
```

IP address **192.168.99.100** is most likely not the IP address of your Windows system. It is an IP address used by the Oracle VM VirtualBox, which is another component of the Docker Toolbox.

When running a Docker container, the IP address of the Windows system hosting the container is passed as an environment variable:

```
--env MF_HOST_IP=192.168.99.100
```

Using the IP address displayed in the Docker Quickstart terminal (192.168.99.100) can pose challenges when running an application inside a Docker container. I-DBC inside the container will use IP address 192.168.99.100 when it proxifies an IOR.

For clients running outside the Docker container that use the proxified IOR to make invocations on the server running inside the Docker container:

- The invocation will succeed if you run the client on the *same* Windows system that is hosting the Docker container.
- The invocation will most likely fail if you run the client on a *different* system from the one hosting the Docker container. IP address 192.168.99.100 is probably not a known IP address to your network.

Using the IP Address of the Windows System

Rather than passing IP address 192.168.99.100 as an environment variable into the Docker container, we recommend that you use the IP address of the Windows system itself.

Pass the IP address of your Windows system as an environment variable when starting the Docker container, using the command:

```
--env MF_HOST_IP=<ip_addr>
```

For example:

```
--env MF_HOST_IP=10.16.16.120
```

I-DBC will then proxify IORs using the IP address passed, in this example 10.16.16.120. Client invocations using this IOR will still fail, however.

In order to be able to use the IP address of the Windows system, you can configure *port forwarding* can be configured in the Oracle VM VirtualBox.

Configure Oracle VM VirtualBox Port Forwarding

Before configuring port forwarding, compile a list of all the published ports used when running Docker containers.

The CORBA Add-on for Cloud, Containers & Virtual Environments samples publish the following ports when running Docker containers:

Port	Description
3000	Insecure port
3001	Insecure port
3002	Insecure port
8885	Secure port

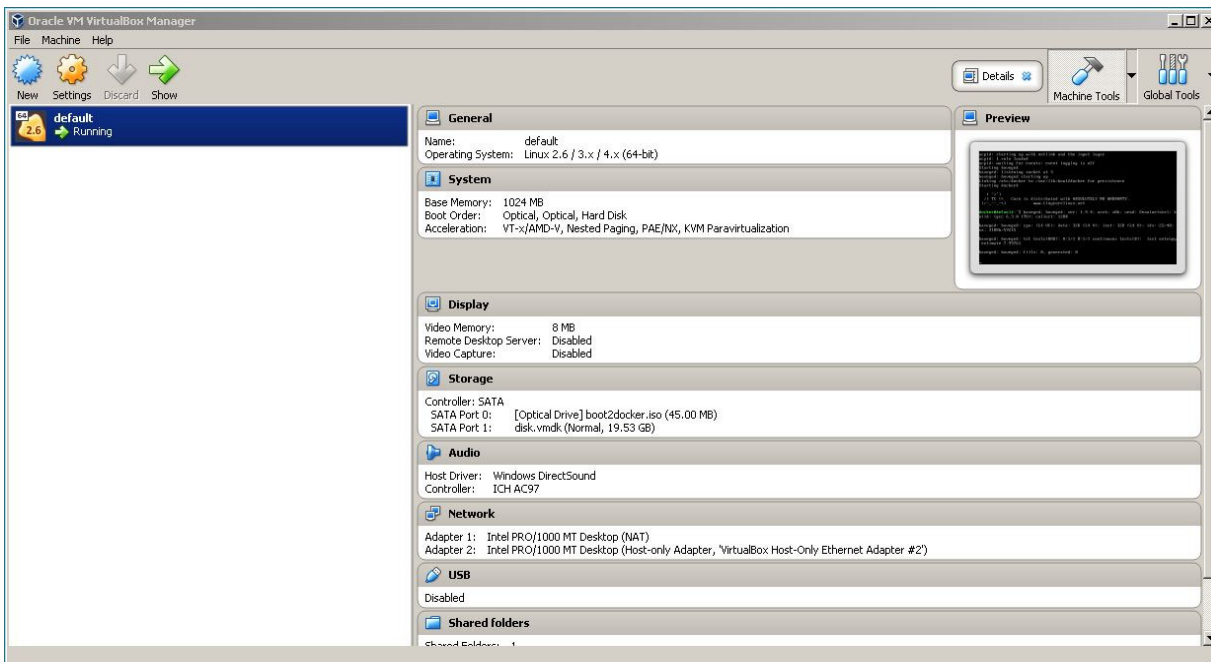
Your application may use ports in addition to or instead of these ports. Be sure to note all of them.

Once a list of all the ports that require forwarding is made, configure the Oracle VM VirtualBox.

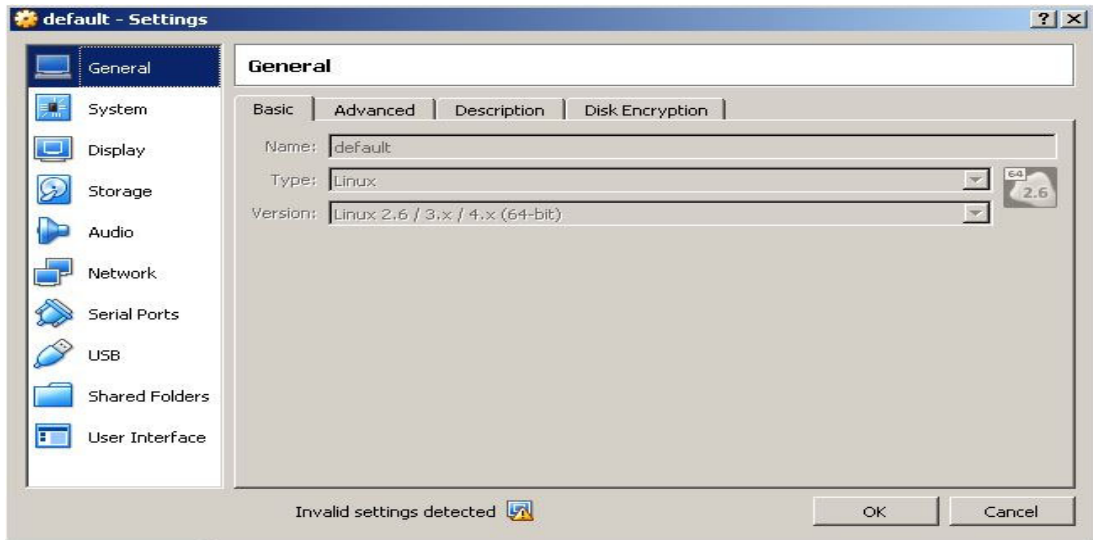
Double-click **Oracle VM VirtualBox** on your Windows system desktop



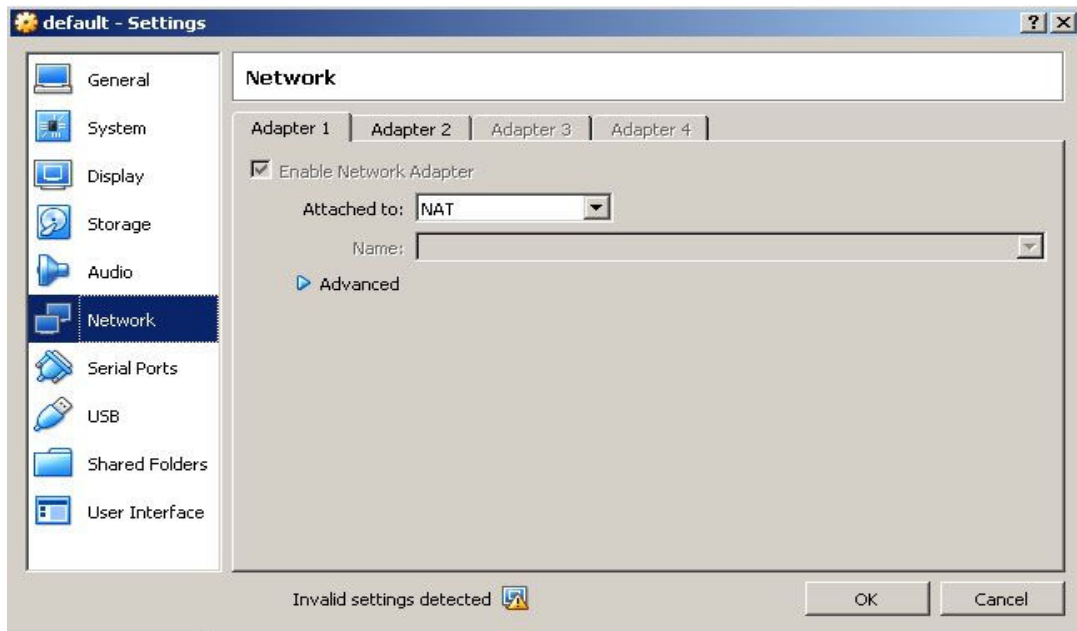
Clicking this displays the Oracle VM VirtualBox Manager window.



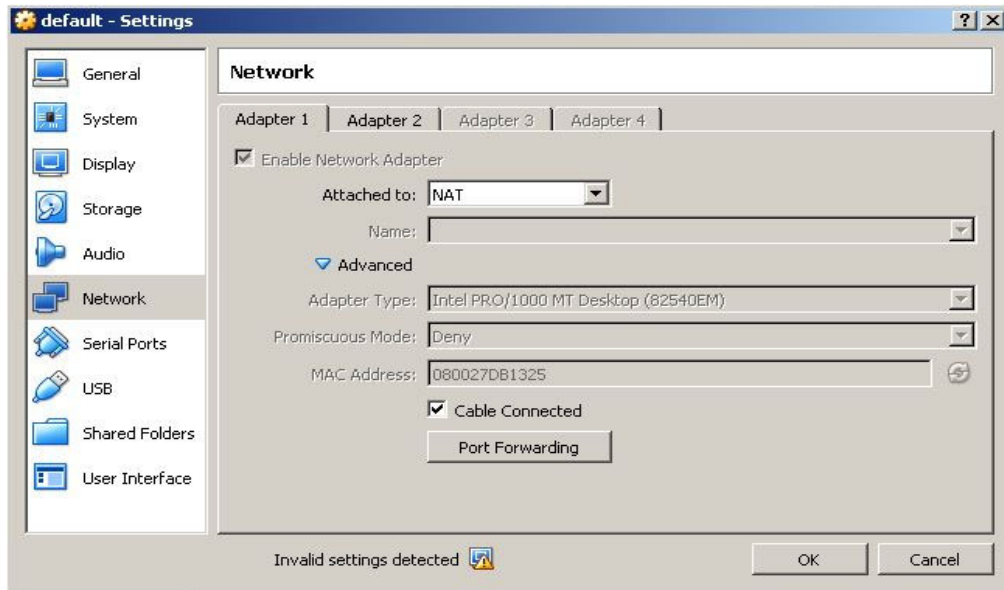
Click **Settings**. The **default-Settings** window displays.



Click **Network**.

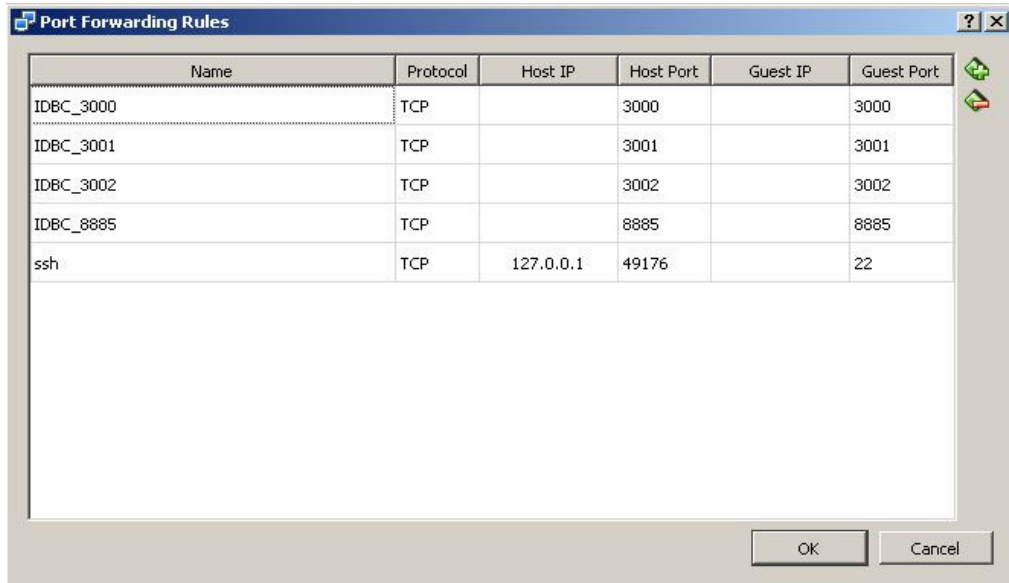


On the **Adapter1** tab, click **Advanced**.



Click **Port Forwarding**.

Use the + icon on the right-hand side of the **Port Forwarding Rules** window to add rows and build up a table similar to the one below, based on the list of ports that require port forwarding.



The **Name** column should provide a descriptive name of the port being forwarded.

If you wish to give the Administration Console access to the I-DBC Security Policy Server (SPS) running inside a Docker container on port 15000, add an entry as follows:

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
...
IDBC_15000	TCP		15000		15000

Once all possible ports that Docker containers might publish are configured into the Port Forwarding rules, the IP address of the Windows system (rather than the IP address of the Oracle VM VirtualBox) can be passed as the MF_HOST_IP environment variable when running Docker containers.

Common Docker Images

In Object Oriented Programming, a class is the "blueprint" used to create an instance of an object. Similarly in Docker, a Docker image is the blueprint used to create an instance of a Docker container.

Classes can also inherit from other classes, providing a "building block" approach to creating classes. Similarly, Docker allows images to be created from other images using the same "building block" approach.

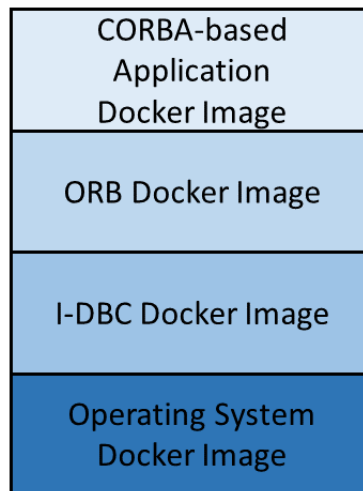
The next few chapters describe the fundamental Docker images required for using CORBA-based applications in a Docker environment. This chapter focuses on the *operating system Docker image* and the *I-DBC Docker image*.

This chapter assumes that the CORBA Add-on for Cloud, Containers & Virtual Environments product is installed at `<installdir>`. See [CORBA in Containers](#) for instructions on installing the CORBA Add-on for Cloud, Containers & Virtual Environments.

The paths and commands given here assume installation on a Linux machine. If you are using a Windows machine, adjust the paths and commands to suit Windows.

The Docker Images

Docker images are built upon each other as follows:



The foundation image is the operating system Docker image. Currently two operating systems are supported:

- CentOS
- Ubuntu

The I-DBC Docker image is built on top of the operating system Docker image. I-DBC is an application that facilitates crossing network boundaries for CORBA-based applications.

The ORB Docker image is built on top of the I-DBC Docker image. The ORB Docker image contains one of the following CORBA products:

- Orbix 3
- Orbix 6

- VisiBroker

The CORBA-based application Docker image is built on top of the ORB Docker image. It contains the CORBA based application that you wish to run inside a Docker container.

Dockerfiles

Docker images are defined by Dockerfiles. Dockerfiles consist of a set of instructions. Some common instructions are:

- **FROM:** Specifies the Docker image that this particular Dockerfile is based on.
- **RUN:** Execute a command as part of the Docker image creation.
- **COPY:** Copy a file into the Docker image.
- **ENV:** Define an environment variable.

The `docker build` command is used to create a Docker image from a Dockerfile.

The Operating System Docker Image

The operating system Docker image is the foundation upon which all other Docker images are built.

It is good practice is to make this image as lightweight as possible by creating the smallest image possible that is still capable of running your CORBA based application.

Currently two operating systems are supported:

- CentOS
- Ubuntu

The Dockerfile for CentOS

The Dockerfile for the CentOS Docker image can be found at `<installdir>/docker/common/centos_layer/Dockerfile`.

It looks as follows:

```
## Our image is based on the official CentOS Docker image
FROM centos:centos7

## We also install some essential tools as well as OpenJDK
RUN yum install -y iproute initscripts
RUN yum install -y net-tools
RUN yum install -y java-1.8.0-openjdk java-1.8.0-openjdk-devel
RUN yum group install -y "Development Tools"

ENV JAVA_HOME /usr/lib/jvm/java
```

The Dockerfile is based on the CentOS Docker image, as indicated in the `FROM` command. This is a 64-bit version of the operating system. Multiple `RUN` commands install tools are needed to run I-DBC and a CORBA ORB. One of the tools installed is an OpenJDK 8, and an environment variable is set to indicate its location.

Building the CentOS Operating System Docker Image

In a window that supports Docker commands, use the following to build the CentOS operating system Docker image. You will need access to the internet for this command to work correctly:

```
cd <installdir>/docker/common/centos_layer
docker build -t base-os-layer .
```

Note:

The '.' character is an essential part of the "docker build" command.

This Docker image is given the name `base-os-layer`. The I-DBC Docker image will build on top of this image.

The Dockerfile for Ubuntu

If you prefer, you can use Ubuntu instead of CentOS in the operating system Docker image. The Dockerfile for the Ubuntu Docker image can be found at `<installdir>/docker/common/ubuntu_layer/Dockerfile`.

It looks as follows:

```
## This image is based on the ubuntu Docker image
FROM ubuntu:18.04
ENV DEBIAN_FRONTEND=noninteractive

## We also install some essential tools as well as OpenJDK
RUN apt-get update
RUN apt-get install -y openjdk-8-jdk
RUN apt-get install -y iproute2 net-tools iputils-ping
RUN apt-get install -y gcc mono-mcs
RUN ln -s /usr/bin/make /usr/bin/gmake
RUN ln -s /usr/lib/jvm/java-1.8.0-openjdk-amd64 /usr/lib/jvm/java

ENV JAVA_HOME /usr/lib/jvm/java-1.8.0-openjdk-amd64
```

The Dockerfile is based on the Ubuntu Docker image, as indicated in the `FROM` command. This is a 64-bit version of the operating system. Multiple `RUN` commands install tools needed to run I-DBC and a CORBA ORB. One of the tools installed is an OpenJDK 8, and an environment variable is set to indicate its location.

Building the Ubuntu Operating System Docker Image

In a window that supports Docker commands, use the following to build the Ubuntu operating system Docker image. You will need access to the internet for this command to work correctly:

```
cd <installdir>/docker/common/ubuntu_layer
docker build -t base-os-layer .
```

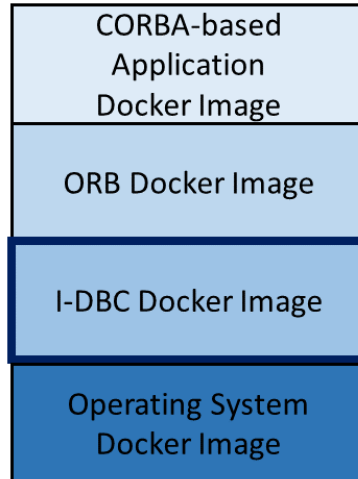
Note:

The '.' character is an essential part of the "docker build" command.

This Docker image is given the name `base-os-layer`. The I-DBC Docker image will build on top of this image.

The I-DBC Docker Image

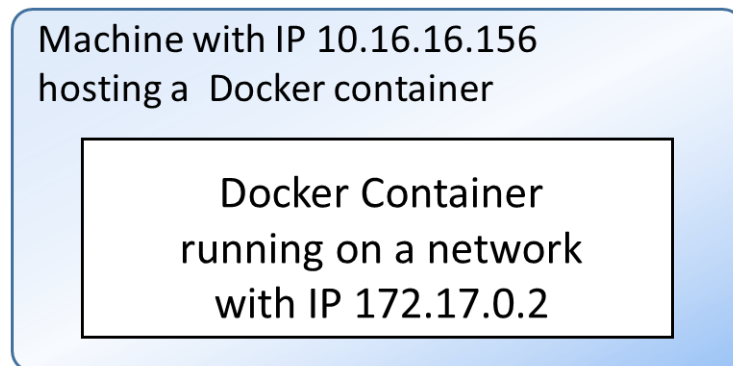
The I-DBC Docker image is built upon the operating system Docker image built according to the instructions above, with the image name `base-os_layer`.



See the ***I-DBC Administrator's Guide*** for detailed information on I-DBC.

Running CORBA-based applications inside a Docker container poses challenges where the network running inside a Docker container is often distinct from, and unknown to, the general network outside the Docker container.

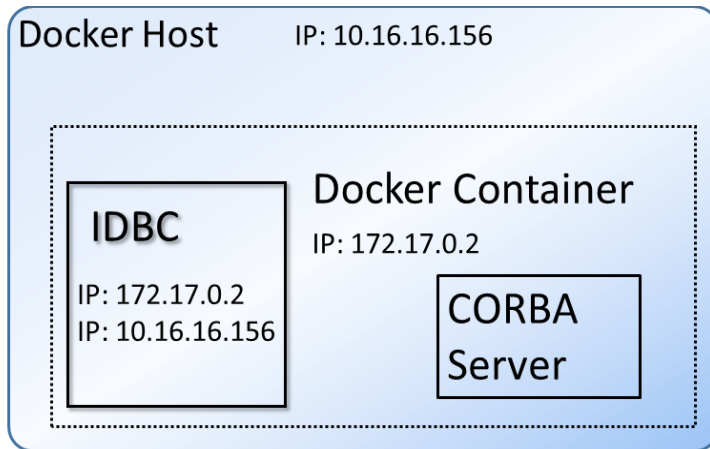
The following diagram illustrates the point, using some example IP addresses.



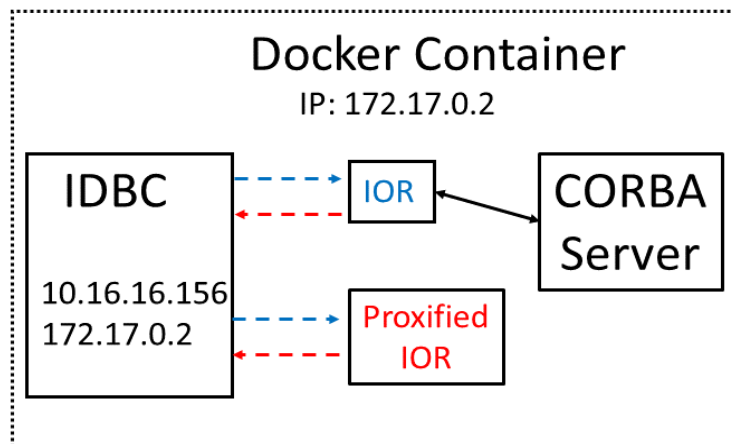
A CORBA-based application running inside the Docker container in the diagram above will publish IORs with an IP address of 172.17.0.2. A client application running outside the Docker container will likely be unable to use an IOR with an IP address of 172.17.0.2.

I-DBC can help with this issue. I-DBC will run inside the container along with the CORBA based application. I-DBC is aware of both the network hosting the Docker container (the machine with IP address 10.16.16.156)

and the network inside the Docker container (the container with IP address 172.17.0.2).



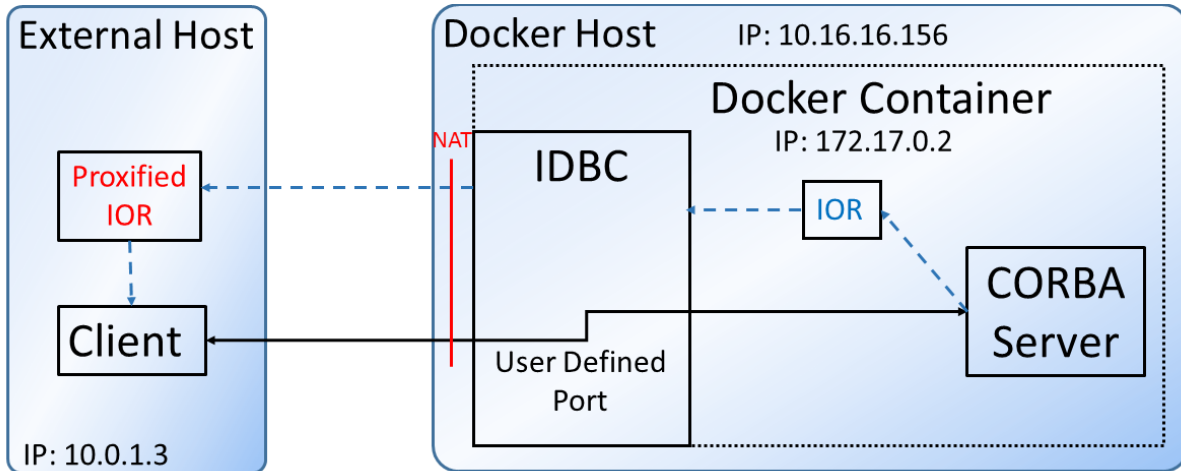
When the application publishes an IOR, I-DBC can be used to “proxify” the IOR. This means that I-DBC transforms the original IOR into a new IOR, replacing information such as the hostname (IP address 172.17.0.2 in the diagram above) and port with a hostname (IP 10.16.16.156 in the diagram above) and port of the machine hosting the Docker container.



The proxified IOR can be used successfully by clients as the IP address in the proxified IOR is a “known” address on the network.

I-DBC:

- Receives client invocations,
- Passes them along to the CORBA-based application running inside the container,
- Receive any replies from the application,
- Passes them back to the client.



The Dockerfile for I-DBC

The Dockerfile for the I-DBC Docker image can be found at:
 <install_dir>/docker/common/idbc_layer/Dockerfile

It looks as follows:

```
## Docker image is based on the previous platform image
FROM base-os-layer:latest

## Pre-Installation User Setup
RUN groupadd --gid 1024 corbagroup
RUN useradd -ms /bin/bash -g corbagroup corba

ENV DBC_USER corba
ENV PRODUCT_HOME /home/corba/microfocus/idbc
ENV IDBC_LIC_DIR /home/corba/microfocus/idbc/idbc/var
ENV IDBC_LIC_DEFAULT_DIR /home/corba/microfocus/idbc/idbc/license
ENV PATH $PATH:$JAVA_HOME/bin

RUN mkdir -p $PRODUCT_HOME
RUN mkdir -p /home/corba/volume
RUN chown -R corba:corbagroup /home/corba
USER corba

ENV PRODUCT_USER corba
ENV COUNTRY uk
ENV COMPANY microfocus
COPY entrypoint_common.sh /home/corba/entrypoint_common.sh

## Install the IDBC/SPS and run its setup scripts
COPY microfocus_SPS-4.0.0.tar.gz $PRODUCT_HOME/microfocus_SPS-4.0.0.tar.gz
COPY microfocus_CLI-4.0.0.tar.gz $PRODUCT_HOME/microfocus_CLI-4.0.0.tar.gz
COPY microfocus_IDBC-4.0.0.tar.gz $PRODUCT_HOME/microfocus_IDBC-4.0.0.tar.gz
COPY mf_idbc_install.sh $PRODUCT_HOME/mf_idbc_install.sh
WORKDIR $PRODUCT_HOME/
RUN /bin/bash $PRODUCT_HOME/mf_idbc_install.sh install
RUN /bin/bash $PRODUCT_HOME/mf_idbc_install.sh configure

ENV IDBC_PRODUCT_DIR $PRODUCT_HOME
```


The Dockerfile is based on the `base-os-layer:latest` Docker image, as indicated by the FROM command. This was the operating system Docker image created in “[The Operating System Docker Image](#)”.

There are several important things to note.

User ID

The RUN command is used to create a group and user ID as follows:

- Group: `corbagroup`
- User ID: `corba`

The ENV command sets the `DCB_USER` variable to “`corba`”, and the `PRODUCT_HOME` variable to the I-DBC installation directory.

The RUN command is used to create the I-DBC installation directory, and to recursively change ownership of user **corba**'s home directory to user **corba**.

The USER command sets the user to **corba**.

All further Docker commands, and all applications run inside Docker containers, will now be run as user **corba**, assuming that no Dockerfiles built upon the Dockerfile for I-DBC specify a different user with the USER command.

I-DBC Environment Variables

The ENV command sets these variables:

- `PRODUCT_USER` to **corba**, indicating that I-DBC is to run as user **corba**.
- `COUNTRY` to **UK**, which will be used when generating certificates.
- `COMPANY` to **microfocus**, which will be used when generating certificates.

Common Entrypoint Helper Script

The COPY command is used to copy the file `entrypoint_common.sh` to the directory `/home/corba/entrypoint_common.sh`.

This file contains helper functions to do things such as:

- Set configuration items for I-DBC
- Start I-IDBC
- Check the running status of I-DBC
- Proxy an IOR
- Start a CORBA server application

These functions are used in the various examples found in:

```
<installdir>/docker/orbix3/application_layer  
<installdir>/docker/orbix6/application_layer  
<installdir>/docker/visibroker/application_layer
```

The entrypoint scripts for the various examples illustrate how to make use of these functions.

When creating your own CORBA based application running inside of a Docker container, you can:

- Use the functions in `entrypoint_common.sh` as illustrated in the various example entrypoint scripts, if they satisfy the requirements of your application.
- Add or update functions in `entrypoint_common.sh` to suit the requirements of your application.

- Provide your own mechanism to do the equivalent of what `entrypoint_common.sh` does.

Install I-DBC

The `COPY` command is used to copy the I-DBC installation files. The installer files are:

- `microfocus_SPS-4.0.0.tar.gz`: Installs a 64-bit version of the Security Policy Server
- `microfocus_CLI_4.0.0.tar.gz`: Installs a 64-bit version of the Security Policy Server client
- `microfocus_IDBC-4.0.0.tar.gz`: Installs a 64-bit version of the I-DBC proxy
- `mf_idbc_install.sh`: Install and configuration script
- `idbc_license.slip`: A license needed to run I-DBC

The `RUN` command is used to run the `mf_idbc_install.sh` script to install and configure I-DBC.

The `ENV` command sets the `DCB_USER` variable to "corba", and the `PRODUCT_HOME` variable to the I-DBC installation directory.

Building the I-DBC Docker Image

In a window that supports Docker commands, use the following to build the I-DBC Docker image:

```
cd <installdir>/docker/common/idbc_layer
docker build -t idbc-layer .
```

Note:

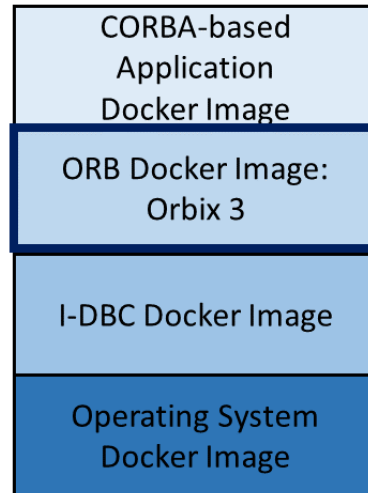
The `'.'` character is an essential part of the `docker build` command.

This Docker image is given the name `idbc-layer`. The various ORB Docker images will build on top of this image.

The Orbix 3 Docker Image

This chapter describes the Orbix 3 ORB Docker image.

The Orbix 3 Docker Image



The Orbix 3 Docker image is built upon the I-DBC Docker image, which is in turn built upon the operating system Docker image.

This chapter assumes that the CORBA Add-on for Cloud, Containers & Virtual Environments product is installed at `<installdir>`.

Paths and commands given in this chapter assume installation on a Linux machine. If using a Windows machine, adjust the paths and commands to suit Windows.

The Dockerfile for Orbix 3

The Dockerfile for the Orbix 3 Docker image can be found at `<installdir>/docker/orbix3/orb_base_layer/Dockerfile`.

It looks as follows:

```
FROM idbc-layer

## Pre-Installation User Setup
USER corba
run mkdir -p /home/corba/microfocus/orbix/orbix33
ENV CORBA_PRODUCT_DIR /home/corba/microfocus/orbix/orbix33

## Copy the Orbix 3 installer files
COPY microfocus_orbix3_lnx.bin ${CORBA_PRODUCT_DIR}/
microfocus_orbix3_lnx.bin
COPY microfocus_orbix3_ssl_lnx.bin ${CORBA_PRODUCT_DIR}/
microfocus_orbix3_ssl_lnx.bin
COPY installer.properties ${CORBA_PRODUCT_DIR}/
installer.properties

## Install Orbix 3
RUN ${CORBA_PRODUCT_DIR}/microfocus_orbix3_lnx.bin -f
${CORBA_PRODUCT_DIR}/installer.properties
RUN ${CORBA_PRODUCT_DIR}/microfocus_orbix3_ssl_lnx.bin -f
${CORBA_PRODUCT_DIR}/installer.properties
```

```

## Install hotfixes on top of the installation if there are any
COPY hotfixes /home/corba/hotfixes
COPY hotfix_install_helper.sh /home/corba/
hotfix_install_helper.sh
RUN /home/corba/hotfix_install_helper.sh

## Add the entrypoint helper script
COPY entrypoint_helper_o3.sh /home/corba/

## Create location for proxified IOR's to go
RUN mkdir -p /home/corba/proxified_iors

## Ready to work
WORKDIR ${CORBA_PRODUCT_DIR}

```

The Dockerfile is based on the `idbc-layer` Docker image, as indicated by the `FROM` instruction. This image was the I-DBC Docker image created in the steps outlined in the “[Common Docker Images](#)” chapter.

User ID

Note that the `USER` instruction indicates that all further Docker commands, and all applications that are run inside Docker containers created from this image, will now be run as user **corba** (unless any Dockerfiles built upon the Dockerfile for Orbix 3 specify a different user with the `USER` instruction).

Installing Orbix 3

The Dockerfile creates the directory path `/home/corba/microfcous/orbix/orbix33`, which is where Orbix 3 will be installed.

The following files are copied:

- `microfocus_orbix3_lnx.bin`: The Orbix 3 installer for Linux
- `microfocus_orbix3_ssl_lnx.bin`: The Orbix 3 SSL installer for Linux
- `installer.properties`: The silent installer file

The `RUN` instruction runs the Orbix 6 installer, referencing the silent installer file.

Orbix 3 Entrypoint Helper Script

The `COPY` instruction is used to copy the file `entrypoint_helper_o3.sh` to the directory `/home/corba`.

The file `entrypoint_helper_o3.sh` includes methods that are found in the separate file `entrypoint_common.sh`, which was installed as part of the I-DBC Docker image.

This file contains helper function `run_securely`, which is used to indicate that Orbix 3 is running in secure mode.

The functions in `entrypoint_helper_o3.sh` and `entrypoint_common.sh` are used in the various demos found in:

```
<installdir>/docker/orbix3/application_layer
```

The entrypoint scripts for the various demos illustrate how to make use of these functions.

When creating your own CORBA based application running inside a Docker container, you can:

- Use the functions in `entrypoint_helper_o3.sh` and `entrypoint_common.sh` as illustrated in the various example `entrypoint` scripts, if they satisfy the requirements of your application.
- Add or update functions in `entrypoint_helper_o3.sh` and `entrypoint_common.sh` to suit the requirements of your application.
- Provide your own mechanism to do the equivalent of what `entrypoint_helper_o6.sh` and `entrypoint_common.sh` do.

Proxified IOR Location

The `RUN` instruction makes a directory called `/home/corba/proxified_ior`s. When using functions from `entrypoint_helper_o6.sh` and `entrypoint_common.sh` to proxify IORs, they will be written to this directory.

Building the Orbix 3 Docker Image

In a window that supports Docker commands, use the following to build the Orbix 3 Docker image:

```
cd <installdir>/docker/orbix3/orb_base_layer
docker build -t orbix3-idbc-layer .
```

Note:

The `'.'` character is an essential part of the `"docker build"` command.

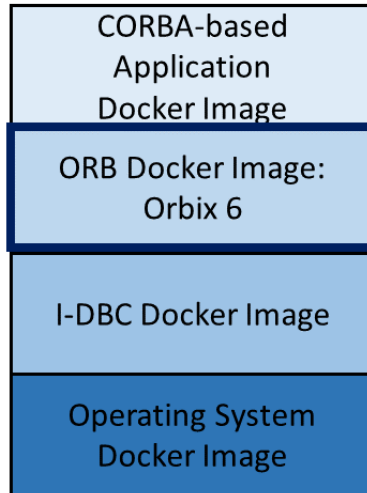
The Docker image is given the name `orbix3-idbc-layer`. All the Orbix 3 demo Docker images in `<installdir>/docker/orbix3/application_layer` are built on top of this image.

As illustrated in the demos, your CORBA-based application would create a Docker image based on the `orbix3-idbc-layer` image as well, following a pattern similar to what is done for each demo.

The Orbix 6 Docker Image

This chapter describes the Orbix 6 ORB Docker image.

The Orbix 6 Docker Image



The Orbix 6 Docker image is built upon the I-DBC Docker image, which is in turn built upon the operating system Docker image.

This chapter assumes that the CORBA Add-on for Cloud, Containers & Virtual Environments product is installed at `<installdir>`.

Paths and commands given in this chapter assume installation on a Linux machine. If using a Windows machine, adjust the paths and commands to suit Windows.

The Dockerfile for Orbix 6

The Dockerfile for the Orbix 6 Docker image can be found at `<installdir>/docker/orbix6/orb_base_layer/Dockerfile`.

It looks as follows:

```
FROM idbc-layer

## Pre-Installation User Setup
USER corba
RUN mkdir -p /home/corba/microfocus/orbix/etc/bin
ENV IT_PRODUCT_DIR /home/corba/microfocus/orbix

## Install Orbix 6
COPY microfocus_orbix6_lnx.bin ${IT_PRODUCT_DIR}/
  microfocus_orbix6_lnx.bin
COPY installer.properties ${IT_PRODUCT_DIR}/installer.properties
RUN ${IT_PRODUCT_DIR}/microfocus_orbix6_lnx.bin -i silent -f
  ${IT_PRODUCT_DIR}/installer.properties

## Install hotfixes on top of the installation if there are any
COPY hotfixes /home/corba/hotfixes
COPY hotfix_install_helper.sh /home/corba/hotfix_install_helper.sh
RUN /home/corba/hotfix_install_helper.sh
```

```

## Copy the Orbix 6 file with idl License points into the install
  etc directory so we can build demos
COPY licenses_idl.txt ${IT_PRODUCT_DIR}/etc/licenses.txt

ENV IT_LICENSE_FILE ${IT_PRODUCT_DIR}/etc/licenses.txt

## Add the entrypoint_helper script
COPY entrypoint_helper_o6.sh /home/corba/

## Create location for proxified IOR's to go
RUN mkdir /home/corba/proxified_ior

# Allows for Java builds using "itant"
#
ENV IT_DOMAIN_NAME orbix6_domain
COPY build_env.sh /home/corba/

## Ready to work
WORKDIR ${IT_PRODUCT_DIR}

```

The Dockerfile is based on the `idbc-layer` Docker image, as indicated by the `FROM` instruction. This image was the I-DBC Docker image created in the steps outlined in the “[Common Docker Images](#)” chapter.

User ID

Note that the `USER` instruction indicates that all further Docker commands, and all applications that are run inside Docker containers created from this image, will now be run as user **corba** (unless any Dockerfiles built upon the Dockerfile for Orbix 6 specify a different user with the `USER` instruction).

Installing Orbix 6

The Dockerfile creates a directory path and sets the `IT_PRODUCT_DIR` environment variable to `/home/corba/microfocus/orbix`, which is where Orbix 6 will be installed.

The following files are copied:

- `microfocus_orbix6_lnx.bin`: The Orbix 6 installer for Linux
- `installer.properties`: The silent installer file

The `RUN` instruction runs the Orbix 6 installer, referencing the silent installer file.

An Orbix 6 license file is copied, and the `IT_LICENSE_FILE` environment variable is set to point to its location.

Orbix 6 Entrypoint Helper Script

The `COPY` instruction is used to copy the file `entrypoint_helper_o6.sh` to the directory `/home/corba`.

The file `entrypoint_helper_o6.sh` includes methods that are found in the separate file `entrypoint_common.sh`, which was installed as part of the I-DBC Docker image.

This file contains helper functions to do things such as:

- Get an IOR from an Orbix 6 configuration file
- Proxy an IOR from an Orbix 6 configuration file
- Run an Orbix 6 deployment
- Start Orbix 6 services

These functions are used in the various demos found in:


```
<installdir>/docker/orbix6/application_layer
```

The entrypoint scripts for the various demos illustrate how to make use of these functions.

When creating your own CORBA based application running inside a Docker container, you can:

- Use the functions in `entrypoint_helper_o6.sh` as illustrated in the various example entrypoint scripts, if they satisfy the requirements of your application.
- Add or update functions in `entrypoint_helper_o6.sh` to suit the requirements of your application.
- Provide your own mechanism to do the equivalent of what `entrypoint_helper_o6.sh` does.

Proxified IOR Location

The `RUN` instruction makes a directory called `/home/corba/proxified_ior`s. When using functions from `entrypoint_helper_o6.sh` and `entrypoint_common.sh` to proxify IORs, they will be written to this directory.

Orbix 6 Domain Name

The `ENV` instruction is used to set the `IT_DOMAIN_NAME` environment variable to the value `orbix6_domain`.

When building and running the Orbix 6 demos in `<installdir>/docker/orbix6/application_layer`, the domain name will be assumed to be **orbix6_domain**.

Build Script

The file `build_env.sh` is copied into the directory `/home/corba`. This script allows for building Orbix 6-based applications before a deployment is performed.

Building the Orbix 6 Docker Image

In a window that supports Docker commands, use the following to build the Orbix 6 Docker image:

```
cd <installdir>/docker/orbix6/orb_base_layer
docker build -t o6-idbc-layer .
```

Note:

The `'.'` character is an essential part of the `"docker build"` command.

The Docker image is given the name `o6-idbc-layer`. All the Orbix 6 demo Docker images in `-3.2-<x>/docker/orbix6/application_layer` are built on top of this image.

As illustrated in the demos, your CORBA-based application would create a Docker image based on the `o6-idbc-layer` image as well, following a pattern similar to what is done for each demo.

Creating Orbix 6 Deployment Descriptors

This chapter describes how to create deployment descriptors for Orbix 6-based applications.

Introduction

When developing applications based on Orbix 6, one of the development steps is to perform an Orbix 6 deployment. The items generated by the deployment process include:

- A configuration file. It can be file based, or a Configuration Repository (CFR).
- Log files for deployed services.
- Database files for the IMR, and potentially other Orbix 6 services.
- A deployment descriptor.

See the **Orbix 6 Deployment Guide** for further information on deployment.

Orbix 6-based applications running inside a Docker container will require deployment. Deployments inside Docker containers will use a deployment descriptor. This chapter explains how to create this deployment descriptor by using the supplied Orbix 6 basic log demo as an example.

This chapter assumes that:

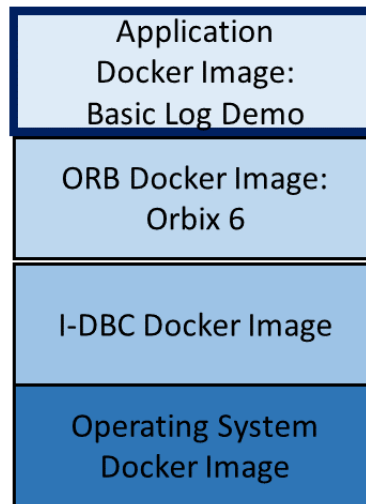
- Orbix 6 is installed outside of Docker at `<orbix6dir>`.
- The CORBA Add-on for Cloud, Containers & Virtual Environments is installed at `<installdir>`.
- Paths and commands given in this chapter assume installation on a Linux machine. If using a Windows machine, adjust the paths and commands to suit Windows.

Note that the basic log demo exists in two places:

- The demo supplied with Orbix 6 at `<orbix6dir>/asp/6.3/demos/corba/enterprise/basic_log`. The client part of the demo is run outside Docker from here.
- The demo supplied with the CORBA Add-on for Cloud, Containers & Virtual Environments at `<installdir>/docker/orbix6/application_layer/basic_log_demo_cxx`. The server part of the demo is run inside Docker from here.

The Basic Log Demo

This demo creates a Docker image built upon the `o6-idbc-layer` Docker image.



The Dockerfile for the demo looks as follows:

```
# Dockerfile for the basic_log demo
#
# Use o6-idbc-layer as the base layer for this image.
# To create the Docker image for this demo run:
#
# docker build -t orbix6_application:latest .
#
FROM o6-idbc-layer

WORKDIR ${IT_PRODUCT_DIR}
ENV TEST_DIR ${IT_PRODUCT_DIR}/asp/6.3/demos/corba/enterprise/
basic_log/simple/cxx_demo

# Build the basic_log demo inside the image
#
RUN . /home/corba/build_env.sh && cd ${TEST_DIR} && make -e

# Add the deployment descriptor file to the image
#
COPY ${IT_DOMAIN_NAME}_dd.xml ${IT_PRODUCT_DIR}/etc/
${IT_DOMAIN_NAME}_dd.xml

# Add the entrypoint.sh file to the image
#
COPY entrypoint.sh ${TEST_DIR}

# When a Docker container is started using this image, run this
script
#
ENTRYPOINT ${TEST_DIR}/entrypoint.sh
```

When creating a Docker image from this Dockerfile:

- The Orbix 6 basic log xx demo is built.
- Files `orbix6_domain_dd.xml` and `entrypoint.sh` are copied.
- The `ENTRYPOINT` instruction indicates that the `entrypoint.sh` script is to be run when a Docker container is started using this image.

File `<installdir>/docker/orbix6/application_layer/basic_log_demo_cxx/README_CXX.txt` has instructions on how to run this demo.

Deployment inside a Docker Container

The file `entrypoint.sh` calls the function `deploy_orbix6_if_needed`, which is part of `entrypoint_helper_o6.sh`. This function will perform a deployment if no prior deployment is detected.

It runs the Orbix `itconfigure` tool in command line mode and takes deployment descriptor `orbix6_domain_dd.xml` as input. The file `orbix6_domain_dd.xml` looks as follows:

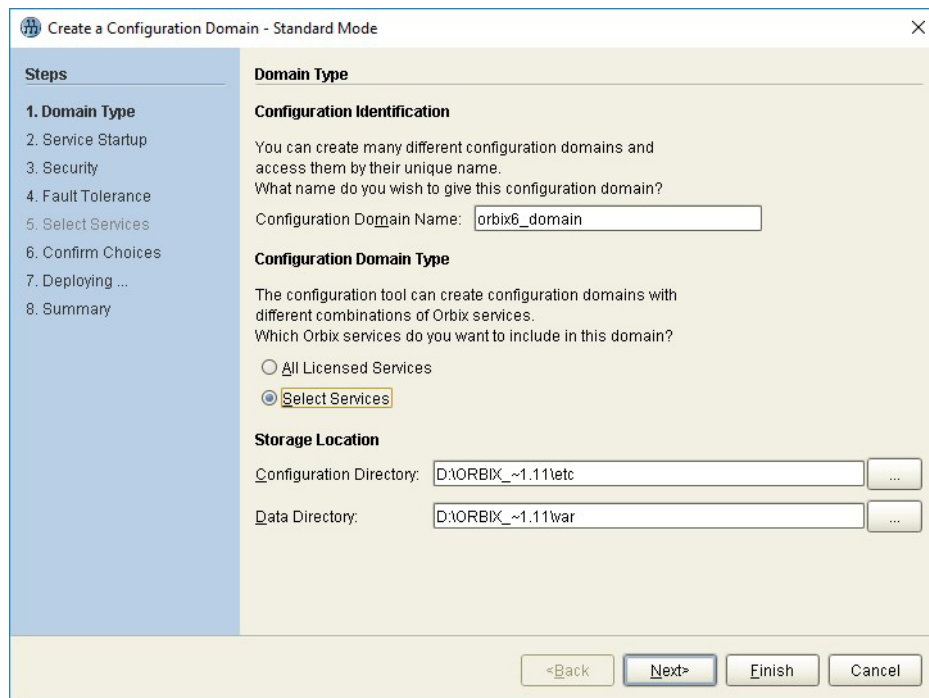
```
<?xml version="1.0" encoding="UTF-8"?>
<dd:descriptor xmlns:dd="http://ns.iona.com/orbix/schema/dd/1.2">
  <!--This deployment descriptor version 1.2.0 has been generated
  by Orbix tools-->
  <dd:configuration>
    <dd:domain>orbix6_domain</dd:domain>
    <dd:source>file</dd:source>
    <dd:location_domain>all_services.location</
dd:location_domain>
  </dd:configuration>
  <!--Concrete node information for this deployment-->
  <dd:nodes>
    <dd:node name="node1" ip="1.1.1.1" profile="node1">
      <dd:policies>
        <dd:policy name="prefer_ipv4" value="true" />
      </dd:policies>
    </dd:node>
  </dd:nodes>
  <!--The following profiles will be deployed-->
  <dd:profile id="node1">
    <dd:service name="locator" link="false">
      <dd:activation mode="manual" />
      <dd:run mode="direct_persistent" instrumented="false"
proxified="false" managed="false" authenticated="false"
perflog="false" dynlog="false" />
      <dd:endpoint protocol="iiop" port="3075" />
    </dd:service>
    <dd:service name="node_daemon" link="false">
      <dd:activation mode="manual" />
      <dd:run mode="direct_persistent" instrumented="false"
proxified="false" managed="false" authenticated="false"
perflog="false" dynlog="false" />
      <dd:endpoint protocol="iiop" port="53079" />
    </dd:service>
    <dd:service name="basic_log" link="false">
      <dd:activation mode="manual" />
      <dd:run mode="direct_persistent" instrumented="false"
proxified="false" managed="false" authenticated="false"
perflog="false" dynlog="false" />
      <dd:endpoint protocol="iiop" port="53093" />
    </dd:service>
    <dd:component name="demos">
      <dd:endpoint protocol="iiop" />
    </dd:component>
  </dd:profile>
</dd:descriptor>
```

Creating the Deployment Descriptor

Whether running the basic log demo, or your own Orbix 6 based application inside a Docker container, you will need to create an `orbix6_domain_dd.xml` file to be used when deploying.

Use the Orbix 6 `itconfigure` tool in GUI mode to create the deployment descriptor file. The descriptor file used in the basic log demo was created as follows:

- 1 Open a window and set the `JAVA_HOME` environment variable to a JDK that can run the Orbix 6 `itconfigure` tool.
- 2 Start the `itconfigure` tool.
- 3 Use standard mode to create a new domain named `orbix6_domain`. It is important to use this name as the deployment functions in `entrypoint_helper_o6.sh` are coded to use a deployment descriptor named `orbix6_domain_dd.xml`. In the **Configuration Domain Type** section, choose the **Select Services** button.



- 4 Navigate through the **Service Startup** and **Security** windows. Choose **Insecure Communication**.
- 5 Navigate through the **Fault Tolerance** window to the **Select Services** window and check the following checkboxes:
 - Location
 - Node Daemon
 - Basic Logging
 - Demos
- 6 Navigate to the **Confirmation** and **Deploying** window.
- 7 Once deployment is complete, the **Summary** window appears. Press **Finish** to end the deployment process. You can close the **Orbix Configuration** window.

The deployment descriptor file is written to

`<orbix6dir>/etc/domains/orbix6_domain/orbix6_domain_dd.xml`.

It will look something like:

```
<?xml version="1.0" encoding="UTF-8"?>
<dd:descriptor xmlns:dd="http://ns.iona.com/orbix/schema/dd/1.2">
  <!--This deployment descriptor version 1.2.0 has been generated by Orbix tools-->
  <dd:configuration>
    <dd:domain>orbix6_domain</dd:domain>
    <dd:source>file</dd:source>
    <dd:location_domain>orbix6_domain.location</dd:location_domain>
  </dd:configuration>
  <!--Concrete node information for this deployment-->
  <dd:nodes>
    <dd:node name="MY-SYSTEM" ip="10.16.16.188" profile="MY-SYSTEM">
      <dd:policies>
        <dd:policy name="prefer_ipv4" value="true" />
      </dd:policies>
    </dd:node>
  </dd:nodes>
  <!--The following profiles will be deployed-->
  <dd:profile id="MY-SYSTEM">
    <dd:service name="locator" link="false">
      <dd:activation mode="manual" />
      <dd:run mode="direct_persistent" instrumented="false" proxified="false"
managed="false" authenticated="false" perflong="false" dynlog="false" />
      <dd:endpoint protocol="iiop" port="3075" />
    </dd:service>
    <dd:service name="node_daemon" link="false">
      <dd:activation mode="manual" />
      <dd:run mode="direct_persistent" instrumented="false" proxified="false"
managed="false" authenticated="false" perflong="false" dynlog="false" />
      <dd:endpoint protocol="iiop" port="53079" />
    </dd:service>
    <dd:service name="basic_log" link="false">
      <dd:activation mode="manual" />
      <dd:run mode="direct_persistent" instrumented="false" proxified="false"
managed="false" authenticated="false" perflong="false" dynlog="false" />
      <dd:endpoint protocol="iiop" port="53093" />
    </dd:service>
    <dd:component name="demos">
      <dd:endpoint protocol="iiop" />
    </dd:component>
  </dd:profile>
</dd:descriptor>
```

Modifying the Deployment Descriptor for use with Docker

Two lines in the generated deployment descriptor above are highlighted:

```
<dd:node name="MY-SYSTEM" ip="10.16.16.188" profile="MY-SYSTEM">
...
<dd:profile id="MY-SYSTEM">
```

These lines have information that reflects the machine where the deployment was run:

- The host name of the machine: **MY-SYSTEM**
- The IP address of the machine: **10.16.16.188**

Since these values will have no meaning when the deployment descriptor is used inside a Docker container, where the host name and IP address will almost certainly be different, you can change the two lines in the file to more general values as follows:

```
<dd:node name="node1" ip="1.1.1.1" profile="node1">
...
<dd:profile id="node1">
```

When deploying inside a Docker container, the values `node1` and `1.1.1.1` will be replaced with the Docker container's actual host name and IP address.

Creating a Deployment Descriptor for your Orbix 6-based Application

The instructions above illustrate how the deployment descriptor for the basic log demo was created. When considering your particular application, you will need to consider several questions including:

- Whether to run insecurely or securely
- Whether fault tolerance is required
- The set of Orbix 6 services your application needs

Once you have determined these, run the `itconfigure` tool as described above, selecting the security, fault tolerance, and services as required by your application. Be sure to use the name `orbix6_domain` as the domain name if you are using the functions supplied in `entrypoint_helper_o6.sh` to deploy inside Docker.

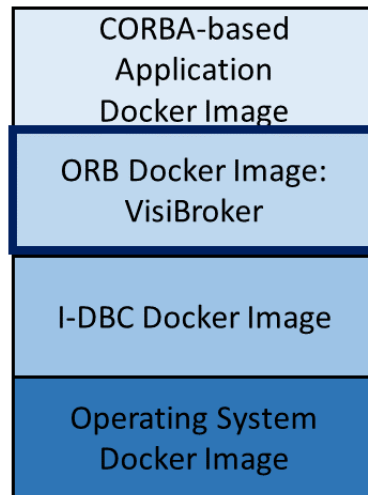
In the Dockerfile for your application, copy `orbix6_domain_dd.xml` into the Docker container, to `/home/corba/microfocus/orbix/orbix6_domain_dd.xml`.

Use the `deploy_orbix6_if_needed` function from `entrypoint_helper_o6.sh` in the entrypoint script for your application to perform the deployment.

The VisiBroker Docker Image

This chapter describes the VisiBroker ORB Docker image.

The VisiBroker Docker Image



The VisiBroker Docker image is built upon the I-DBC Docker image, which is in turn built upon the operating system Docker image.

This chapter assumes that the CORBA Add-on for Cloud, Containers & Virtual Environments product is installed at `<installdir>`.

Paths and commands given in this chapter assume installation on a Linux machine. If using a Windows machine, adjust the paths and commands to suit Windows.

The Dockerfile for VisiBroker

The Dockerfile for the VisiBroker Docker image can be found at `<installdir>/docker/visibroker/orb_base_layer/Dockerfile`.

It looks as follows:

```
## Based on our idbc layer
FROM idbc-layer

## Pre-Installation User Setup
USER corba
RUN mkdir -p /home/corba/microfocus/Visibroker
ENV VB_INST_DIR /home/corba/microfocus/Visibroker

## Install Visibroker & configure
COPY microfocus_visibroker_lnx.bin /home/corba/microfocus_visibroker_lnx.bin
COPY silentproperties.txt /home/corba/silentproperties.txt
RUN /home/corba/microfocus_visibroker_lnx.bin -f /home/corba/silentproperties.txt

## Install hotfixes on top of the installation if there are any
COPY hotfixes /home/corba/hotfixes
```

```

COPY hotfix_install_helper.sh /home/corba/
hotfix_install_helper.sh
RUN /home/corba/hotfix_install_helper.sh

## Add the entrypoint_helper script
COPY entrypoint_helper_vb.sh /home/corba/

## Create location for proxified IOR's to go
RUN mkdir /home/corba/proxified_ior

## Ready to work
WORKDIR $VB_INST_DIR

```

The Dockerfile is based on the `idbc-layer` Docker image, as indicated by the `FROM` instruction. This image was the I-DBC Docker image created in the steps outlined in the “[Common Docker Images](#)” chapter.

User ID

Note that the `USER` instruction indicates that all further Docker commands, and all applications that are run inside Docker containers created from this image, will now be run as user **corba** (unless any Dockerfiles built upon the Dockerfile for VisiBroker specify a different user with the `USER` instruction).

Installing VisiBroker

The Dockerfile creates the directory path `/home/corba/microfocus/visibroker`, which is where VisiBroker will be installed. The `ENV` instruction sets the environment variable `VB_INST_DIR` to point to this directory.

The following files are copied:

- `microfocus_visibroker_lnx.bin`: The VisiBroker installer for Linux
- `silentproperties.txt`: The silent installer file

The `RUN` instruction runs the VisiBroker installer, referencing the silent installer file.

The license file `license.slip` is copied to the VisiBroker installation license directory.

Install HotFixes

If any HotFixes are available, they are applied by copying:

- The HotFixes directory
- The `hotfix_installer_helpers.sh` script

The `hotfix_installer_helpers.sh` script will install any HotFixes it finds in the HotFixes directory.

VisiBroker Entrypoint Helper Script

The `COPY` instruction is used to copy the file `entrypoint_helper_vb.sh` to the directory `/home/corba`.

The file `entrypoint_helper_vb.sh` includes methods that are found in the separate file `entrypoint_common.sh`, which was installed as part of the I-DBC Docker image.

This file contains helper functions to do things such as:

- Start a VisiBroker feature
- Start a VisiBroker feature and wait for an IOR to be created

- Deploy a VisiBroker service.

The functions in `entrypoint_helper_vb.sh` and `entrypoint_common.sh` are used in the various examples found in:

```
<installdir>/docker/visibroker/application_layer
```

The entrypoint scripts for the various examples illustrate how to make use of these functions.

When creating your own CORBA based application running inside a Docker container, you can:

- Use the functions in `entrypoint_helper_vb.sh` and `entrypoint_common.sh` as illustrated in the various example entrypoint scripts, if they satisfy the requirements of your application.
- Add or update functions in `entrypoint_helper_vb.sh` and `entrypoint_common.sh` to suit the requirements of your application.
- Provide your own mechanism to do the equivalent of what `entrypoint_helper_vb.sh` and `entrypoint_common.sh` do.

Proxified IOR Location

The `RUN` instruction makes a directory called `/home/corba/proxified_iors`. When using functions from `entrypoint_common.sh` to proxify IORS, they will be written to this directory.

Building the VisiBroker Docker Image

In a window that supports Docker commands, use the following to build the VisiBroker Docker image:

```
cd <installdir>/docker/visibroker/orb_base_layer
docker build -t vb-idbc-layer .
```

Note:

The `'.'` character is an essential part of the `"docker build"` command.

The Docker image is given the name `vb-idbc-layer`. All the VisiBroker example Docker images in `<installdir>/docker/visibroker/application_layer` are built on top of this image.

As illustrated in the examples, your CORBA-based application would create a Docker image based on the `vb-idbc-layer` image as well, following a pattern similar to what is done for each example.

The VisiBroker Smart Agent Relay

This chapter describes the VisiBroker Smart Agent Relay.

VisiBroker's Smart Agent (osagent) is a dynamic, distributed directory service. It allows client programs to locate object implementations, enabling the client to connect to those implementations and invoke their behavior. The Smart Agent is proprietary to VisiBroker, and so does not operate with other ORB implementations.

For detailed information on what the Smart Agent is and how to use it, see the VisiBroker (for C++ or Java) **Developer's Guide** chapter entitled "Using the Smart Agent".

The Smart Agent in Containerized Environments

The VisiBroker Smart Agent Relay (osarelay) enables you to use Smart Agent in containerized environments.

The VisiBroker Smart Agent uses both UDP and TCP to communicate with VisiBroker clients and server implementations. UDP is primarily used for communications from clients and servers to the Smart Agent, and is preferred to TCP because:

- UDP is very lightweight relative to TCP, minimizing network traffic overhead.
- UDP broadcast is used to enable running Smart Agents to be automatically discovered by clients, servers and other Smart Agents.

Although UDP is the primary transport used by the Smart Agent, TCP is used to support the Location Service functionality (see the VisiBroker (for C++ or Java) **Developer's Guide** chapter "Using the Location Service").

However, the use of UDP is a problem in some containerized environments, such as Docker, which do not allow UDP communications across the container boundary. This prevents the Smart Agent from being able to operate in these conditions. The Smart Agent Relay (osarelay) has been developed to provide a solution for this issue.

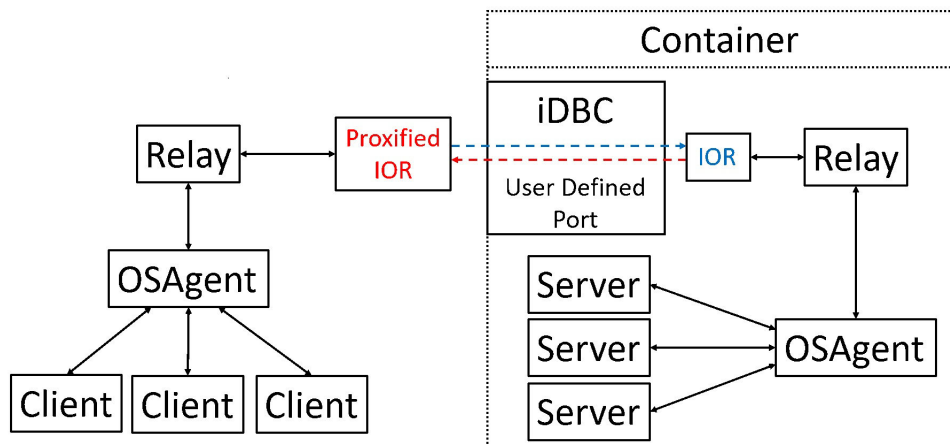
Containers enable application environments to be isolated. Connecting CORBA clients to services running either side of a container boundary NAT layer (see "[What is the CORBA Add-on for Cloud, Containers & Virtual Environments](#)") requires the use of a proxy server, such as I-DBC, to manage the address translation within the CORBA object references. The I-DBC is described in the **Micro Focus IIOP Domain Boundary Controller (I-DBC) v.4.0.0 Deployment Guide**.

The Smart Agent Relay (in concert with I-DBC) enables the Smart Agent to operate across the container boundary. It does this by using CORBA to extend Smart Agent functionality, to send CORBA messages over the domain boundary to another Smart Agent relay. This in turn enables external Smart Agents to talk to internal containerized Smart Agents.

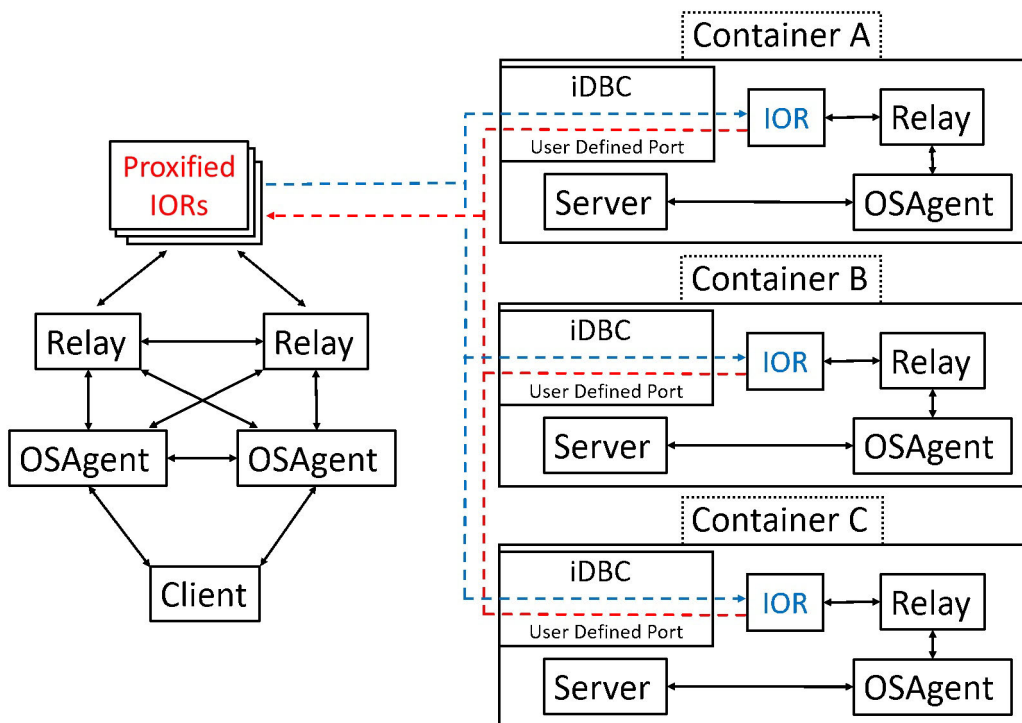
The Smart Agent Relay has been designed to complement Smart Agent in such a way as to be installable non-intrusively along with existing VisiBroker implementations.

Topology of the Smart Agent Relay

The diagram below shows a typical topology of a containerized application that can be invoked upon from outside the container. This example assumes that your services are running inside containers and your clients are executed outside. This is not a hard restriction; it is also perfectly possible to run clients inside containers and run servers outside containers.



A more complex multi-container example is depicted below:



This diagram shows multiple containers, each hosting a server. As before, each server registers with a Smart Agent that is running inside its container. In order to make that service available externally, a Smart Agent Relay is included. This Smart Agent's IOR is proxified by I-DBC and made available to the external Relays. Once the external Relays are able to establish IIOP connections to the internal Relays and vice-versa, any requests from clients and subsequent responses are relayed via the internal

Smart Agent Relay. I-DBC dynamically proxifies object references that are passed as IIOP request or response messages between relays so that these object references can be used by clients effectively to call onto the servers via the i-DBC proxies.

Configuring I-DBC for Use with the Smart Agent Relay Within a Container

To correctly set up interaction between i-DBC and the Smart Agent Relay within a container, there are some options that need to be configured correctly. By default, I-DBC handles *incoming* connections and works as a proxy to forward messages. Some settings need to be activated for *outgoing* connections to be passed through the Smart Agent Relay.

First, ensure that you set ports for I-DBC's `privateDomain` correctly:

- `configs.iDBCProxyCluster1.shared.proxy.privateDomain.acceptors.[0].localAddress.port`

This enables the Smart Agent Relay that exists inside the container to send messages out, via i-DBC. Make sure that whatever value ports you use are published for use with the container.

- `configs.iDBCProxyCluster1.shared.proxy.privateDomain.acceptors.[1].localAddress.port`

This sets up the `privateDomain`'s second acceptor as well, which is necessary for a secure connection.

Secondly, activate the use of the `publicDomain` connectors:

- `configs.iDBCProxyCluster1.shared.proxy.publicDomain.connectors.[0].useConnector`
- `configs.iDBCProxyCluster1.shared.proxy.publicDomain.connectors.[1].useConnector`

Both of these must be set to `true` to allow for outgoing connections. As above, the second connector is for secure connections.

Finally, you must change a pair of I-DBC's proxification options for compatibility with the Smart Agent and Smart Agent Relay:

- `configs.iDBCProxyCluster1.shared.proxy.proxificationOptions.useOriginalKey`

You must set this to `true` so that proxified IORs preserve the correct information passed from the Smart Agent Relay to the Smart Agent.

- `configs.iDBCProxyCluster1.shared.proxy.proxificationOptions.visiOSAgentPerPOA`

This option must be set to `true` to allow the Smart Agent Relay to pass server references bound using `BindSupportPolicy`. This option is specific for use with the Smart Agent Relay so see the section "[I-DBC Proxification using visiOSAgentPerPOA](#)" for more information.

For more information on the above options, and any others you wish to alter, please refer to the ***I-DBC Administrator's Guide***. For an example on how these options are set, see the `basic_bank_agent_with_relay` examples provided with the installation.

I-DBC Proxification using visiOSAgentPerPOA

If you need the Smart Agent Relay to be able to pass server references that are bound to a local Smart Agent using a BindSupportPolicy of `BY_POA`, you must set the I-DBC option `visiOSAgentPerPOA` to `true`.

When binding a Server object reference to a Smart Agent using the `BY_POA` BindSupportPolicy, Smart Agent passes a 'service' IOR (which identifies only the POA and not a specific object) between instances of itself and its clients. VisiBroker clients can use a service IOR as supplied by Smart Agent to generate a full object IOR that may then be used to make an IIOP request back to a specific object.

I-DBC maintains an IOR table which contains a mapping of proxified IORs that it has generated to their original IORs. When binding `BY_POA`, the proxified IOR will be based upon a service IOR. However, the Client generates an IOR (based on the proxified service IOR but with an object identifier included) when making its IIOP request back via I-DBC. Because I-DBC's IOR table does not contain an entry for this generated object IOR, I-DBC responds to the Client with an `OBJECT_NOT_EXIST` message.

Setting the

`configs.iDBCProxyCluster1.shared.proxy.proxificationOptions.visiOSAgentPerPOA` option to `true` ensures that when I-DBC cannot initially find an entry in its IOR table for the object IOR provided, it will try again using a service IOR that is based on the supplied object IOR. This enables I-DBC to locate the appropriate entry for the service in its IOR table and thus identify the correct route back to the Server inside the container so that the request can be delivered.

Note that if your Servers only bind to Smart Agent using the BindSupportPolicy `BY_INSTANCE`, full object IORs will be registered with Smart Agent and so the `visiOSAgentPerPOA` option is not required.

Configuring the Smart Agent Relay

You can control the behavior of the Smart Agent Relay (`osarelay`) using a number of options. These are:

- Command line switches used to control behavior that is in common with the Smart Agent (`osagent`).
- VisiBroker-style properties used to control `osarelay`'s contact with Smart Agents.

Command Line Switches

The command line switches allow you to control behavior that is in common with the Smart Agent

To display details of the available command line switches, run:

```
osarelay -?
```


Logging

You can use the following switches to enable logging:

Option	Description
-v	Turns verbose mode on, which provides information and diagnostic messages during execution. On UNIX, the verbose output is sent to stdout.
+l <options>	Show or enable the logging level: <ul style="list-style-type: none">o - Turn logging on. Log levels: <ul style="list-style-type: none">f - Fatale - Errorw - Warningi - Informationald - Debugginga - All

For example, to start the Smart Agent Relay with comprehensive logging set to on, enter:

```
osarelay +l oa -v
```

By default, log messages are directed to `stdout`. Log messages can alternatively be redirected to a file. The log file name will be generated (based off its process ID), but the location and the maximum log file size can be controlled using the following switches:

Option	Description
-d <pathname>	Sets the OSARELAY_LOG_DIR log directory location.
-ls <size>	Specifies the trimming log size of 1024KB block. Max value is 512, therefore the largest log size is 512MB

Ports

The Smart Agent Relay will partner with the Smart Agents that are listening for traffic on the UDP port defined by the `OSAGENT_PORT` variable, as described in the chapter "Using the Smart Agent" in the VisiBroker (for C++ or Java) *Developer's Guide*. The UDP port that will be used by Smart Agent Relay for communicating with its partner Smart Agents can be set using the `OSARELAY_PORT` environment property. This can be overridden on the Smart Agent Relay's command line using the `-p` switch described below.

Option	Description
-p <UDP_port_number>	Overrides the setting of <code>OSARELAY_PORT</code> (registry setting on Windows systems).

The Smart Agent Relay does not share the same UDP port used by the Smart Agent for listening for discovery requests. If neither `OSARELAY_PORT` nor `-p` is specified then Smart Agent Relay will listen on the port at `(OSAGENT_PORT + 1)`.

Properties

Some aspects of Smart Agent Relay are set using properties, which are defined in the VisiBroker style used by `OSAgent` itself.

Initializing the SmartAgent Relay

Smart Agent Relay finds a partner Smart Agent by either:

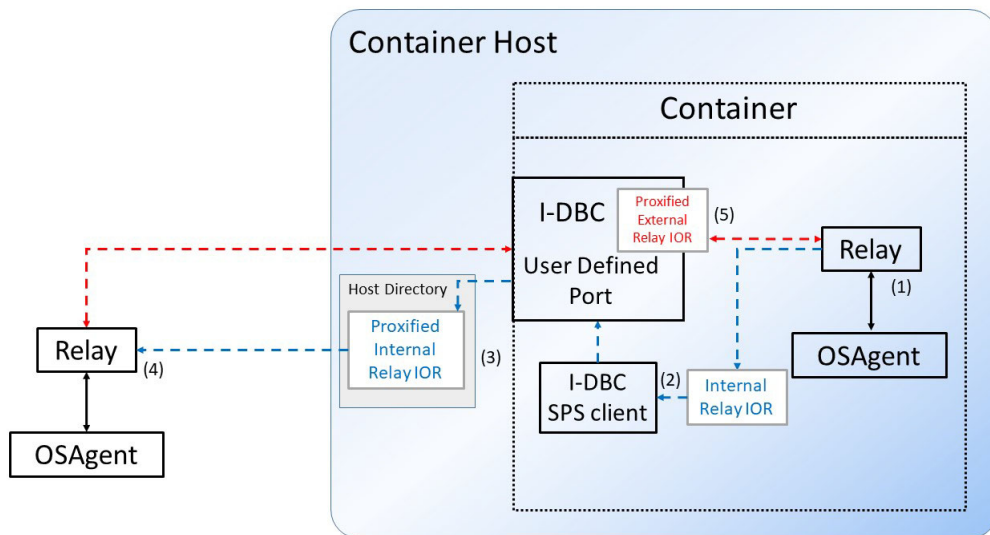
- Sending a direct UDP message to `vbroker.agent.addr: OSAGENT_PORT`, or
- Sending out a UDP broadcast message on `OSAGENT_PORT`.

As described in “Ports”, Smart Agent Relay uses a distinct port for communicating with Smart Agent. It cannot share `OSAGENT_PORT` unless it will be hosted separately from Smart Agent. By using a port number that is distinct from `OSAGENT_PORT`, Smart Agent Relay can reside on the same host as Smart Agent. Since Smart Agent is unaware of Smart Agent Relay's port, Smart Agent Relay must make the initial contact to establish a relationship between the two.

If at start-up the Smart Agent Relay is unable to establish contact with a Smart Agent, it will periodically retry attempts to locate a partner Smart Agent with the frequency defined by the property `vbroker.agent.relay.discoverAgentTimeout` (by default, 2 seconds).

During initialization, Smart Agent Relay will by default share information about other running Smart Agent Relays that it is aware of. However, to get an initial connection established, one Smart Agent Relay must be provided with the IOR for a second Smart Agent Relay.

In the topology depicted below, a Smart Agent Relay (1) that is internal to the Container Host is started first.



During start-up, Smart Agent Relay writes out its own IORs to file:

IOR File	Description
<code>osarelay.iior</code>	Transient IOR for the Smart Agent Relay service.
<code>osarelayadmin.iior</code>	Persistent IOR for the Smart Agent Relay service.

These are default filenames which can be overridden using the properties `vbroker.agent.relay.iior` and `vbroker.agent.relay.adminiior`.

When the Container is started, its start-up script proxifies the Internal Relay IOR (2) by using the I-DBC SPS Client. An example of such a command is provided below:

```
/usr/xtradyne/sps/bin/spsclient -u admin -p admin -d /usr/xtradyne/sps/adm -C "ior deployTransient iDBCProxyCluster1 IOR:123...789 {}" > proxifiedInternalRelay.ior
```

The `osarelayadmin.ior` file of the internal relay after proxification is then copied (3) to an area of the Container Host's filesystem that will be visible to the external relay when it is started.

The external Smart Agent Relay is then started with the internal Relay's proxified IOR supplied on the command line (4) using the property `vbroker.agent.relay.admins`.

An example command line might be:

```
osarelay -Dvbroker.agent.relay.admins=file://  
<path_to_volume>/proxifiedInternalRelay.ior
```

During its initialization, the external Relay will make a call to the internal Relay passing its own IOR. Note that I-DBC proxifies this IOR on-the-fly (5). Once this stage is complete, both internal and external Smart Agent Relays are aware of each other.

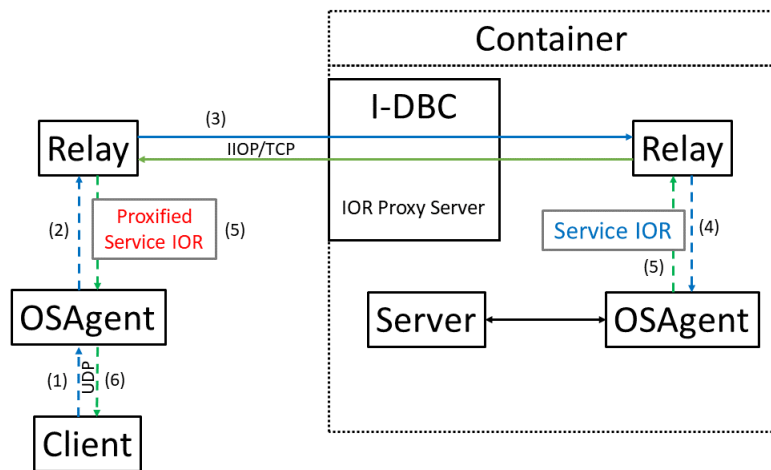
Satisfying Smart Agent Requests

The following sections describe possible outcomes of a Smart Agent request. These are:

- A [Successful Request/Response Cycle](#)
- [No response from the Internal Smart Agent Relay](#)
- [No response from the Internal Smart Agent](#)

Successful Request/Response Cycle

The following diagram illustrates how requests for containerized CORBA services are satisfied using a combination of Smart Agents and Smart Agent Relays.



When a client makes a request for a service (1) on a Smart Agent (OSAgent) which the Smart Agent cannot immediately satisfy, it forwards this request (2) on to any further Smart Agents and Smart Agent Relays that it is aware of. In the arrangement depicted above, the external Relay will package the content of the request into a CORBA one-way request and forward it (3) across the container boundary (via I-DBC) to its peer Relay inside the container. The internal Relay then reconstitutes the Smart Agent request and sends it over UDP to any running internal Smart Agents (4). In this instance, the internal Smart Agent does have a registered entry for the requested service, so it is able to respond to the internal Relay with a Service IOR (5). The internal Relay packages the content of the response, including the Service IOR, into a CORBA one-way request and sends it back across the container boundary. I-DBC recognizes that the payload of the CORBA message contains an IOR and proxifies it on-the-fly, such that the IOR received by the external Relay is already proxified. The external Relay then reconstitutes a Smart Agent response message, including the Proxified Service IOR, and sends it on to the external Smart Agent which is then able to satisfy the original client request (6).

Transient Error Mitigation

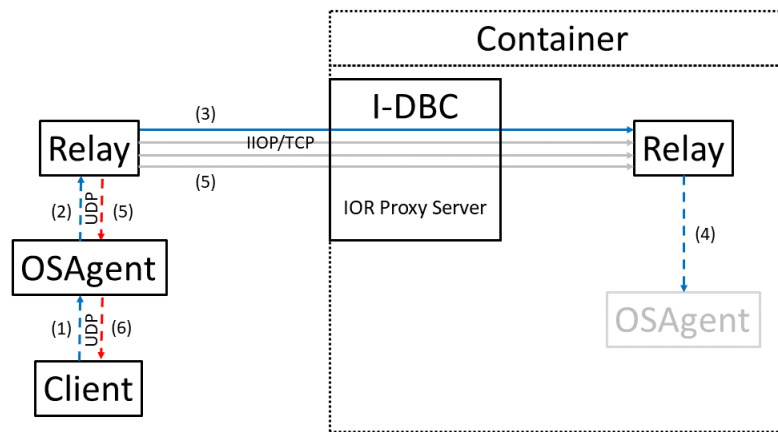
As stated previously (see [“The Smart Agent in Containerized Environments”](#)), Smart Agent uses UDP for communication with VisiBroker clients and server implementations. UDP is an “unreliable” transport, in that it does not

guarantee whether data has arrived at its destination, nor, if it has, whether it is intact. The Smart Agent application layer protocol builds in retry behavior and message integrity checking to address these issues. The CORBA communication between Relays is TCP based but in the event of short-duration connection issues it might also suffer message loss.

Smart Agent Relay uses a combination of timeouts and retry schedules to mitigate the risk of a message (whether UDP-based or CORBA one-way) not successfully arriving at its destination. These measures are configurable and are described in the following sections.

No response from the Internal Smart Agent Relay

If the external Smart Agent Relay does not receive a response from the internal Relay within a specific period of time, it will retry the request. This is illustrated below:



The external arrangement of Client, Smart Agent and Smart Agent Relay follows the [Successful Request/Response Cycle](#) previously described. However, in the example illustrated above, the requests forwarded on by the internal Relay are not being responded to (perhaps because its partner internal Smart Agent(s) has become unavailable).

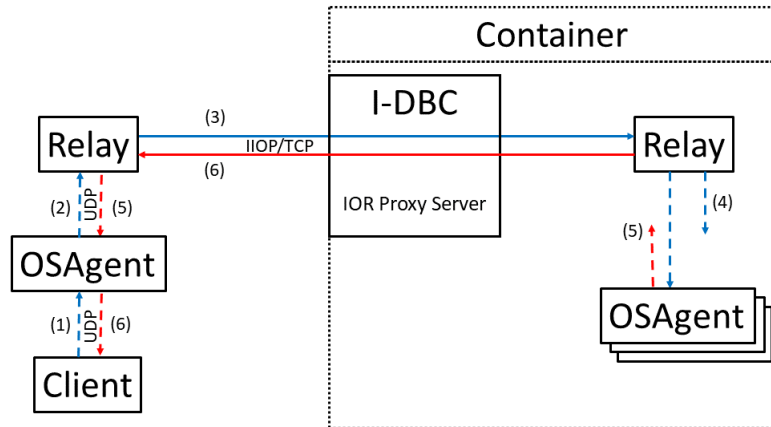
The initial client request (1) is forwarded by Smart Agent (2) on to the external Smart Agent Relay, which re-packages the request into a CORBA one-way request and sends it to the internal Relay (3) In this case, the internal Relay reconstitutes the Smart Agent request and sends it on to an address and port where it believes its partner Smart Agent is listening (4).

Timeout and retry properties

In this scenario no response is forthcoming from that request, and therefore no Relay response is generated by the internal Smart Agent Relay. After a period of time defined by `vbroker.agent.relay.relayRequestTimeout`, the CORBA request (3) times out and is retried. This occurs as many times as defined by `vbroker.agent.relay.maxFullRequestRetries`. If no response has been received after the maximum number of timeouts for that request has occurred, a TIMEOUT response will be returned by the external Relay.

No response from the Internal Smart Agent

Transient errors can potentially occur at the link between the internal Smart Agent Relay and its partner internal Smart Agent(s), as shown below:



Such errors could result in either the request failing to arrive at the Smart Agent (4) or the response failing to arrive back at the Relay (5). Smart Agent Relay mitigates against both these circumstances by implementing a timeout for a response to be received.

If the internal Relay does not receive a response from any of its internal Smart Agents within the time defined by `vbroker.agent.relay.agentRequestTimeout`, it times out and immediately returns a `TIMEOUT` response to the external Relay. This is then forwarded on to the external Smart Agent which can then respond to the original client request with `FAIL`.

Property Reference

The properties used by the Smart Agent Relay, together with the corresponding environment variables where there are any, are as follows:

Property	Environment variable	Default	Description
<code>vbroker.agent.relay.admins</code>	<code>none</code>	<code>""</code>	Comma-separated list of proxified IORs of peer Smart Agent Relays. Can use either of <code>osarelay.ior</code> or <code>osarelayadmin.ior</code> as the source IORs.
<code>vbroker.agent.relay.discoverAgentTimeout</code>	<code>OSARELAY_DISCOVER_AGE_NT_TIMEOUT</code>	2000	The period (in milliseconds) between attempts to locate a partner Smart Agent while the Relay is without a partner.
<code>vbroker.agent.relay.relayRequestTimeout</code>	<code>OSARELAY_RELAY_REQUEST_TIMEOUT</code>	3000	The time, in milliseconds, between attempts to retry a failed request.
<code>vbroker.agent.relay.maxFullRequestRetries</code>	<code>OSARELAY_MAX_REQUEST_RETRIES</code>	0	The maximum number of times a failed request is retried before returning a failure response.
<code>vbroker.agent.relay.agentRequestTimeout</code>	<code>OSARELAY_AGENT_REQUEST_TIMEOUT</code>	1000	The time, in milliseconds, between attempts to retry a request.

Property	Environment variable	Default	Description
vbroker.agent.relay.timeoutCheckInterval	OSARELAY_TIMEOUT_INTERVAL	500	The time, in milliseconds, between checks for Smart Agent request timeouts.
vbroker.agent.relay.announceSelf	none	true	Whether a relay should automatically register itself as a destination for locate requests from another relay when it becomes aware of that other relay.
vbroker.agent.relay.pushRelays	none	true	When a relay becomes aware of another relay if this property is true it will send all current registered destinations to it.
vbroker.agent.relay.pullRelays	none	true	When a relay becomes aware of another relay if this property is true it will request all the other relay's known destinations and register them as destinations.
vbroker.agent.relay.unreachableCleanupAfter	none	600	The minimum limit, in seconds, on how long a relay destination will be retained after it is first detected to be no longer reachable.
vbroker.agent.relay.adminior	none	osarelay.admin.ior	The filename to output the Relay service IOR to. This is a persistent lifecycle IOR (that is, the IOR will be the same if the relay is restarted if configured with a fixed IIOP listen port).
vbroker.agent.relay.ior	none	osarelay.ior	The filename to output a transient lifecycle IOR for the Relay service (that is, previous file contents will become invalid whenever the service is restarted).
vbroker.agent.relay.port	OSARELAY_PORT	OSAGENT_PORT +1 (14001)	The port number used to connect the Relay to the OSAGENT_PORT domain.
vbroker.agent.relay.addrFile	OSARELAY_ADDR_FILE	OSAGENT_ADDR_FILE (null)	A file that stores the IP address or host name of a host running a Smart Agent.
vbroker.agent.relay.localFile	OSARELAY_LOCAL_FILE	OSAGENT_LOCAL_FILE (null)	Specifies which network to use on Multi-home machines.
vbroker.agent.relay.logDir	OSARELAY_LOG_DIR	VBROKER_ADM	Specifies the directory for the OSARELAY log to reside in.
vbroker.agent.relay.logLevel	none	i	Specifies the log level of messages to be written to the log file. Acceptable values are: <ul style="list-style-type: none"> • Debug(d) • Informational(i) • Error(e) • Warning(w) • Fatal(f) • All(a) <p>Equivalent to the -l switch described under "Logging".</p>

Property	Environment variable	Default	Description
vbroker.agent.relay.logSize	none	1	Sets the maximum log file size (in megabytes). Equivalent to the <code>-ls</code> switch described under "Logging". The maximum size is 512.
vbroker.agent.relay.verbose	none	false	Turns Smart Agent Relay logging's verbose mode on (true) and off (false). Equivalent to the <code>-v</code> switch described under "Logging".
vbroker.agent.relay.broadcastOff	none	false	When this is set to true, Smart Agent Relay will not send out a broadcast to find other agents. You can use this in combination with <code>vbroker.agent.relay.addrFile</code> ; if this is set to true, the relay will communicate only with the agents listed in <code>addrFile</code> .
vbroker.agent.relay.ignoreSignal	none	none	Allows you to specify that certain signals are to be ignored. Possible options are: <ul style="list-style-type: none"> • Quit(quit) • Hangup(hup) • Interrupt(int) <p>Bourne and Korn shell users are recommended to run the Smart Agent Relay with <code>ignoreSignal=hup</code>.</p> <p>If you wish to ignore more than one of these signals, list the option separately for each one. For example:</p> <pre>vbroker.agent.relay.ignoreSignal=hup vbroker.agent.relay.ignoreSignal=int</pre> <p>Note that this usage of repeated statements is unique to this option.</p>

In addition to `osagent`-specific and `osrelay`-specific options the Smart Agent Relay will accept any (C++) ORB configuration option to modify its behavior as a service. For example, in order to configure a fixed IIOP listen port to 12345 one might pass the argument:

```
-Dvbroker.se.iiop_tp.scm.iiop_tp.listener.port=12345
```

See the **VisiBroker C++ Programmer's Guide** for the complete list of ORB options.

Note

These properties can also be added to a properties file, and passed to the relay using the `-OSAppropStorage` option. This is the recommended method for most environments.

For example:

```
osrelay -OSAppropStorage /path/to/relay_properties.txt
```

The properties in this file should be separated by new lines.

Options passed to the Smart Agent Relay take priority over Environment Variables, and the Environment Variables take priority over options found inside the property file.

Updating SPS Configuration Items

When you are using I-DBC inside Docker, it is essential to configure the Security Policy Server (SPS) properly to allow your application to run correctly. The `entrypoint_common.sh` script, copied into the Docker container during the creation of the I-DBC Docker image, provides an example of how to configure SPS using commands run in a shell script.

Introduction

Whether expanding on the functions in `entrypoint_common.sh`, or providing your own mechanism, one challenge is to determine the SPS configuration item names and values that must be set to allow your application to work as expected.

This section will walk through the steps to determine the SPS configuration item name of the server SSL version -

```
configs.iDBCProxyCluster1.shared.proxy.SSL.SSLServer.crypto.method.
```

The approach is to use the Administration Console to make the update. Then use the `diff` command on a "before" and "after" version of the SPS configuration file to determine the server SSL version configuration item name.

Once you understand how to determine the name of this configuration item, the same process can be used to determine the name of other configuration items.

The combination of the SPS configuration item names and their corresponding values allow for I-DBC to be configured using an automated approach (as is done via `entrypoint_common.sh`) rather than non-automated interactive approach (as is done via the Administration Console).

When running outside Docker, paths and commands given in this document assume installation on a Linux machine. If using a Windows machine, adjust the paths and commands to suit Windows.

Prerequisites

Before you make any update, ensure that:

- The Administration Console is installed. See the instructions in the [CORBA in the Cloud or in Virtual Environments](#) and [CORBA in Containers](#) chapters. The following procedures assume that the Administration Console is installed in `<installdir>/adminconsole`.
- The CORBA Add-on for Cloud, Containers & Virtual Environments is installed. See the instructions in the [CORBA in the Cloud or in Virtual Environments](#) and [CORBA in Containers](#) chapters. The following procedures assume that the CORBA Add-on for Cloud, Containers & Virtual Environments is installed in `<installdir>`.
- You have a JRE installation that can run the Administration Console
- You have Docker installed
- You can determine the IP address of your machine

You will need two windows:

- A window capable of running Docker commands to run the Docker container. I-DBC will be run inside this container.
- A window to run the Administration Console. The Administration Console will be run outside Docker. It will connect to the SPS running inside the Docker container.

Build the Base OS and I-DBC Docker Images

Open a window capable of running Docker commands. This window will be used to build the Docker images, as well as run the Docker container.

Two Docker images must be created:

- The base operating system image. The CORBA Add-on for Cloud, Containers & Virtual Environments supports the use of either CentOS or Ubuntu as the base Docker image; this example uses a CentOS base image.
- An I-DBC image, built upon the CentOS base image.

Build the Base Docker Image

Build the base Docker image, using CentOS in this example, by issuing the following command:

```
cd <installdir>/docker/common/centos_layer
docker build -t base-os-layer .
```

Note:

The '.' character is an essential part of the `docker build` command.

See [“The Dockerfile for CentOS”](#) for more information.

Build the I-DBC Docker Image

Build the I-DBC Docker image by issuing the following command.

```
cd <installdir>docker/common/idbc_layer
docker build -t idbc-layer .
```

Note:

The '.' character is an essential part of the `docker build` command.

See [“The Dockerfile for I-DBC”](#) for more information.

Run the I-DBC Docker Image

Run the Docker container in the same window where the Docker images were built, as follows:

- 1 Determine the IP address of your machine.
- 2 Run the following command, replacing `<ip_addr>` with your actual IP address:

```
docker run --name idbc_container --publish 15000:15000
--env MF_HOST_IP=<ip_addr> --env MF_IDBC_PORT=3000
-it idbc-layer
```

- This command starts a Docker container named `idbc_container`. Port 15000 is opened into the container, which will allow the Administration Console to connect from outside the Docker container.
- The command also starts an interactive shell in the container. You will see a command prompt similar to the following:

```
[corba@c41afce343e6 idbc]$
```

Save the Current SPS Configuration

Inside the Docker container, save a "before" version of the SPS configuration file. This will allow you to use the `diff` command after the Administration Console is used to update the configuration item for the server SSL version. In the Docker container interactive shell enter:

```
cd /home/corba/microfocus/idbc/sps/adm
cp dbc.config dbc.config_before
```

Start I-DBC inside Docker

In the same window, issue the following commands to start I-DBC:

```
cd /home/corba/microfocus/idbc
./startStop.sh start
```

Look for a message similar to the following:

```
Starting ... ok
```

Change the Server SSL Version

Open a second window. Be sure to set the `JAVA_HOME` environment variable to point to a version of Java that can run the Administration Console.

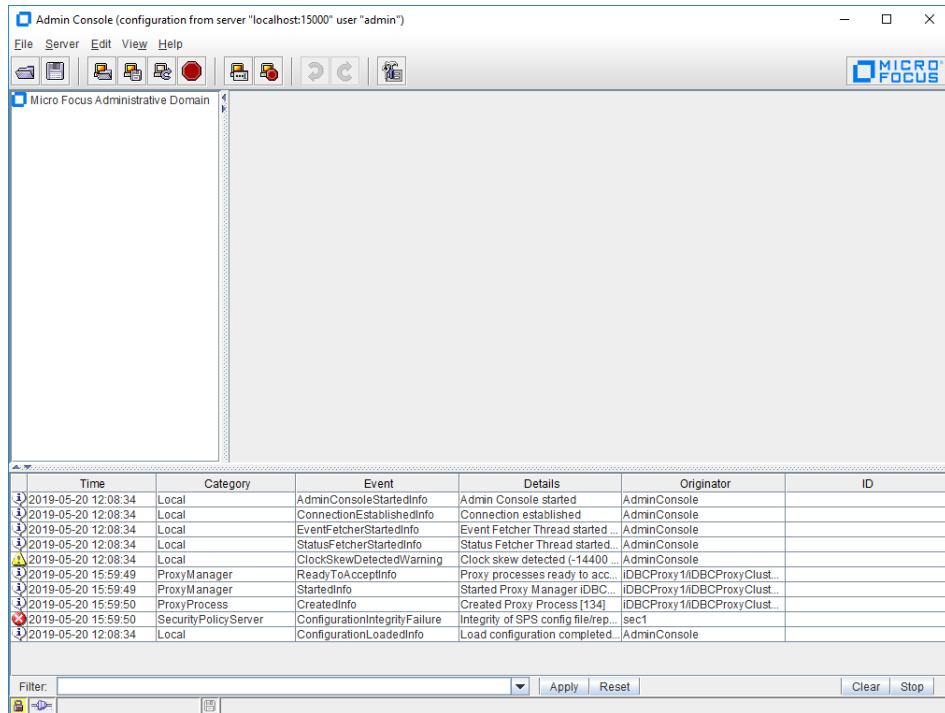
Start the Administration Console:

```
cd <installdir>/adminconsole/bin
./AdminConsole
```

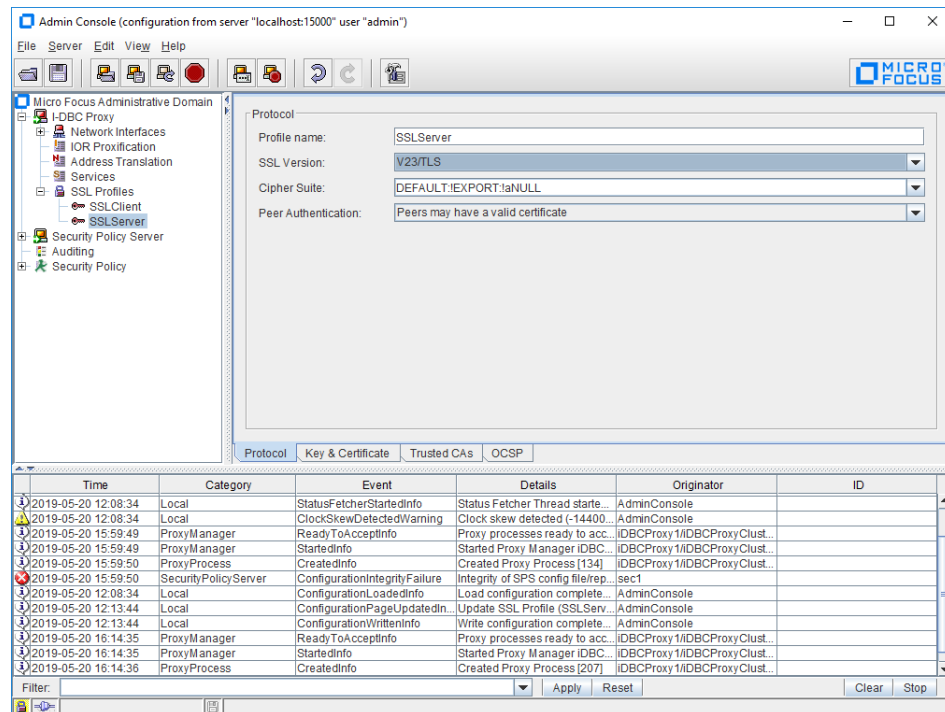
The **Login on Security Policy Server...** window displays.

You can accept the default Address and User ID values. Enter the password (**admin**) and press **OK**. This will connect the Administration Console to the Security Policy Server running inside Docker.

The **Admin Console** window displays:

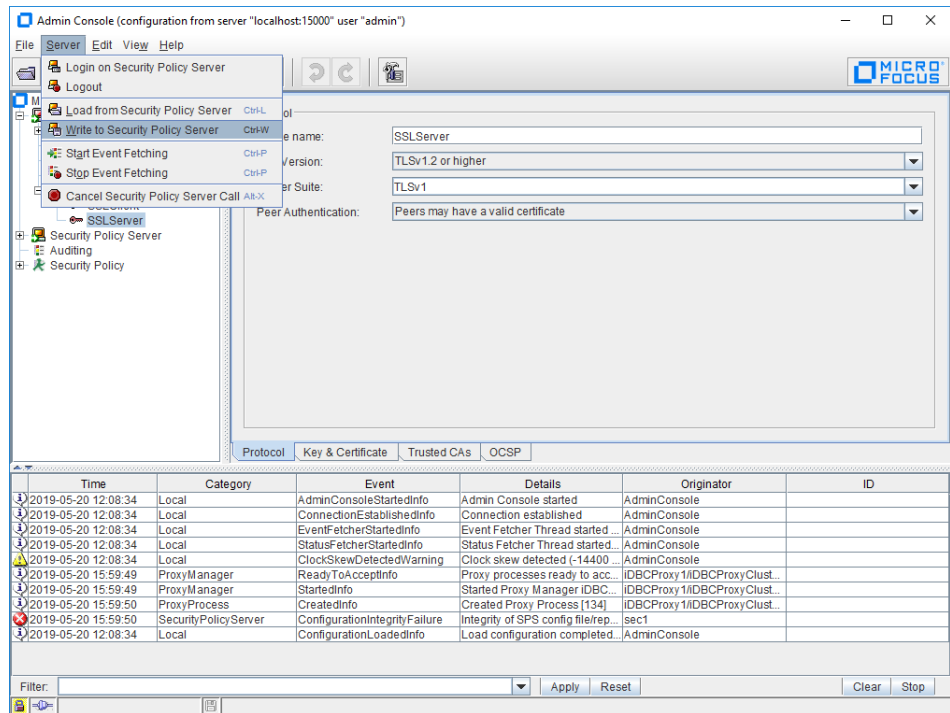


Double click **Micro Focus Administrative Domain**. Then expand **I-DBC Proxy** and **SSL Profiles**. Click on **SSLServer**.



In the **SSL Version:** drop-down, select **TLSv1.2 or higher**.

From the **Server** menu item select **Write to Security Policy Server**.



A pop-up appears indicating that DCB components need to be restarted. Press **Yes**.

The Administration Console will then work to save to updated value into the SPS configuration file inside Docker.

Diff the config file

In the window where the Docker container was started, use the interactive shell to navigate to the SPS configuration directory and run the `diff` command on the current SPS config file against the "before" copy of the file:

```
cd /home/corba/microfocus/idbc/sps/adm
diff dbc.config dbc.config_before
```

The results of the `diff` command may look something like this:

```
92d91
<     "filename" = "ProxyKey.pem"
1932,1933c1931
< -----END CERTIFICATE-----
< "
---
> -----END CERTIFICATE-----"
1993,1994c1991
< -----END CERTIFICATE-----
< "
---
> -----END CERTIFICATE-----"
1998,1999c1995,1996
<     "ciphersuite" = "TLSv1"
<     "method" = "tlsv1_2"
```

```

---
>         "ciphersuite" = "DEFAULT:!EXPORT"
>         "method" = "v23"
4141c4138
<     "version" = "3.2.0"
---
>     "version" = "3.1"
4179,4180c4176,4177
< "version" = "1"
< }
---
> "version" = "0"
> }

```

Even though only one configuration item was updated, the Administration Console drives multiple updates into the SPS configuration file.

Looking closely at the diff results, we can see `"method" = "tlsv1_2"`, which is related to the server SSL version update we just made. We have two pieces of information:

- 1 The SPS configuration file value for the server SSL version is: `tlsv1_2`
- 2 The partial SPS configuration item name for the server SSL version is: `method`

Structure of the SPS Configuration File

The full name of the server SSL version configuration item must still be determined. To get the full name, you need to traverse the SPS configuration file. In order to do that, an understanding of the structure of the SPS configuration file is helpful.

A simplified configuration file looks like this example:

```

{
  "Scope1" = {
    "config_item1" = "value1"
    "config_item1" = "value2"
  }
  "Scope2" = {
    "config_item3" = "value3"
    "Scope3" = {
      "config_item4" = "value4"
    }
  }
}

```

The configuration file is divided into *scopes*. Each scope begins with '{' and ends with '}'. Except for the scope at the beginning of the file - the *main* scope - each scope has a name.

Inside each scope there are configuration items with their corresponding values. A scope can also contain a scope.

In the simplified configuration file:

- There are 3 scopes: Scope1, Scope2 and Scope 3
- Scope3 is a "sub-scope" contained within Scope2, where Scope2 is the "parent scope"
- Scope1 and Scope2 are "sibling" scopes where they share the same parent - the "main" scope

The name of a configuration item depends on the scope it is in, as well as all the scopes it is contained in.

Consider "config_item4". It is contained in "Scope3", which itself is contained in "Scope2". So the full configuration item name for "config_item4" is: `Scope2.Scope3.config_item4`.

Determine the Full Name of the Server SSL Version Configuration Item

The output of the `diff` command (see ["Diff the config file"](#)) showed that:

- The SPS configuration file value for the server SSL version is `tlsv1_2`
- The partial SPS configuration item name for the server SSL version is `method`

We know that part of the server SSL version configuration item is "method". We need to traverse all the scopes in the configuration file to determine the full name.

Start by using the `vi` editor to edit the SPS configuration file, using the following commands:

```
cd /home/corba/microfocus/idbc/sps/adm
vi dbc.config
```

Use the `vi '/'` command to find "tlsv1_2" in the configuration file:

```
/tlsv1_2
```

The cursor will land on the "method" = "tlsv1_2" line in the configuration file:

```
...
}
"cipherSuite" = "TLSv1"
"method" = "tlsv1_2"
}
...
```

Use the `vi '%'` command to determine the scope containing the "method" configuration item. Move the cursor onto the '}' character below the "method" configuration item. Type '%'.

The cursor will land on the "crypto" = { line in the configuration file:

```
...
}
"SSLServer" = {
"crypto" = {
"RSA" = {
"keyFiles" = {
"certificate" = {
...

```

The "method" configuration item is contained in the "crypto" scope.

From this, you can begin to build the server SSL version configuration item name: `crypto.method`

Just above "crypto" in the file, we see the "SSLServer" scope is the parent scope for the "crypto" scope. Continue to build the server SSL version configuration item name `SSLServer.crypto.method`

Move the cursor onto the '}' character above the "SSLServer" scope. Type '%'.
The cursor will land on the "SSLClient" scope in the configuration file:

```
...
}
"SSL" = {
  "SSLClient" = {
  ...
```

"SSLClient" is a sibling scope to "SSLServer". Both scopes are contained within the "SSL" scope. The server SSL version configuration item name is now: `SSL.SSLServer.crypto.method`

Move the cursor onto the '}' character above the "SSL" scope. Type '%'.
The cursor will land on the "CSIV2" scope in the configuration file:

```
...
}
"CSIV2" = {
  "CSS" = {
  ...
```

Scope "CSIV2" is a sibling scope to "SSL". But we need to find the parent scope of "SSL".

Move the cursor onto the '}' character above the "CSIV2" scope. Type '%'.
The cursor will land on the "ADF" scope in the configuration file:

```
...
}
"proxy" = {
  "ADF" = {
    "cache" = {
  ...
```

Scope "ADF" is a sibling scope to "SSL". We can see that scope "proxy" is the parent scope of "ADF". Since "ADF" is a sibling scope to "SSL", "proxy" is also the parent scope of "SSL". The server SSL version configuration item name is now:

```
proxy.SSL.SSLServer.crypto.method
```

Put the cursor on the '}' character above "proxy" and use '%' to get the sibling scope to "proxy".
The cursor will land on the "nodeManager" scope in the configuration file:

```
...
}
"shared" = {
  "nodeManager" = {
    "PAM" =
  ...
```

Scope "nodeManager" is a sibling scope to "proxy". We can see that scope "shared" is the parent scope of "nodeManager". Since "nodeManager" is a sibling scope to "proxy", "shared" is also the parent scope of "proxy". The server SSL version configuration item name is now:

```
shared.proxy.SSL.SSLServer.crypto.method
```

Put the cursor on the '}' character above "shared" and use '%' to get the sibling scope to "shared".
The cursor will land on the "description" scope in the configuration file:


```

...
}
"description" = {
  "edition" = "enterprise"
}
...

```

Scope "description" is a sibling scope to "shared". But we need to find the parent scope of "shared".

Move the cursor onto the '}' character above the "description" scope. Type '%'.

The cursor will land on the "DBC's" scope in the configuration file:

```

...
}
"configs" = {
  "iDBCProxyCluster1" = {
    "DBC's" = {
      "iDBCProxy1" = {
    }
  }
}
...

```

Scope "DCBS" is a sibling scope to "shared". We can see that scope "iDBCProxyCluster1" is the parent scope of "DBC's". Since "DBC's" is a sibling to scope "shared", "iDBCProxyCluster1" is also the parent of scope "shared". Further, we can see that scope "configs" is the parent of scope "iDBCProxyCluster1".

Note that the parent scope of "configs" is the main scope. So we now have the complete name of the server SSL version configuration item:

```

configs.iDBCProxyCluster1.shared.proxy.SSL.SSLServer.
crypto.method

```

Using the Configuration Item Name and Value

We have determined the following:

- The SPS configuration file value for server SSL version is `tlsv1_2`
- The full SPS configuration item name for the server SSL version is `configs.iDBCProxyCluster1.shared.proxy.SSL.SSLServer.crypto.method`

As shown in the `entrypoint_common.sh`, the Set Dictionary Value command - `setdictvalue` - can be used to set a configuration item in the SPS configuration item. The steps to using `setdictvalue` to update a configuration item are:

- 1 Rename the SPS configuration file as we will create a new configuration file
- 2 Use the `cat` command to pipe the contents of the renamed SPS configuration file into the `setdictvalue` command
- 3 The `setdictvalue` command will use the configuration item name and value to update the configuration file
- 4 Pipe the results of `setdictvalue` into the new configuration file
- 5 Remove the renamed configuration file

You must perform these steps **before** starting I-DBC.

For example:

```
cd /home/corba/microfocus/idbc/sps/adm
mv dbc.config dbc.config.tmp
cat dbc.config.tmp | setdictvalue "-" "
configs.iDBCProxyCluster1.shared.proxy.SSL.SSLServer.
crypto.method " " tlsv1_2" > dbc.config
rm dbc.config.tmp
```

Published Ports with Docker

When you ran the Docker container in the example above (see [“Run the I-DBC Docker Image”](#)), the command was as follows:

```
docker run --name idbc_container --publish 15000:15000 --
env MF_HOST_IP=<ip_addr> --env MF_IDBC_PORT=3000 -it idbc-
layer
```

This particular Docker command published port 15000, which allows the Administration Console to connect to the Security Policy Server. While your application will require other ports to be opened in order to access it from outside Docker, it might be considered a security risk for your application to have port 15000 published.

Once you understand all the SPS configuration item names and values that need to be set for your application to work correctly, and you have automated the process of setting those configuration items, you may want to consider removing port 15000 as one of the published ports when running a Docker container.

Using I-DBC to Proxy Transient and Persistent IORs

Usually a Docker container will have its own network that makes it challenging for a CORBA client running outside of the Docker container to contact a CORBA server running inside the Docker container.

IORs created inside of the Docker container are "proxified", providing an address and port that is accessible outside of the Docker container. Invocations from clients using the proxified IOR will pass through I-DBC running inside the Docker container and passed along to the server.

A proxified IOR can be either transient or persistent. A transient IOR is only valid for the lifetime of the server that created the IOR. A persistent IOR is valid over successive instantiations of the target server and POA.

The `entrypoint_common.sh` file has a `proxify_ior` function. The `spsclient` command line tool is used to proxify an IOR. Currently this function proxifies all IORs as transient by using the `deployTransient` subcommand of `spsclient`. This means that:

- Docker "forgets" about all proxified IORs when the Docker container is stopped.
- If a Docker container is stopped and subsequently restarted, all IORs are proxified as transient IORs again, allowing for an orderly restart of I-.

If persistent proxified IORs are preferred, then the `proxify_ior` function in `entrypoint_common.sh` can use the `deploy` subcommand of `spsclient` instead of `deployTransient`. (Alternatively you may have your own implementation to produce the same effect).

Be aware that when a Docker container is stopped, I-DBC will remember all persistent proxified IORs. If the Docker container is restarted, then care must be taken not to reproxify any persistent IORs, as I-DBC is already aware of them.

Using CORBA in a Kubernetes Environment

Kubernetes is a container orchestration system. It is an open-source system for automating the deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery.

For more information about Kubernetes, see <https://kubernetes.io/>.

Introduction

Kubernetes provides a number of powerful capabilities such as:

- Scaling of container-based applications
- Monitoring of applications
- Managing of applications
- Rolling out new versions of applications without downtime

Kubernetes is supported in many cloud provider environments, and can also be run on-premises.

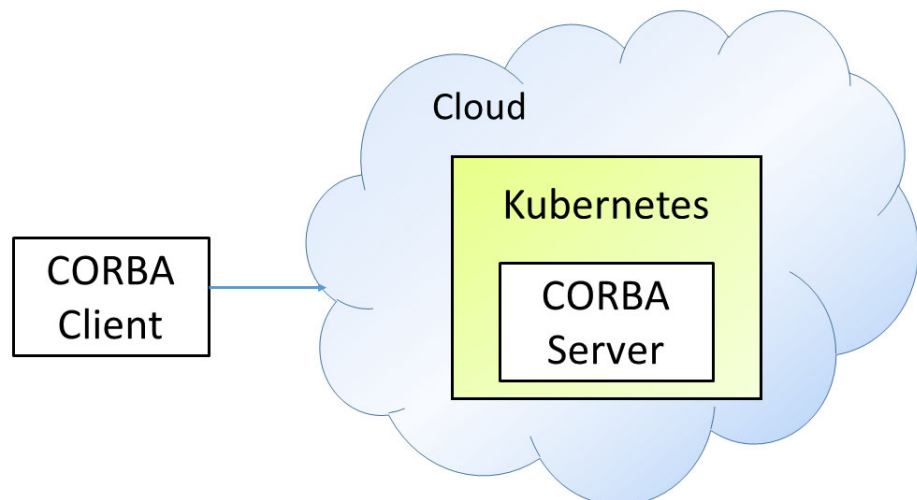
This chapter describes how to use the CORBA Add-on for Cloud, Containers & Virtual Environments to run your CORBA application in a Kubernetes environment.

Example Configuration

There are many ways to leverage CORBA applications with Kubernetes. One approach is to set up the client and server sides of the application as follows:

- Run the server side of the application inside Kubernetes
- Run the client side of the application outside Kubernetes

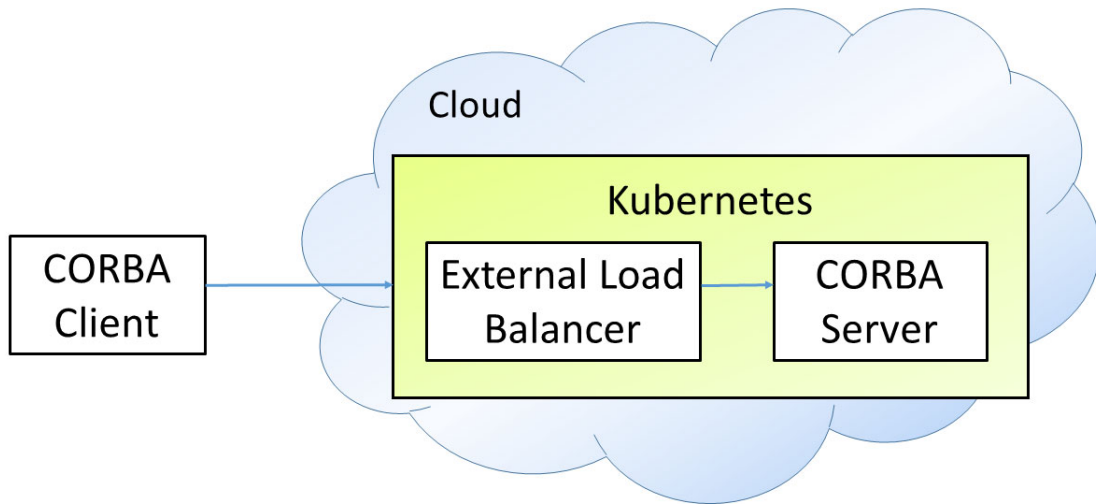
This chapter focuses on that approach, as shown in the following diagram.



External Load Balancer

The CORBA client must be able to access, from outside, the CORBA server running inside Kubernetes. Kubernetes provides several ways to gain that access. One way is to use an external load balancer. In addition to providing access from the outside world, the external load balancer balances incoming traffic from clients across multiple server replicas.

External load balancers are common in cloud provider environments, and can be used on-premises as well.



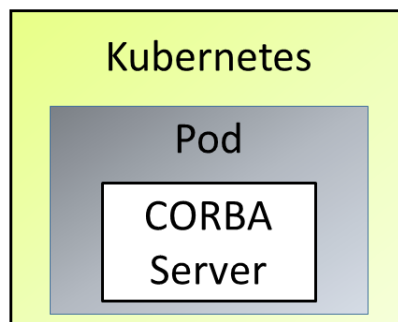
All the Kubernetes examples in this document assume the use of an external load balancer.

Pods

An important Kubernetes concept is that of *objects*. There are several different object types in Kubernetes, with one type being a *pod*.

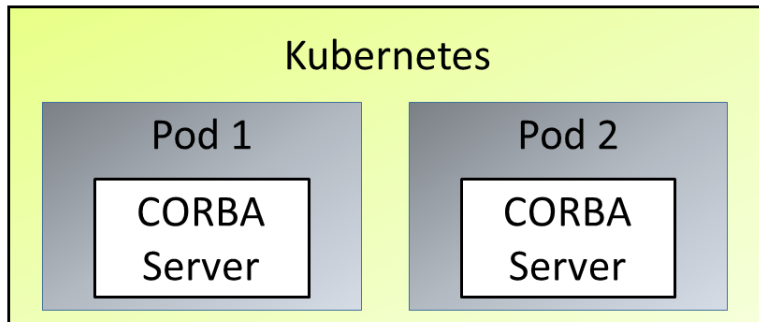
A pod is a group of related containers. From a Kubernetes perspective, they are the basic unit for deployment, horizontal scaling, and replication.

For CORBA applications inside a Docker container, the container will be packaged into a Kubernetes pod.



Replicas

If the CORBA server running inside a Kubernetes pod is developed in a way that permits replication, then the CORBA server can run in multiple pods. This can be done in Kubernetes by using *replica sets*, where multiple instances of the same pod are run. In the figure below, Pod 1 and Pod 2 form a replica set.



Kubernetes Cluster

A Kubernetes cluster will have a control plane or master node, and will often have multiple nodes where the pods run. The control plane and nodes can run on separate physical or virtual machines.

In the previous figure, each pod can run on a separate node. This can provide a degree of high availability, where, if the node running the CORBA server in Pod 1 goes down, the CORBA server running in Pod 2 is still available.

Kubernetes Deployments

Deploying the server side of a CORBA application inside Kubernetes requires the following:

- A pod, which runs the Docker container, which in turn runs the server side of the CORBA application
- A replica set, which is a set of pods running concurrently on separate Kubernetes nodes

To define these to Kubernetes, create a *deployment object*.

A deployment consists of several important components:

- The name and location of the Docker image containing the CORBA server
- The environment variables passed to the Docker container
- The ports opened to the Docker container
- Volume mounts, if any
- Checks that can be run to determine if the pod is ready to process client invocations, and to determine if the required processes running in the pod are indeed running, and potentially if they are running well or not

Note:

While Kubernetes pods and Docker containers are distinct, the CORBA Add-on for Cloud, Containers & Virtual Environments examples described here

run only a single container in each pod. Therefore when this chapter uses the term *pod*, it is often interchangeable with the term *container*.

When running Docker containers manually using the `docker run` command, the information provided to the command includes:

- The ports to open allowing traffic into the container
- The environment variables and their values passed into the container
- Volumes mounted into the container
- The Docker image to run

Since Kubernetes takes control of running the pod, all this information is defined in the deployment.

Kubernetes needs to know the name and location of the Docker image so that it can start pods that run the Docker container. Normally you will publish the Docker image to a registry that Kubernetes can access.

Licensing for the CORBA product and I-DBC (see “[Components](#)”) is provided via a Docker volume when running Docker containers manually. For Kubernetes, a *secret* object type (see “[Kubernetes Secrets](#)”) is used to contain licensing information.

Kubernetes provides powerful monitoring and managing of pods using the following methods:

- `readinessProbe`
- `livenessProbe`

A *readinessProbe* is a means by which Kubernetes can tell when a pod is ready to service client invocations. The CORBA Add-on for Cloud, Containers & Virtual Environments examples look for the existence of a proxified IOR. Once the proxified IOR is created inside the pod, the pod is considered ready.

A *livenessProbe* is a means by which Kubernetes can tell when the pod is running well or not. The CORBA Add-on for Cloud, Containers & Virtual Environments examples check that both I-DBC and the server are running.

These probes are configured with initial wait, retry, and timeout values. If the probe fails, then Kubernetes considers the pod not to be running well. It will schedule an automatic restart of the pod, where the old pod is stopped and a fresh pod is started.

Kubernetes Services

A *service* is another Kubernetes object type. Services provide the means to discover the CORBA server operating in Kubernetes.

Kubernetes supports multiple service types. For the CORBA Add-on for Cloud, Containers & Virtual Environments examples, the *LoadBalancer* service type will be used. The LoadBalancer service type works with the external load balancer.

The service definition provides two important pieces of information required by the CORBA client:

- The hostname or IP address of the external load balancer
- The external load balancer port number

When the CORBA client makes an invocation directed at the hostname/IP and port of the service, the external load balancer receives the network traffic, and forwards it via Kubernetes to one of the pods in the replica set. The pod will pass the network traffic through the Docker container to the CORBA server.

Kubernetes Secrets

A *secret* is a Kubernetes object that is intended to hold sensitive information. The CORBA Add-on for Cloud, Containers & Virtual Environments examples use a secret object to hold the licensing information needed to run the CORBA server and I-DBC.

The secret is mounted as a volume in the deployment object.

YAML Files

There are multiple ways to create objects in Kubernetes:

- By using the `kubectl` command with operands to define the object
- By using the `kubectl` command to apply a `.yaml` file

This chapter will:

- Illustrate how to define the *secret* object to hold licensing information by running a script that in turn runs using the `kubectl` command
- Illustrate how to define the service and deployment objects by using `.yaml` files

Some advantages to using `.yaml` files are:

- Defining a complex object such as a deployment requires a complex `kubectl` command with many operands. Putting the complexity in a structured `.yaml` file makes for a much simpler `kubectl` command, and makes the object definition much more readable
- The `.yaml` files can be maintained in source control

CORBA Application Considerations for Kubernetes

There are some aspects to consider before attempting to put a CORBA application into Kubernetes.

Client Considerations

The CORBA Add-on for Cloud, Containers & Virtual Environments examples use the external load balancer as the entry into Kubernetes. When a client makes an invocation, a connection will be established between the client and the external load balancer. The external load balancer will then direct this invocation via Kubernetes to one of the pods in the replica set.

While this connection remains in place, client invocations over this connection will always go to the same pod. A typical sequence might be:

- A client makes an invocation to a `Bank` object, implemented by a CORBA server running in Kubernetes
- A connection is established between the client and the external load balancer
- The external load balancer, via Kubernetes, directs the invocation to pod 1
- The server processes the invocation and returns an `Account` object back to the client via I-DBC, which proxifies the `Account` object on-the-fly
- The client then makes an invocation on the `Account` object
- Assuming the connection between the client and the external load balancer is intact, the invocation on the `Account` object will still be directed to pod 1
- I-DBC gets the invocation, recognizes the proxified `Account` object, deproxifies it, and passes it to the CORBA sever

It is an important consideration that connections always use the same pod, because things can break down very quickly. Consider the following hypothetical scenario:

- As in the scenario above, the client receives an `Account` object from the server
- The client makes an invocation on the `Account` object
- The client's ORB decides it will make this invocation on a new connection
- A second connection is established between the client and the external load balancer
- The external load balancer will then direct this invocation via Kubernetes to one of the pods
- If pod 2 is chosen for this invocation:
 - I-DBC gets the invocation but does not recognize the proxified IOR, as it was proxified by the I-DBC instance running in pod 1
 - I-DBC throws an `OBJECT_NOT_EXIST` exception
 - The client invocation fails

Care must be taken if the client makes multiple related invocations. If the invocations are directed to a server running in a Kubernetes pod, then the invocations must occur on the same connection between the client and the external load balancer.

Even if the client makes a single invocation, problems can occur. For example:

- A client makes an invocation to a `Bank` object, implemented by a CORBA server running in Kubernetes - initially this is a `LocateRequest` with the `Bank` object key
- A connection is established between the client and the external load balancer
- The external load balancer, via Kubernetes, directs the invocation to pod 1
- The CORBA server is meant to be contacted indirectly via an IMR - the invocation goes to the IMR, and not to the server
- The IMR in pod 1 returns a `LocateReply` with a `LOCATION_FORWARD` status, and the reference to the actual `Bank` object
- I-DBC in pod 1 proxifies the object reference and returns it to the client
- The client's ORB detects the `LOCATION_FORWARD` status and establishes a second connection to the `Bank` object
- A `LocateRequest` is sent to the `Bank` object
- The external load balancer directs the `LocateRequest` via Kubernetes to pod 2
- I-DBC in pod 2 receives the `LocateRequest`, but does not recognize the `Bank` object reference, as it was proxified by the I-DBC instance running in pod 1
- I-DBC throws an `OBJECT_NOT_EXIST` exception
- The client invocation fails

If you make client invocations to a CORBA server running in Kubernetes, and the expectation is that the same CORBA server will process all client invocations, then you must ensure that all invocations take place over the same connection between the client and the external load balancer.

If you are unsure, you may want to increase the logging level of your CORBA client. The ORB may provide useful logging messages regarding when connections are created and terminated.

In some cases like the IMR scenario, the client has little control over what connections the ORB might make. The documentation for your particular CORBA product may describe an API call or a configuration item that can be used to keep invocations on the same connection.

If in a scenario such as the IMR, there is no available API call or configuration item to keep everything on one connection, an alternative approach would be to re-architect the application. For example, when using Kubernetes with replicas, the external load balancer and Kubernetes are able to find the CORBA servers running in pods. Perhaps an IMR is no longer needed as the external load balancer and Kubernetes are providing similar capabilities.

If there is no way to avoid the application opening multiple connections, one possible fallback is to not use replication and to run the CORBA server in a single pod.

Server Considerations

There are also aspects to consider from the server side before attempting to deploy a CORBA application into Kubernetes.

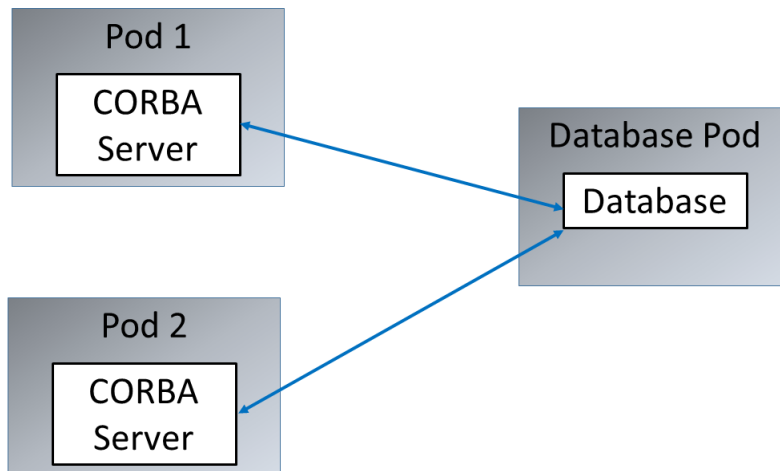
Servers and State

When running a CORBA server inside multiple Kubernetes pods, the issue arises of whether the server is stateless or maintaining state. For example:

- A client makes an invocation to a CORBA server running inside Kubernetes
- The invocation is directed to the server running in pod 1
- The server maintains a count of how many times it was called, and returns '1' to the client
- The client makes a second invocation - it is directed to the server running in pod 2
- This server returns '1' to the client
- Since the client made two invocations, it is expecting a value of '2' - the client is now confused and unsure why the second invocation returned a value of '1'

This is the type of problem that can arise when the server maintains state.

Ideally, the CORBA server will be completely stateless. If the CORBA server must maintain state, then a workable approach would be to maintain that state outside of the server itself. For example, the server in each pod might maintain state in a common database as illustrated in the figure below:



The state of each server is co-ordinated and maintained by a database running in a pod outside the replica set.

Server Proxified IOR

Another consideration is the IOR the client will use to contact the server. A single IOR is used to make invocations to servers spread across multiple pods. In order to achieve this, the proxified IOR in each pod must have the same:

- Hostname or IP address
- Port number
- Object key

When IORs are proxified in each pod, they will be given the hostname or IP address, and port number, from the Kubernetes service that was defined for this application.

The object key becomes the issue. Some ways to ensure the object key is the same across all pods are:

- Use a server that is direct persistent
- If the server is not direct persistent, it can use the fact that I-DBC allows a server to set its object key when proxifying IORs - look at the entry point script for [The Greeter Example](#) for an approach to setting an object key that is common across all pods

Special ORB Facilities

An ORB implementation may provide facilities such as:

- Replication of services or servers
- High availability
- Failover

If your application makes use of such facilities, you may want to consider what Kubernetes provides with replica sets. It may be that you can greatly simplify your application by avoiding specific ORB facilities such as these, and have a simplified server running in Kubernetes, leveraging the high availability facilities that Kubernetes provides.

Consider a Single Replica

If your client or server is such that it cannot be made suitable to run in Kubernetes with replicated pods, then you might still want to consider running just a single pod.

While this approach means you do not have an application that can scale up to meet demand, the Kubernetes liveness and readiness probes help ensure an application is running well and available as much as possible. If a probe detects a problem and restarts a pod, there will be a small window of unavailability, but the pod will eventually be made available again.

The Greeter Example

Introduction

A greeter example is provided to illustrate how a CORBA application can run in Kubernetes. This can be used as a guide to getting your CORBA applications running in Kubernetes. Keep in mind that this is a generalized example, and that your CORBA applications might require more than what this example illustrates.

The example is provided for these CORBA products:

- Orbix 3
- Orbix 6
- VisiBroker

Each product has a C++ and Java version of the example. The example illustrates how to run with secure connections.

The instructions that follow will walk you through building and running the VisiBroker for Java version of the example. Note that for all examples, a `README` file is provided with instructions on how to build and run it.

Prerequisites

Before attempting to run the example, make sure the following prerequisites are met:

- The CORBA Add-on for Cloud, Containers & Virtual Environments is installed at `<installdir>` with all HotFixes applied
- The CORBA product (in the case of this example, VisiBroker) is installed at `<ORBDir>`
- A 64-bit JDK 8 is available to build and run the demo
- A Kubernetes cluster is available for testing
- The cluster has an external load balancer
 - This example will create a service object of type `LoadBalancer`, which requires an external load balancer
 - Most cloud providers make an external load balancer available to a Kubernetes cluster
 - For on-premises Kubernetes clusters, both commercial and open source external load balancers are available. Be sure that your on-premises Kubernetes cluster has one.
 - Some Kubernetes implementations may have little to no support for an external load balancer. For this reason, you may want to avoid implementations such as MiniKube.
- The cluster can access the Docker registry containing the Docker image for the server side of the greeter example
- The `docker` command is available.
- The `kubectl` command is available and is configured to access the Kubernetes cluster.
- Your user ID has permissions to create the following Kubernetes objects:
 - Secrets

- Services
- Deployments
- The required namespace is set.

In the following examples, all Kubernetes objects (secrets, services and deployments) are created in the current namespace. (If you have not actively set a namespace, this will be the `default` namespace.) If you wish the objects to be created in a specific namespace, you need to either set the namespace in the current context, or switch to a context that has the required namespace. Refer to the Kubernetes documentation (see <https://kubernetes.io/docs/home/>) for instructions on how to work with namespaces and contexts.

- Optionally, `perl`

IDL

The greeter example IDL looks as follows:

