

Introduction

Pramati Server instances can be configured manually by modifying the configuration XML files. Server instances of type J2EE, Message Server and Cluster can be manually created with simple configuration in the XML files. This section describes the structure of a Server instance and discusses the procedure to create different types of Server instances in Standalone and Cluster scenarios.

Topics in this section are divided into:

- 1 Overview of Pramati Server instances and Pramati Server Cluster
- 2 Node directory structure and configuration with reference to the XML files.
- 3 Creating Server instances and Clusters.
- 4 Pramati Server architecture and the functional blocks of the Pramati Services Framework.

Understanding Pramati Server instances

An instance of the application server, which some vendor term 'server node', can serve either as a Standalone server or a cluster server node. Pramati Server instance can either be a Standalone Server or Server Cluster Node.

Cluster is a logical grouping of two or more server instances running on a single machine or multiple machines and functioning as a single server. All servers in a cluster are transparent to the user. Pramati Server leverages this capability to provide scalability and high availability to applications with the ability to handle high workloads and reduce application downtimes due to failovers.

When the server starts, an instance of the server is spawned as a separate process with its own JVM. Attributes of the instance—sizing, runtime behaviour and performance—is defined by the configuration of the server and is specified in a set of server configuration files (XMLs).

Key terminology

The following table provides a brief description for the various terms used in this document.

Table 1: Terminology

| Term | Description |
|-------------------------|---|
| Installation | Pramati Server product Installation |
| Install-Dir | The directory under which Pramati Server is installed |
| Nodes Directory | The directory under which all configured server instances exist |
| Server | The installed server with default settings |
| Server Instance | An instance of a Pramati Server- of any type (J2EE, Web cluster node, EJB cluster node, Message Server) will normally run in one VM. |
| Standalone Server | An instance that is a self contained J2EE Server running in a single VM |
| Instance Directory | The directory under the <i>Nodes Directory</i> that holds the configuration and other files/ information for a Server Instance |
| Message Server | An instance which functions as a Standalone Message Server |
| Embedded Message Server | Full featured Message Server that runs <i>inside</i> a Standalone Server instance |
| JMS Adapter | JMS Adapter is setup in a J2EE Server instance determining the connectivity to a Message Server (Embedded or external Standalone Message Server or HA-Message Server) |
| Cluster | A logical server comprising a group of underlying nodes forming a single entity that internally works with a collection of nodes. |
| Cluster Node | Server instance that serves as a node of a cluster |
| J2EE Cluster | A cluster with both EJB and WEB Nodes |
| J2EE Node | A Server instance that serves as a node serving both Web and EJB requests. Can exist as a part of a J2EE Cluster |
| BOTH Node | Same as J2EE Node |
| Web Cluster | A cluster of Web Nodes |
| Web Node | A Server instance that serves as a node serving Web requests alone. Can either exist as part of a J2EE Cluster or a Web cluster |
| EJB Cluster | A cluster of EJB Nodes |
| EJB node | Server instance that serves as a node serving EJB requests alone. Can either exist as part of a J2EE Cluster or a Web cluster |
| Message Server Cluster | A cluster of Message Server Nodes |
| HA Message Server | Message Server Cluster that only supports failover, and not LoadBalancing (as of 3.5, Message Server Cluster is only HA Message Server) |
| Message Server Node | A Server instance that serves as a node serving Message Server requests alone |
| Load Balancer | A request dispatcher for managing requests |

Table 1: Terminology

| Term | Description |
|---------------|---|
| Configuration | Modifying the default settings for the server |

Pramati Server configuration files

Configuration files required for setting up Pramati Server are located in the /templates directory under the installation root. These files, with their descriptions, are listed in the following table.

