Application Distribution

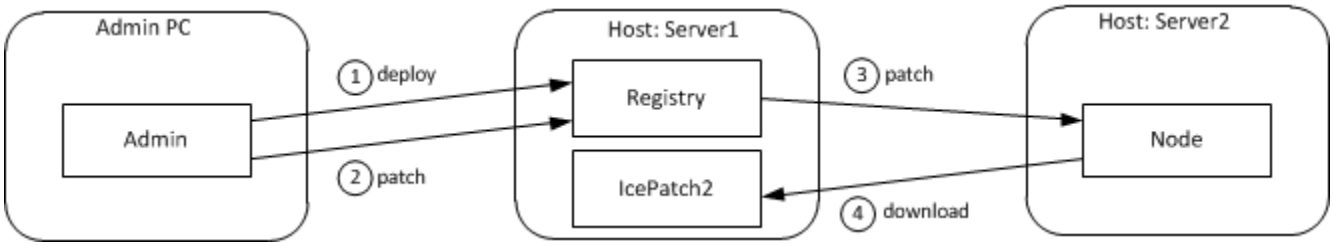
On this page:

Using IcePatch2 to Distribute Applications

In the section so far, "deployment" has meant the creation of descriptors in the registry. A broader definition involves a number of other tasks:

- Writing IceGrid configuration files and preparing data directories on each computer
- Installing the IceGrid binaries and dependent libraries on each computer
- Starting the registry and/or node on each computer, and possibly configuring the systems to launch them automatically
- Distributing your server executables, dependent libraries and supporting files to the appropriate nodes.

The first three tasks are the responsibility of the system administrator, but IceGrid can help with the fourth. Using an IcePatch2 server, you can configure the nodes to download servers automatically and patch them at any time. The illustration below shows the interactions of the components:



Overview of application distribution.

As you can see, deploying an IceGrid application has greater significance when IcePatch2 is also involved. After deployment, the administrative tool initiates a patch, causing the registry to notify all active nodes that are configured for application distribution to begin the patching process. Since each IceGrid node is an IcePatch2 client, the node performs the patch just like any IcePatch2 client: it downloads everything if no local copy of the distribution exists, otherwise it does an incremental patch in which it downloads only new files and those whose signatures have changed.

The benefits of this feature are clear:

- The distribution files are maintained in a central location
- Updating a distribution on all of the nodes is a simple matter of preparing the master distribution and letting IceGrid do the rest
- Manually transferring executables and supporting files to each computer is avoided, along with the mistakes that manual intervention sometimes introduces.

Deploying an IcePatch2 Server

If you plan to use IceGrid's distribution capabilities, we generally recommend deploying an IcePatch2 server along with your application. Doing so gives you the same benefits as any other IceGrid server, including on-demand activation and remote administration. We will only use one server in our sample application, but you might consider replicating a number of IcePatch2 servers in order to balance the patching load for large distributions.

Patching Considerations

Deploying an IcePatch2 server with your application presents a chicken-and-egg dilemma: how do the nodes download their distributions if the IcePatch2 server is included in the deployment? To answer this question, we need to learn more about IceGrid's behavior.

Deploying and patching are treated as two separate steps: first you deploy the application, then you initiate the patching process. The `icegridadmin` utility combines these steps into one command (`application add`), but also provides an option to disable the patching step if so desired.

Let's consider the state of the application after deployment but before patching: we have described the servers that run on each node, including file system-dependent attributes such as the pathnames of their executables and default working directories. If these pathnames refer to directories in the distribution, and the distribution has not yet been downloaded to that node, then clearly we cannot attempt to use those servers until patching has completed. Similarly, we cannot deploy an IcePatch2 server whose executable resides in the distribution to be downloaded.

We are ignoring the case where a temporary IcePatch2 server is used to bootstrap other IcePatch2 servers.