```
module GreeterDemo
{
  interface Greeter
  {
    string hello(in string from);
  };
};
```

The IDL defines a `hello` operation that takes a string as input, and returns a string to the caller.

Build the Server

The server side of the example is packaged into a Docker image.

Build the base Docker images

Before you can build the example server image, the base layer images must be built:

Linux:

```
cd <installdir>/docker
./buildbaselayers.sh -v centos
```

Windows:

```
cd <installdir>\docker
buildbaselayers.bat -v centos
```

Note that in the commands above, `centos` can be replaced by `ubuntu` depending on which system you are using.

Build the Docker image for the server

The Dockerfile that packages the server into a Docker image looks as follows:

```
## (C) Copyright 2020 Micro Focus or one of its affiliates
## Based on our idbc layer
FROM vb-idbc-layer

## Build current example
WORKDIR $VB_INST_DIR
ENV DEMO_DIR ${VB_INST_DIR}/examples/vbroker/kubernetes/greeter

# construct and install trustpoints
RUN mkdir ${DEMO_DIR}/trustpoints-idbc; \
  # Construct iDBC SSLServer trustpoint - concatenate CA certificates in
  trustpoints directory.
  for cert in ${DEMO_DIR}/trustpoints/*; do cat $cert >> ${DEMO_DIR}/
  trustpoints-idbc/client-ca-list.pem; done; \
  # Construct iDBC SSLClient trustpoint - concatenate CA certificates for
  proxified CORBA Server
  cat ${DEMO_DIR}/identities/frans/cert1 ${DEMO_DIR}/identities/frans/cert2
  > ${DEMO_DIR}/trustpoints-idbc/server-ca-chain.pem; \
  # install iDBC Proxy CA certificate into VisiBroker demo trustpoints
  directory
  cp ${IDBC_PRODUCT_DIR}/idbc/adm/ProxyCACert.pem ${DEMO_DIR}/trustpoints;

RUN . bin/vbroker.sh; \
  cd ${DEMO_DIR}; \
  make java

COPY entrypoint_vb.sh ${DEMO_DIR}
ENTRYPOINT ${DEMO_DIR}/entrypoint_vb.sh
```

The Dockerfile:

- Sets up the trustpoints that allow the example to run with TLS security
- Compiles the server side of the example
- Copies the entrypoint script into the Docker image
- Indicates that the entrypoint script should be run when the Docker container is started

The entrypoint_vb.sh script looks as follows:

```
#!/bin/bash

# (C) Copyright 2020 Micro Focus or one of its affiliates

# Pull in functions
. ~/entrypoint_helper_vb.sh

# Set IDBC security information
#
# Parm 1: Client verify flags
# 0 - Peer certificates are ignored
# 1 - Peers may have a valid certificate
# 3 - Peers must have a valid certificate
#
# Unless an anonymous ciphersuite is used, the server
# "should" present the client with a certificate
```



```

#
#       Configure IDBC to use a value of 1
#
# Parm 2:  Client CA file (PEM format)
#
#       IDBC establishes a connection to the server and
#       requires a CA in order to validate the server
#       certificate
#
#       Configure IDBC with the CA that authenticates the client/server
#       certificates
#
# Parm 3:  Client certificate file (PEM format)
#
#       The server verify flags (see below) indicate that peer certificates
#       "may" be supplied, and IDBC "will" provide a client certificate
#
#       Configure IDBC with the client certificate
#
# Parm 4:  Client private key file (PEM format)
#
#       The server verify flags (see below) indicate that peer certificates
#       "may" be supplied, and IDBC "will" provide a client certificate
#
#       Configure IDBC with the client certificate private key
#
# Parm 5:  Client private key passphrase
#
#       The server verify flags (see below) indicate that peer certificates
#       "may" be supplied, and IDBC "will" provide a client certificate
#
#       Configure IDBC with the private key passphrase
#
# Parm 6:  Client ciphersuite
#
#       Configure IDBC with the client ciphersuite for connections to the
server
#
# Parm 7:  Client transport protocol
#
#       Configure IDBC with the client transport protocol for connections to
the server
#
# Parm 8:  Server verify flags
#       0 - Peer certificates are ignored
#       1 - Peers may have a valid certificate
#       3 - Peers must have a valid certificate
#
#       Configure IDBC to use a value of 1
#
# Parm 9:  Server CA file (PEM format)
#
#       Configure IDBC with the CA that authenticates the server certificate
#
# Parm 10: Server certificate file (PEM format)
#
#       Configure IDBC with the server certificate
#
# Parm 11: Server private key file (PEM format)
#

```

```

#       Configure IDBC with the server certificate private key
#
# Parm 12: Server private key passphrase
#
#       Configure IDBC with the private key passphrase
#
# Parm 13: Server ciphersuite
#
#       Configure IDBC with the server ciphersuite
#
# Parm 14: Server transport protocol
#
#       Configure IDBC with the server transport protocol
#
set_idbc_security_info "1" \
    "${DEMO_DIR}/trustpoints-idbc/server-ca-chain.pem" \
    "" \
    "" \
    "" \
    "TLSv1" \
    "tlsv1" \
    "1" \
    "${DEMO_DIR}/trustpoints-idbc/client-ca-list.pem" \
    "${DEMO_DIR}/identities/frans/cert0" \
    "${DEMO_DIR}/identities/frans/key" \
    "frans" \
    "TLSv1" \
    "tlsv1"

# Register/License Products, Start up, and check IDBC
register_vb
start_idbc
check_idbc

# Prepare VisiBroker Environment
cd $VB_INST_DIR; . bin/vbroker.sh

# Start Server
start_java_server_and_wait_for_ior "${DEMO_DIR}" \
    "Server" \
    "-DORBpropStorage=java_server.properties -
Dvbroker.agent.enableLocator=false" \
    "greeter.ior" \
    "greeter.ior" \
    "GreeterJava"

# Wait for Client Connection
container_ready

```

The `entrypoint_vb.sh` script:

- Pulls in helper functions
- Sets up I-DBC configuration for secure TLS connections
- Sets up licensing for VisiBroker
- Starts I-DBC and checks it is running correctly
- Starts the greeter example Java server. Note that it sets the object key to `GreeterJava`. This ensures that the object key in the proxified IOR is the same across all pods.

- Waits for client invocations

Package the server side of the greeter example into a Docker image as follows:

Linux:

```
cd <installdir>/kubernetes/visibroker/  
greeter_java  
docker build -t vb-greeter:latest .
```

Windows:

```
cd <installdir>\kubernetes\visibroker\  
greeter_java  
docker build -t vb-greeter:latest .
```

Build the Client

The client is meant to run outside Kubernetes. Set the VisiBroker environment and run the `build` command. The `JAVA_HOME` environment variable must point to a 64-bit JDK. JDK 8 works well:

Linux:

```
cd <ORBDir>/bin  
export JAVA_HOME=<YOUR_JDK>  
. ./vbroker.sh  
cd <ORBDir>/examples/vbroker/kubernetes/greeter  
make java
```

Windows:

```
cd <ORBDir>\bin  
set JAVA_HOME=<YOUR_JDK>  
vbroker.bat  
cd <ORBDir>\examples\vbroker\kubernetes\greeter  
vbmake.bat
```

Tag and Push the Server Docker Image to a Registry

The server side of the greeter example is packaged into a Docker image named `vb-greeter:latest`. Most likely, this image is managed by the Docker daemon where the image was built. It is also likely that Kubernetes will not have visibility to this Docker image.

The Docker image needs to be placed into a Docker registry that Kubernetes has access to. The Docker registry you use will depend on your environment. The following is an example where a public registry at "Docker Hub" is used.

Assume the following:

- Your Docker Hub ID is: *mydockerhubid*
- Your Docker Hub public registry is: *my_registry*
- Your Kubernetes cluster can access this registry

Assume that the greeter Docker image at Docker Hub is called:

```
mydockerhubid/my_registry:vb-greeter
```

To get the `vb-greeter:latest` Docker image from your local Docker installation to the Docker Hub registry, two steps are required:

- 1 Tag the image to match the name you chose at Docker Hub
- 2 Push the image to Docker Hub

Run the following to tag image `vb-greeter:latest`:

Linux and Windows:

```
docker tag vb-greeter:latest mydockerhubid/my_registry:vb-greeter
```

Run the following to push the image to the Docker Hub registry:

Linux and Windows:

```
docker push mydockerhubid/my_registry:vb-greeter
```

The Docker image now resides in a Docker registry that the Kubernetes cluster can access.

Note that the above is for illustrative purposes only. You may not want to have your Docker images containing your enterprise applications residing on a public Docker registry. Modify the instructions above to suit your specific circumstances regarding the Docker registry.

Create a Kubernetes Secret Object to Hold Licensing Information

The server side of the greeter example that is packaged into a Docker image requires these licenses:

- A license file to run I-DBC
- A single license or set of license files to run VisiBroker

When you run Docker containers manually, these license files are supplied using a Docker volume. When Docker containers are running inside Kubernetes, these license files need to be put in a Kubernetes *secret* object. The secret is then mounted as a volume when the deployment is created.

To create the Kubernetes secret, use the following commands:

Linux:

```
cd <installdir>/kubernetes/visibroker/greeter_java  
../../bin/create_secret.sh -v
```

Windows:

```
cd <installdir>\kubernetes\visibroker\greeter_java  
..\..\bin\create_secret.bat -v
```

Check that the secret was created correctly by running the `kubectl get secrets` command:

```
kubectl get secrets
```

The output will look similar to:

NAME	TYPE	DATA	AGE
mf-kube-licenses-visi	Opaque	5	18s

Create the Kubernetes Service for the Greeter Example

A Kubernetes service object exposes the greeter example server running in a set of pods. The service will be defined as a *LoadBalancer* service. The service will contain an external address (an IP address or DNS name) and port that a greeter client can use to access the greeter server.

When creating a Kubernetes LoadBalancer service, the port number that is visible from outside Kubernetes is often dynamically assigned. Usually this port number is in the range of 30000-32767.

Several layers of port number indirection then occur across various Kubernetes objects. This indirection poses a difficulty for CORBA applications.

To address this difficulty and remove some of the indirection, the example manually chooses the port number we want the greeter example server to be exposed on, in the range of 30000-32767. Take care to choose a port that is not being used by some other service defined to Kubernetes. Start by getting a list of all Kubernetes services:

Linux and Windows:

```
kubectl get services --all-namespaces
```

This may produce output similar to:

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	56d
kube-system	kube-dns	ClusterIP	10.100.0.10	<none>	53/UDP,53/TCP	56d
nginx	LoadBalancer	ClusterIP	10.99.107.79	<none>	8080:32392/TCP	43h
...						

Look under the `PORT(S)` column for ports in the range 30000-32767. In the sample output above, port 32392 is used by the `nginx` service. So that port is not available for the greeter service.

For illustration purposes, this example uses port 31025.

YAML File Template

To facilitate the creation of the greeter service, a *template* named `vb-greeter_service_template.yaml` file is provided:

```
# (C) Copyright 2020 Micro Focus or one of its affiliates
#
# Define the Kubernetes Service object for the VisiBroker
  Basic Bank Portable Demo (C++).
#
apiVersion: v1
kind: Service
metadata:
  labels:
    run: vb-greeter
    name: vb-greeter
    namespace: default
spec:
  ports:
# Inside the Kubernetes cluster, the service is exposed on
  this port
#
  - port: REPLACE_PORT
```

```

# Outside the Kubernetes cluster, the service is exposed
  on this port
#
  nodePort: REPLACE_PORT
# The pod is listening on this port
#
  targetPort: 3000
  protocol: TCP
  selector:
    run: vb-greeter
  type: LoadBalancer

```

The `.yaml` file defines:

- A service object named `vb-greeter`
- The service type is `LoadBalancer`
- When invocations are directed toward this service, it will run `vb-greeter`, which will be set up when the deployment object is created
- There are three port definitions:
 - **port:** This is the port the service is exposed on inside the Kubernetes cluster. Its value in the template is `REPLACE_PORT`. This value must be replaced by the port number we chose for this service: `31025`
 - **nodePort:** This is the port the service is exposed on outside the Kubernetes cluster. Its value in the template is `REPLACE_PORT`. This value must be replaced by the port number we chose for this service: `31025`
 - **targetPort:** This is the port number that will be opened on the pod to allow traffic in. It is set to `3000`. This value does not need to be changed.

Updating the YAML File Template

The `.yaml` file template has two values that need to be replaced:

- **port:** `REPLACE_PORT`
- **nodePort:** `REPLACE_PORT`

Each `REPLACE_PORT` should be substituted with `31025`, the port chosen for the greeter example. (Again, this port number is for illustrative purposes. You must ensure that a port number that is not currently in use by your Kubernetes cluster is chosen.)

This can be done manually, or by a script provided to make the updating easier.

In either case, do **not** update the template file directly. Copy the `vb-greeter_service_template.yaml` file first to `vb-greeter_service.yaml`.

If updating manually:

- 1 Manually copy `vb-greeter_service_template.yaml` to `vb-greeter_service.yaml`
- 2 Use an editor to substitute all occurrences of `REPLACE_PORT` with `31025`

If `perl` is available, a script can be used to copy the template file and make the substitutions:

Linux:

```
cd <installdir>/kubernetes/visibroker/greeter_java
perl ../../bin/prepare_service.pl vb-greeter 31025
```

Windows:

```
cd <installdir>\kubernetes\visibroker\greeter_java
perl ..\..\bin\prepare_service.pl vb-greeter 31025
```

Create the Service

File `vb-greeter_service.yaml` contains a copy of the template, with the correct port numbers for the service. Use the `kubectl apply` command to create the service:

Linux and Windows:

```
kubectl apply -f vb-greeter_service.yaml
```

Check the Service

Get a list of services using the `kubectl get` command:

Linux and Windows:

```
kubectl get services
```

This may produce output similar to:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	59d
vb-greeter	LoadBalancer	10.100.250.197	a8fa.us-east-2.elb. amazonaws.com	31025:31025/TCP	13s

The `vb-greeter` service should appear under the `NAME` column.

Under the `PORT(S)` column you should see: `31025:31025/TCP` (the port number will depend on the port number you chose.)

Under `EXTERNAL-IP` you should see:

- An IP address
- A DNS name

What you see depends on the external load balancer Kubernetes is using.

Note:

If you see a DNS name, then you should wait for DNS propagation before defining the deployment object for the greeter example. For a cloud provider such as Amazon AWS, a DNS name is used, and it usually takes a few minutes for propagation. Note also that these DNS names are not always pingable.

Create the Kubernetes Deployment for the Greeter Example

A Kubernetes deployment object creates the replica set and pods.

YAML File Template

To facilitate creating the greeter deployment, a template `vb-greeter_deployment_template.yaml` file is provided:

```
# (C) Copyright 2020 Micro Focus or one of its affiliates
#
# Define the Kubernetes Deployment object for the VisiBroker Basic Bank
Portable Demo (C++).
#
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: vb-greeter
    name: vb-greeter
spec:
  replicas: 2
  selector:
    matchLabels:
      run: vb-greeter
  template:
    metadata:
      labels:
        run: vb-greeter
    spec:
      containers:
      - env:
        - name: MF_LOAD_BALANCER_ADDRESS
          value: REPLACE_LOAD_BALANCER_ADDRESS
        - name: MF_KUBERNETES_SERVICE_NAME
          value: "vb-greeter"

      # The port on which IDBC will listen.
      # For secure communication, this needs to be defined as
MF_IDBC_SECURE_PORT.
      # For insecure communication, this needs to be defined as MF_IDBC_PORT.
        - name: REPLACE_IDBC_PORT_NAME
          value: "3000"

        - name: MF_PROXIFIED_IOR_DIR_INTERNAL
          value: /home/corba/proxified_iors
        image: REPLACE_IMAGE_NAME
        imagePullPolicy: Always
        name: vb-greeter
        ports:
        - containerPort: 3000
          protocol: TCP
        volumeMounts:
        - mountPath: /home/corba/volume
          name: secret-volume-for-licenses
          readOnly: true
      # Check the readiness of the pod by:
      # - Ensuring the server IOR has been proxified
```



```

#
  readinessProbe:
    exec:
      command:
        - cat
        - /home/corba/proxified_iors/greeter.ior
      initialDelaySeconds: 30
      periodSeconds: 10
# Check the health of the pod by:
# - Ensuring the I-DBC Proxy is running
# - Ensuring the SPS is running
# - Ensuring the server Java process is running
# Note: using pgrep to check that the server Java process exists
# is very simplistic. A more thorough approach would be for the
# server to have a heartbeat/ping method to return its status.
#
  livenessProbe:
    exec:
      command:
        - /bin/bash
        - -c
        - ${IDBC_PRODUCT_DIR}/idbc/bin/checkproxy.sh && ${IDBC_PRODUCT_DIR}/
sps/bin/checksps.sh && pgrep vbj
      initialDelaySeconds: 30
      periodSeconds: 10
  volumes:
  - name: secret-volume-for-licenses
    secret:
      defaultMode: 420
      secretName: mf-kube-licenses-visi

```

The .yaml file defines:

- A deployment object named `vb-greeter`
- Two replicas will be run when this deployment object is created - essentially, it will start two pods
- The following environment variables are passed into each pod:
 - **MF_LOAD_BALANCER_ADDRESS:** It has a value of `REPLACE_LOAD_BALANCER_ADDRESS`. This must be replaced with the `EXTERNAL-IP` of the service object defined for the greeter example.
 - **MF_KUBERNETES_SERVICE_NAME:** It has a value of `vb-greeter`. Client invocations directed to the `vb-greeter` service will be relayed to one of the pods in this replica set.
 - **REPLACE_IDBC_PORT_NAME:** This environment variable name must be replaced by:
 - `MF_IDBC_SECURE_PORT` when the client connects over a TLS connection
 - `MF_IDBC_PORT` when the client connects over an insecure connection.
 - **MF_PROXIFIED_IOR_DIR_INTERNAL:** It has a value of `/home/corba/proxified_iors`. This value should not be changed.
- The Docker image name containing the greeter example server. It has a value of `REPLACE_IMAGE_NAME`. See [“Tag and Push the Server Docker Image to a Registry”](#) for the image name that must be used here.

- A volume mount for the volume containing the Kubernetes secret object that holds the licensing information. The volume is mounted to the directory `/home/corba/volume` in each pod.
- A [readinessProbe](#). This defines a command that checks for the existence of the greeter example server proxified IOR file `/home/corba/proxified_ior/greeter.ior`. When this file exists, Kubernetes will consider the pod to be in a ready state.
- A [livenessProbe](#). This defines three commands:
 - A command to check if the I-DBC proxy is running
 - A command to check if the I-DBC SPS is running
 - A command to check if the greeter example Java server is running
 The pod is considered healthy when the commands indicate that all these are running. If the commands indicate something is not running, Kubernetes will consider the pod to be unhealthy and will restart it.
- A volume is defined for the Kubernetes `secret` `mf-kube-licenses-visi`. This volume is mounted into each pod.

Updating the YAML File Template

The `.yaml` file template has three values that need to be replaced:

- `value: REPLACE_LOAD_BALANCER_ADDRESS`
- `name: REPLACE_IDBC_PORT_NAME`
- `image: REPLACE_IMAGE_NAME`

The `REPLACE_LOAD_BALANCER_ADDRESS` should be substituted with the `EXTERNAL-IP` value that was assigned to the greeter example service when it was created.

The `REPLACE_IDBC_PORT_NAME` should be substituted with:

- `MF_IDBC_PORT` when the client is using insecure connections
- `MF_IDBC_SECURE_PORT` when the client is using secure connections

For the greeter example, the client is using secure connections, so substitute with `MF_IDBC_SECURE_PORT`.

The `REPLACE_IMAGE_NAME` should be substituted with the Docker image tag name that was pushed to a Docker registry that Kubernetes can access. See [“Tag and Push the Server Docker Image to a Registry”](#).

You can carry out these substitutions manually, or by using the provided perl script `prepare_deployment.pl`.

In either case, do **not** update the template file directly. Copy the `vb-greeter_deployment_template.yaml` file first to `vb-greeter_deployment.yaml`.

If updating manually:

- Manually copy `vb-greeter_deployment_template.yaml` to `vb-greeter_deployment.yaml`
- Use an editor to make the substitutions noted above

If `perl` is available, you can use the `prepare_deployment.pl` script to copy the template and make the substitutions:

Linux:

```
cd <installdir>/kubernetes/visibroker/greeter_java
```

```
perl ../../bin/prepare_deployment.pl vb-greeter
<DOCKER_IMAGE_TAG_NAME> secure
```

Windows:

```
cd <installdir>\kubernetes\visibroker\greeter_java
perl ..\..\bin\prepare_deployment.pl vb-greeter
<DOCKER_IMAGE_TAG_NAME> secure
```

Create the Deployment

The file `vb-greeter_deployment.yaml` contains a copy of the template, with the appropriate substitutions for the deployment. Use the `kubectl apply` command to create the deployment:

Linux and Windows:

```
kubectl apply -f vb-greeter_deployment.yaml
```

This command creates:

- A *deployment* object named `vb-greeter`
- A *replica set* with a prefix name of `vb-greeter`
- A set of pods. Each pod's name will be unique, but will have a prefix of `vb-greeter`

Check the Deployment

Get a list of deployments using the `kubectl get` command:

Linux and Windows:

```
kubectl get deployments
```

This may produce output similar to:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
vb-greeter	0/2	2	0	21s

The `vb-greeter` deployment should appear under the `NAME` column.

The `READY` column may initially report `0/2`. This indicates that 0 of the 2 replicas are ready. Once the pods fully initialize and the `readinessProbe` determines each pod is ready, it should report `2/2`.

Check the Replica Set

Get a list of replica sets using the `kubectl get` command:

Linux and Windows:

```
kubectl get replicaset
```

This may produce output similar to:

NAME	DESIRED	CURRENT	READY	AGE
vb-greeter-694f684fc4	2	2	0	25s

Under the `NAME` column, the replica set name should appear with a name prefixed by `vb-greeter`.

Under the `DESIRED` column, the number of desired replicas is indicated. For the `vb-greeter` deployment, the number of desired replicas is 2.

Under the `CURRENT` column, the number of current replicas is indicated. For the `vb-greeter` deployment, there should be 2 replicas running.

Under the `READY` column, the number of replicas (pods) that have initialized successfully and are ready to process client invocations is listed. Initially this may indicate that 0 replicas are ready. Once all the replicas initialize, the `READY` column should report 2.

Check the Pods

Get a list of pods using the `kubectl get` command:

Linux and Windows:

```
kubectl get pods
```

This may produce output similar to:

NAME	READY	STATUS	RESTARTS	AGE
vb-greeter-694f684fc4-k827j	1/1	Running	0	31m
vb-greeter-694f684fc4-kzwzv	1/1	Running	0	31m

Under the `NAME` column, two pods should appear with their names prefixed by `vb-greeter`.

Under the `READY` column, each pod may initially report 0/1, indicating 0 of 1 containers is currently ready. Once the pod initializes, the value should change to 1/1.

Under the `STATUS` column, the value should indicate `Running`.

Under the `RESTARTS` column, the number should report 0, indicating that the `readinessProbe` and `livenessProbe` have not detected any problem requiring the pod to restart.

If the pods do not come up correctly, see [“Troubleshooting”](#).

Display the Pod Logs

For each pod name, issue a `kubectl logs` command:

Linux and Windows:

```
kubectl logs <pod name>
```

This shows the output generated by the Docker container running in each pod. It should show that I-DBC has started successfully, and the server has been started and that its IOR has been proxified.

Extract the IOR from a Pod

At this point, there should be two pods running. Each pod is running a copy of the greeter example server.

As noted previously, in order to run a CORBA server successfully with replication in Kubernetes, the server's proxified IOR must look identical across each pod. So it does not matter which pod you copy the IOR from.

The server writes the proxified IOR file to this location inside the Docker container:

```
/home/corba/proxified_iors/greeter.ior
```

The client expects the IOR file to exist in the directory:

```
<ORBDir>/examples/vbroker/kubernetes/greeter
```

Run the following commands to copy the proxified IOR out of one of the pods to the appropriate directory:

Linux:

```
cd <ORBDir>/examples/vbroker/kubernetes/greeter
kubect1 get pods
kubect1 cp <POD_NAME>:/home/corba/proxified_ior/greeter.ior ./greeter.ior
```

Windows:

```
cd <ORBDir>\examples\vbroker\kubernetes\greeter
kubect1 get pods
kubect1 cp <POD_NAME>:/home/corba/proxified_ior/greeter.ior .\greeter.ior
```

Note:

The following message from the `copy` command can be ignored:

```
tar: Removing leading '/' from member names
```

Run the Client Outside Kubernetes

Run the client, accessing the server side of the example running inside Kubernetes from the outside world:

Linux:

```
cd <ORBDir>/bin
export JAVA_HOME=<YOUR_JDK>
. ./vbroker.sh
cd <ORBDir>/examples/vbroker/kubernetes/greeter
vbj -Dvbroker.agent.enableLocator=false
-DORBpropStorage=java_client.properties Client
greeter.ior CORBA_client
```

Windows:

```
cd <ORBDir>\bin
export JAVA_HOME=<YOUR_JDK>
vbroker.bat
cd <ORBDir>\examples\vbroker\kubernetes\greeter
vbj -Dvbroker.agent.enableLocator=false
-DORBpropStorage=java_client.properties Client
greeter.ior CORBA_client
```

This may produce client output similar to:

```
Tue Apr 07 05:05:57 EDT 2020 :: Client CORBA_client :: Obtaining
GreeterFactory IOR from greeter.ior
Tue Apr 07 05:05:57 EDT 2020 :: Client CORBA_client :: Sending hello to
greeter from CORBA_client
Tue Apr 07 05:05:58 EDT 2020 :: Client CORBA_client :: Reply: Greeter
707@vb-greeter-694f684fc4-k827j/192.168.95.205 received your hello
```

Run the client multiple times. To see that client invocations are spread across the two pods, get a list of pods, and list the logs for each pod:

```
kubect1 get pods
kubect1 logs <pod 1>
kubect1 logs <pod 2>
```

A `kubectl logs <pod n>` command will display the entire pod log. At the bottom you should see output similar to:

```
...
Tue Apr 07 09:05:58 UTC 2020 :: Greeter 707@vb-greeter-
694f684fc4-k827j/192.168.95.205 :: Hello from CORBA_client
...
```

Troubleshooting

This section lists some things to look out for if you encounter problems running a CORBA application in Kubernetes.

Pods will not start

After creating a deployment and issuing `kubectl get pods`, the pods should be expected to be in a running state, with 1/1 pods ready for each pod. If this does not happen, issue `kubectl describe pod <pod name>`. The output from this command is often helpful in determining why the pods do not start.

Some things to check are:

- Is the Docker image name correct in the deployment `.yaml` file?
- Can the Kubernetes cluster access the Docker registry containing the Docker image?
 - If the registry is private, then Kubernetes may require some definitions that provide the appropriate credentials to access the registry.
- Is the pod able to mount all the volumes?
 - The pod must mount the Kubernetes `secret` volume. Be sure that this secret is created and the deployment correctly mounts it.
- Are the liveness and readiness probes failing? You might need to adjust the `initialDelaySeconds` and `periodSeconds` settings in the deployment YAML file to provide more time for IDBC and the CORBA server to start before the probes are applied.

If pods are never being marked as ready, then the readiness probe command might be incorrect. If pods are being unexpectedly restarted, the liveness probe command might need adjusting.

One approach is to open a shell into the pod (for instance, by using `kubectl exec`) and to run the probe commands manually. If the liveness probe is causing the pod to restart before the investigation is complete, you can edit the deployment `.yaml` file to either extend the liveness probe's `initialDelaySeconds` value (to provide more time) or comment out the `livenessProbe` section of the file to remove the probe entirely until testing is complete. Note that if you do edit the deployment `.yaml` file, you will need to delete the existing deployment and create a new one for the changes to take effect.

The Service has no External IP Address or DNS Name

Your Kubernetes environment must have an external load balancer. If not, then the service object for your application is incorrect, and the client will be unable to invoke on servers running inside pods.

Be aware that if you are testing with MiniKube, it may have limited to no support for an external load balancer. Consider testing in a cloud environment such as Amazon AWS, Microsoft Azure, or Google Cloud. Alternatively, you may have an on-premises Kubernetes cluster with an external load balancer available for development testing.

OBJECT_NOT_EXIST Exceptions

If you are running more than one replica, ensure that the client does not open multiple connections to the external load balancer for a set of related invocations. If the client does open multiple connections, a first invocation might go to pod 1, and a second invocation that depends on the first invocation might go to pod 2, when it really should go to pod 1. Pod 1 might recognize the object key of the second invocation, but pod 2 most likely will not, resulting in the `OBJECT_NOT_EXIST` exception.

I-DBC will not start in the pod

This is often due to a licensing issue. The things to check are:

- The Kubernetes *secret* is created
- The secret was created with a valid license
- The secret is mounted correctly into the Kubernetes deployment

Also, be sure that any I-DBC configuration items set in the Docker entry point script are correct.

The CORBA Application will not start in the pod

This is often due to a licensing issue. The things to check are:

- The Kubernetes *secret* is created
- The secret was created with a valid license
- The secret is mounted correctly into the Kubernetes deployment

The IOR does not proxify correctly in the pod

One thing to watch out for is the external address defined in the service not matching the external address defined in the deployment. If the addresses do not match, delete both the service and the deployment, and recreate them.

Also, in a cloud environment such as Amazon AWS, when a service is defined, it creates a DNS name for the service address. It takes a few minutes for this name to propagate. If an attempt to proxify the IOR is made before the propagation occurs, the proxification may fail. Give the DNS propagation a few minutes after creating the service object before attempting to create the deployment object.

Index

A

Administration Console 8, 50
Amazon AWS 7, 129

C

CentOS 53, 54, 92
Cloud environments 7
Commands
 docker build 54, 55, 63, 67, 77
Common Docker images 53
Configuring the Smart Agent relay 82
Containers 79
CORBA-based application Docker
 image 54

D

Deployment descriptors
 Orbix 6 69
deployTransient 100
diff command 95
Docker 22, 35, 91
 Desktop for Windows 47
 for Windows 47
 Quickstart terminal 47
 Toolbox 47
docker build command 54, 55, 63, 67, 77
Docker images 1, 53, 54, 92, 113
 CORBA-based application 54
 I-DBC 53, 56, 60
 operating system 53, 54
 ORB 53
 Orbix 3 61
 Orbix 6 65
 using Kubernetes 105
 VisiBroker 75
docker run command 92, 100, 106
Dockerfiles 54
 Basic log demo application 70
 for CentOS 54
 for Orbix 3 61
 for Orbix 6 65
 for Ubuntu 55
 for VisiBroker 75
 I-DBC 58

E

entrypoint_common.sh 91, 100
Environment variables
 JAVA_HOME 117
environment variables
 I-DBC 59
 MF_HOST_IP 51

F

Functions
 proxify_ior 100

G

Google Cloud 7, 129
GUI installation 13, 18, 27, 36

I

I-DBC 5, 8, 24, 79, 106, 107, 108, 109, 116, 118
 Docker image 53, 56, 60
I-DBC environment Variables 59
Install I-DBC 60
Installation
 GUI 13, 18, 27, 36
 in Containers 22
 in the Cloud or in Virtual
 Environments 8
 performing a silent installation 41
 prerequisites 11, 24
 silent 39
 SPS Client 43
 SPS client 43
 steps 11, 25
 troubleshooting 6
 uninstallation 6
Installing
 Orbix 3 62
 Orbix 6 66
 VisiBroker 76

J

JAVA_HOME environment variable 117
JRE 91

K

Kubernetes 35, 103
 cluster 105, 112
 context 113
 deploying 105
 deployment object 105
 Docker images 113
 External Load Balancer 104
 greeter example 112
 kubectl apply command 121, 125
 kubectl command 107, 112
 kubectl exec command 128
 kubectl get command 119, 121, 125, 126
 kubectl get secrets command 118
 kubectl logs command 126, 128
 livenessProbe 106
 livenessprobe 128

- load balancer 108, 112
- LoadBalancer service type 106, 119
- objects 104
- pod 104
- readinessProbe 106
- readinessprobe 128
- replica set 105
- secret object 106, 118, 124
- service 106
- YAML files 107

Kubernetes cluster 105

L

Logging 83

M

MF_HOST_IP environment variable 51

Microsoft Azure 7, 129

MiniKube 112

N

Network Address Translation 5, 7

O

Object Oriented Programming 53

Operating system Docker image 53, 54

Oracle VM VirtualBox 47, 48, 51

ORB Docker image 53

Orbix 3 5, 53

- Docker Image 61
- Dockerfile 61
- installing 62

Orbix 6 5, 53

- Deployment descriptors 69
- Docker Image 65
- Dockerfile 65
- installing 66

OSAgent 83

osagent 79

osarelay 79

P

Performing a silent installation 41

perl 113, 124

Ports 83

ports

- published 100

Properties

- Smart Agent Relay 83

Proxify 100

S

Satisfying Smart Agent Requests 86

Security Policy Server 50, 91

- configuration file 91
- diff 95
- new 99
- structure 96

Server SSL Version

- change 93

Server SSL version 91, 97

- configuration item complete name 99

Silent installation 39

Smart Agent 79

- satisfying requests 86

Smart Agent Relay 79

- configuring 82
- logging 83
- ports 83
- properties 83

SPS Client 43

- Configuration 44
- Installation 43

spsclient tool 100

T

TCP 79

Troubleshooting 6

U

Ubuntu 53, 54, 92

UDP 79, 86

Uninstallation 6

V

vbroker.agent.relay.admins 88

vbroker.agent.relay.agentRequestTimeout 88

vbroker.agent.relay.discoverAgentTimeout 88

vbroker.agent.relay.maxFullRequestRetries 88

vbroker.agent.relay.relayRequestTimeout 88

vbroker.agent.relay.timeoutCheckInterval 89

Virtual Environments 7

Virtual machines 10

VisiBroker 5, 54, 112, 116, 118

- Docker Image 75
- Dockerfile 75
- installing 76
- Smart Agent 79
- Smart Agent Relay 79

 - configuring 82
 - logging 83
 - ports 83
 - properties 83

VMWare vCloud 7

VMWare vSphere 7