Table 2: Pramati Server configuration files

| Configuration File | Description |
|---------------------|--|
| server-config.xml | The main configuration file where all the details related to services bootstrapping is stored. When you create a node manually, make sure that you copy this file in the config directory from the template. |
| resource-config.xml | All the resource related information like Connection Factory, Data Source properties, Data Source Cluster, Mail Resource and Message Server Resource are stored here. |
| web-config.xml | Necessary configuration information for the Web Container Service is stored here. |
| deploy-config.xml | Contains configuration information for the deploy service. |
| logging-config.xml | Logging parameters like rotation type, max file size and rotation time are configured here. |
| security-config.xml | Contains information about security providers, security policy, certification manager and realm. |
| web-cache.xml | Contains dynamic cache configuration. Dynamic Cache service boosts the network throughput and access speed in the web container of Pramati Server. |
| cluster-config.xml | Stores cluster related information including configuration details for individual nodes. |
| jms-config.xml | Information related to Message Server Queues and Topics are stored here. |

Types of Pramati Server instances that can be configured

There are three basic types of Pramati Server configurations:

- J2EE (serves EJB and web applications)
- Message Server (serves pure Message Server applications)
- Load Balancer (acts as the Pramati Request Dispatcher)

These are set up in the configuration XML, `server-config.xml`. The default instance is J2EE. In a J2EE configuration, the Server can be set up as a Standalone Server or a Server Cluster Node. In a Message Server configuration, the Server can be set up as a Standalone Server or a High Availability Node.

Note: In the J2EE configuration, the Server can run Message Server in an embedded mode.

From these three basic configuration types, the following Server instances can be set up:

Standalone Server configurations derived from an instance

- Standalone Server
- Standalone Server with embedded Message Server
- Standalone Message Server

The most basic type of J2EE Server instance is the Standalone Server. It exists on a single machine, is not part of any cluster, and acts as an independent server. The Standalone Server is the default server type and can be configured to run embedded Message Server.

Note: By default embedded Message Server is checked as enabled in a J2EE server instance.

Server Cluster Node configurations derived from an instance

- J2EE Node (termed BOTH Node)
- Web Node
- HA Message Server Node
- EJB Node
- Load Balancer Node

The Server Cluster J2EE Node has both Web and EJB services enabled in `server-config.xml` file. A node can be configured to as a Standalone Message Server by disabling other services like EJB and Web—as long as there are no inter-service dependencies. The type of server instance is determined based on the specific services that are enabled through the Pramati Services Framework. Details are discussed in the following sections.

Structure of Nodes Directory

Creating a Server instance (in the node directory) makes a directory each for Standalone Server and Server Cluster Node.

Each configured server instance, has an unique name by which it is identified in a cluster. Instances can be created and configured manually without the aid of the Console or Shell.

While creating server instances (in the node directory), there will be one DIR for each server instance (in case of Standalone servers) and one for each cluster-node in case of Cluster. The type of server instance is determined based on the specific services enabled. Details are discussed in the following sections.

Install Directory Structure

/server

-bin

(Server startup scripts are stored here)

-lib

(External and internal libraries used by the server are stored here)

-templates

(This directory contains various configuration XMLs and their DTDs along with the cloudscape DB property file)

-nodes

(Server Instances Directory containing node specific configuration information)

Server Instance Directory Structure*-<node-name>*

(A unique name which identifies the node in a cluster)

-config

(List of all configuration XML files which influences the runtime behavior of the node.)

-logs

(Stack traces and log messages for the node are stored here)

-archives

(This directory holds all extracted application archives)

-temp

(A temporary directory)

Procedure for creating Pramati Server Instances

Server instances are created in the ‘nodes’ directory of a Pramati Server installation. There will be one DIR for each server instance (in case of Stanalone servers) and one for each cluster-node in case of Cluster. The type of server instance is determined based on the specific servcies enabled. Details are discussed in the following sections.

The common steps involved in configuring a Server instance, of any type, are:

- 1 Under the nodes directory, create a sub-directory (henceforth referred to as *instance-directory*) for the required instance.
- 2 Name of the sub directory will be the name of the instance.
- 3 Create the following sub directory under this instance’s directory
 - *config*
 - the other directories described in the ‘Structure of Nodes Directory’ above will be automatically created after the instance starts up.
- 4 Depending on the type of instance needed, copy the required config files from the <install_dir>/server/templates to the configuration directory created above. The specific files needed are listed in the Table 3 (*Configuration files needed for Server Instances*) provided in the following section.