For these reasons, we assume that the IcePatch2 server and supporting libraries are distributed by the system administrator along with the IceGrid registry and nodes to the appropriate computers. The server should be configured for on-demand activation so that its node starts it automatically when patching begins. If the server is configured for manual activation, you must start it prior to patching.

IcePatch2 Server Template

The Ice distribution includes an IcePatch2 server template that simplifies the inclusion of IcePatch2 in your application. The relevant portion from the file `config/templates.xml` is shown below:

XML

```
<server-template id="IcePatch2">
    <parameter name="instance-name" default="${application}.IcePatch2"/>
    <parameter name="endpoints" default="default"/>
    <parameter name="directory"/>

    <server id="${instance-name}" exe="icepatch2server"
        application-distrib="false" activation="on-demand">
        <adapter name="IcePatch2" endpoints="${endpoints}">
            <object identity="${instance-name}/server" type="::IcePatch2::FileServer"/>
        </adapter>
        <adapter name="IcePatch2.Admin" id="" endpoints="tcp -h 127.0.0.1"/>
        <property name="IcePatch2.InstanceName" value="${instance-name}"/>
        <property name="IcePatch2.Directory" value="${directory}"/>
    </server>
</server-template>
```

Notice that the server's pathname is `icepatch2server`, meaning the program must be present in the node's executable search path. The only mandatory parameter is `directory`, which specifies the server's data directory and becomes the value of the `IcePatch2.Directory` property. The value of the `instance-name` parameter is used as the server's identifier when the template is instantiated; its default value includes the name of the application in which the template is used. This identifier also affects the identities of the two well-known objects declared by the server.

Consider the following sample application:

XML

```
<icegrid>
    <application name="PatchDemo">
        <node name="Node">
            <server-instance template="IcePatch2" directory="/opt/icepatch2/data"/>
            ...
        </node>
    </application>
</icegrid>
```

Instantiating the `IcePatch2` template creates a server identified as `PatchDemo.IcePatch2` (as determined by the default value for the `instance-name` parameter). The well-known objects use this value as the category in their identities, such as `PatchDemo.IcePatch2/server`.

In order to refer to the `IcePatch2` template in your application, you must have already configured the registry to use the `config/templates.xml` file as your default templates, or copied the template into the XML file describing your application.

Adding Distribution to a Deployment

A distribution descriptor provides the details that a node requires in order to download the necessary files. Specifically, the descriptor supplies the proxy of the IcePatch2 server and the names of the subdirectories comprising the distribution, all of which are optional. If the descriptor does not define the proxy, the following default value is used instead:

```
${application}.IcePatch2/server
```

You may recall that this value matches the default identity configured by the IcePatch2 server template described above. Also notice that this is an indirect proxy, implying that the IcePatch2 server was deployed with the application and can be started on-demand if necessary.

If the descriptor does not select any subdirectories, the node downloads the entire contents of the IcePatch2 data directory.

In XML, a descriptor having the default behavior as described above can be written as shown below:

XML

```
<distrib/>
```

To specify a proxy, use the `icepatch` attribute:

XML

```
<distrib icepatch="PatchDemo.IcePatch2/server"/>
```

Finally, select subdirectories using a nested element:

XML

```
<distrib>
    <directory>dir1</directory>
    <directory>dir2/subdir</directory>
</distrib>
```

By including only certain subdirectories in a distribution, you are minimizing the time and effort required to download and patch each node. For example, each node in a heterogeneous network might download a platform-specific subdirectory and another subdirectory containing files common to all platforms.

