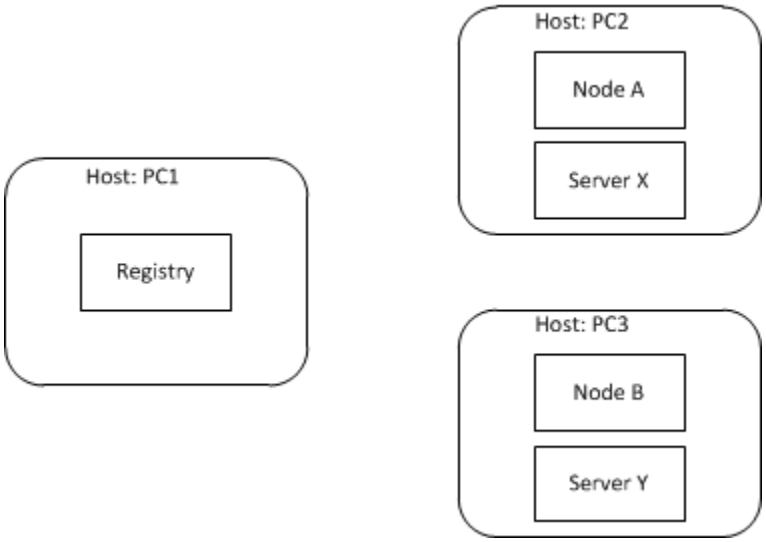
IceGrid Architecture

An IceGrid domain consists of a *registry* and any number of *nodes*. Together, the registry and nodes cooperate to manage the information and server processes that comprise *applications*. Each application assigns servers to particular nodes. The registry maintains a persistent record of this information, while the nodes are responsible for starting and monitoring their assigned server processes. In a typical configuration, one node runs on each computer that hosts Ice servers. The registry does not consume much processor time, so it commonly runs on the same computer as a node; in fact, the registry and a node can run in the same process if desired. If fault tolerance is desired, the registry supports replication using a master-slave design.

On this page:

Architecture of a Simple IceGrid Application

As an example, this illustration shows a very simple IceGrid application running on a network of three computers. The IceGrid registry is the only process of interest on host PC1, while IceGrid nodes are running on the hosts PC2 and PC3. In this sample application, one server has been assigned to each node.



Simple IceGrid application.

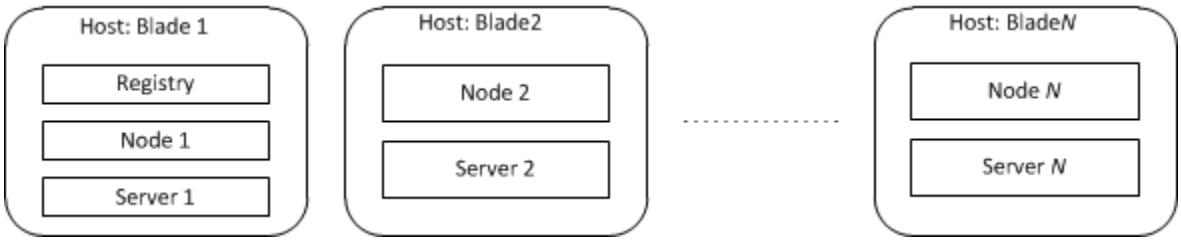
From a client application's perspective, the primary responsibility of the registry is to resolve indirect proxies as an Ice location service. As such, this contribution is largely transparent: when a client first attempts to use an indirect proxy, the Ice run time in the client contacts the registry to convert the proxy's symbolic information into endpoints that allow the client to establish a connection.

Although the registry might sound like nothing more than a simple lookup table, reality is quite different. For example, behind the scenes, a locate request might prompt a node to start the target server automatically, or the registry might select appropriate endpoints based on load statistics from each computer.

This also illustrates the benefits of indirect proxies: the location service can provide a great deal of functionality without any special action by the client and, unlike with direct proxies, the client does not need advance knowledge of the address and port of a server. The extra level of indirection adds some latency to the client's first use of a proxy; however, all subsequent interactions occur directly between client and server, so the cost is negligible. Furthermore, indirection allows servers to migrate to different computers without the need to update proxies held by clients.

Server Replication with IceGrid

IceGrid's flexibility allows an endless variety of configurations. For example, suppose we have a grid network and want to replicate a server on each blade, as shown below:



Replicated server on grid network.

Replication in Ice is based on object adapters, not servers. Any object adapter in any server could participate in replication, but it is far more likely that all of the replicated object adapters are created by instances of the same server executable that is running on each computer. We are using this configuration in the example shown above, but IceGrid requires each server to have a unique name. `Server 1` and `Server 2` are our unique names for the same executable.

The binding process works somewhat differently when replication is involved, since the registry now has multiple object adapters to choose from. The description of the IceGrid application drives the registry's decision about which object adapter (or object adapters) to use. For example, the registry could consider the system load of each computer (as periodically reported by the nodes) and return the endpoints of the object adapter on the computer with the lowest load. It is also possible for the registry to combine the endpoints of several object adapters, in which case the Ice run time in the client would select the endpoint for the initial connection attempt.

Deploying an IceGrid Application

In IceGrid, *deployment* is the process of describing an application to the registry. This description includes the following information:

- Replica groups
  A *replica group* is the term for a collection of replicated object adapters. An application can create any number of replica groups. Each group requires a unique identifier.
- Nodes
  An application must assign its servers to one or more nodes.
- Servers
  A server's description includes a unique name and the path to its executable. It also lists the object adapters it creates.
- Object adapters
  Information about an object adapter includes its endpoints and any well-known objects it advertises. If the object adapter is a member of a replica group, it must also supply that group's identifier.
- Objects
  A *well-known object* is one that is known solely by its identity. The registry maintains a global list of such objects for use during locate requests.

IceGrid uses the term *descriptor* to refer to the description of an application and its components; deploying an application involves creating its descriptors in the registry. The are several ways to accomplish this:

- You can use a command-line tool that reads a file containing an XML representation of the descriptors.
- You can create descriptors interactively with the graphical administration tool.
- You can create descriptors programmatically via IceGrid's administrative interface.

The registry server must be running in order to deploy an application, but it is not necessary for nodes to be active. Nodes that are started after deployment automatically retrieve the information they need from the registry. Once deployed, you can update the application at any time.

See Also

- Locators
- Connection Establishment
- Object Adapters
- Object Adapter Replication
- Well-Known Objects