Distributing Applications and Servers

A distribution descriptor can be used in two contexts: within an application, and within a server. When the descriptor appears at the application level, it means every node in the application downloads that distribution. This is useful for distributing files required by all of the nodes on which servers are deployed, especially in a grid of homogeneous computers where it would be tedious to repeat the same distribution information in each server descriptor. Here is a simple XML example:

XML

```
<icegrid>
    <application name="PatchDemo">
        <distrib>
            <directory>Common</directory>
        </distrib>
        ...
    </application>
</icegrid>
```

At the server level, a distribution descriptor downloads the specified directories for the private use of the server:

XML

```
<icegrid>
    <application name="PatchDemo">
        <distrib>
            <directory>Common</directory>
        </distrib>
        <node name="Node">
            <server id="SimpleServer" ...>
                <distrib>
                    <directory>ServerFiles</directory>
                </distrib>
            </server>
        </node>
    </application>
</icegrid>
```

When a distribution descriptor is defined at both the application and server levels, as shown in the previous example, IceGrid assumes that a dependency relationship exists between the two unless the server is configured otherwise. IceGrid checks this dependency before patching a server; if the server is dependent on the application's distribution, IceGrid patches the application's distribution first, and then proceeds to patch the server's. You can disable this dependency by modifying the server's descriptor:

XML

```
<icegrid>
    <application name="PatchDemo">
        <distrib>
            <directory>Common</directory>
        </distrib>
        <node name="Node">
            <server id="SimpleServer" application-distrib="false" ...>
                <distrib>
                    <directory>ServerFiles</directory>
                </distrib>
            </server>
        </node>
    </application>
</icegrid>
```

Setting the `application-distrib` attribute to `false` informs IceGrid to consider the two distributions independent of one another.

Server Integrity during Distribution

Before an IceGrid node begins patching a distribution, it ensures that all relevant servers are shut down and prevents them from reactivating until patching completes. For example, the node disables all of the servers whose descriptors declare a dependency on the application distribution.

Distribution Descriptor Variables

The node stores application and server distributions in its data directory. The pathnames of the distributions are represented by reserved variables that you can use in your descriptors:

- `application.distrib`
  This variable can be used within server descriptors to refer to the top-level directory of the application distribution.
- `server.distrib`
  The value of this variable is the top-level directory of a server distribution. It can be used only within a server descriptor that has a distribution.

The XML example shown below illustrates the use of these variables:

XML

```
<icegrid>
    <application name="PatchDemo">
        <distrib>
            <directory>Common</directory>
        </distrib>
        <node name="Node">
            <server id="Server1" exe="${application.distrib}/Common/Bin/Server1" ...>
            </server>
            <server id="Server2" exe="${server.distrib}/Server2Files/Bin/Server2" ...>
                <option>-d</option>
                <option>${server.distrib}/Server2Files</option>
                <distrib>
                    <directory>Server2Files</directory>
                </distrib>
            </server>
        </node>
    </application>
</icegrid>
```

Notice that the descriptor for `Server2` supplies the server's distribution directory as command-line options.

Using Distribution in the Ripper Application

Adding an application distribution to our ripper example requires two minor changes to our descriptors:

XML

```
<icegrid>
    <application name="Ripper">
        <replica-group id="EncoderAdapters">
            <load-balancing type="adaptive"/>
            <object identity="EncoderFactory" type="::Ripper::MP3EncoderFactory"/>
        </replica-group>
        <server-template id="EncoderServerTemplate">
            <parameter name="index"/>
            <parameter name="exepath" default="/opt/ripper/bin/server"/>
            <server id="EncoderServer${index}" exe="${exepath}" activation="on-demand">
                <adapter name="EncoderAdapter" replica-group="EncoderAdapters"
                    endpoints="tcp"/>
            </server>
        </server-template>
        <distrib/>
        <node name="Node1">
            <server-instance template="EncoderServerTemplate" index="1"/>
            <server-instance template="IcePatch2" directory="/opt/ripper/icepatch"/>
        </node>
        <node name="Node2">
            <server-instance template="EncoderServerTemplate" index="2"/>
        </node>
    </application>
</icegrid>
```

An application distribution is sufficient for this example because we are deploying the same server on each node. We have also deployed an IcePatch2 server on `Node1` using the template.

See Also

- IcePatch2
- IceGrid Administrative Utilities