- 5 Edit the required configuration XML to setup the information as described in the following sections, relevant to the type of INstance being setup.
- 6 The instance can be started from the command line using the *-node* option, after running `setup.bat /sh:`

```
com.pramati.Server -node <node name>
```

Configuration files:

The configuration files determine the type of Server instance and the behaviour of the server instance. For each server instance type a specific set of configuration files are required. The table below lists the config files required for each Instance type. These will need to be copied from templates directory into the <instance>/config directory.

Configuration File Dependency

Table 3: Configuration files needed for Server Instances

| Configuration File | Standalone Server | J2EE (BOTH) Cluster Node | Web Cluster Node | Standalone Message Server | HA Message Server Node | EJB Cluster Node | Load Balancer |
|----------------------------|------------------------------------|--------------------------|------------------|---------------------------|------------------------|------------------|---------------|
| server-config.xml | Required | Required | Required | Required | Required | Required | Required |
| resource-config.xml | Required | Required | Can Have | Required | Required | Can Have | - |
| web-config.xml | Required | Required | Required | - | - | - | Required |
| deploy-config.xml | Required | Required | Required | - | - | Required | - |
| logging-config.xml | Can Have | Can Have | Can Have | Can Have | Can Have | Can Have | Can Have |
| security-config.xml | Required | Required | Required | Required | Required | Required | Required |
| system-security.xml | Required | Required | Required | Required | Required | Required | Required |
| web-cache.xml | Required | Required | Required | - | - | - | Required |
| cluster-config.xml | - | Required | Required | - | Required | Required | Needed, if HA |
| jms-config.xml | Needed for embedded Message Server | - | - | Required | Required | - | - |

Table 3: Configuration files needed for Server Instances

| Configuration File | Standalone Server | J2EE (BOTH) Cluster Node | Web Cluster Node | Standalone Message Server | HA Message Server Node | EJB Cluster Node | Load Balancer |
|--------------------|-------------------|--------------------------|------------------|---------------------------|------------------------|------------------|---------------|
| web-lbconfig.xml | - | - | - | - | - | - | Required |

Note: In most cases, the defaults used in the configuration file templates will be sufficient to start an instance.

Using the NodeCreator

The NodeCreator is a utility for creating or removing nodes and clusters. You can configure and use the NodeCreator with both the command line and XMLs.

Using the NodeCreator with the Command line

To run the NodeCreator:

Run NodeCreator.bat (for Windows) or NodeCreator.sh (for Unix) located at <install_dir>/server/bin/.

Execute `java com.pramati.NodeCreator` class by passing the following values:

Table 4: Operations that can be performed using the Node Creator class

| Operation | Description |
|--------------------|--|
| -createj2ee | To create a Standalone J2EE node |
| -createj2eecluster | To create multiple J2EE nodes in a cluster |
| -createjms | To create a Message Server node |
| -createjmscluster | To create multiple Message Server nodes in a cluster |

The various values (given as {parametername value} format) required to perform the above operations are:

Table 5: Values for the Node Creator class

| Value | Description |
|--------------|--|
| -name | Name(s) of the nodes. In case of a cluster, the names are separated by a comma ',' |
| -ip | The IP of the machine on which this node is being created. The default value is local host. In case of a cluster, the IPs are separated by a comma ',' |
| -namingport | Naming port(s) of the nodes. In case of a cluster, the ports are separated by a comma ',' |
| -httpport | HTTP port(s) of the nodes. In case of a cluster, the ports are separated by a comma ',' |
| -clustername | In case of a cluster, the name of the cluster. |

You can also specify the System Property 'install.root', if the program is not being run from the Server root.

Example for creating a J2EE Standalone node

```
java -Dinstall.root=. com.pramati.NodeCreator -createj2ee -name my
node1 -namingport 9191 -httpport 8181
```

Example for creating a J2EE cluster

```
java -Dinstall.root=. com.pramati.NodeCreator -createj2eecluster -name my
node1,mynode2 -namingport 9191,9292 -httpport 8181,8282 -clustername myclus-
ter
```

The help snippet that shows up on the command shell is as follows:

```
Usage: java com.pramati.NodeCreator [operation] [option value] [flag]
```

Using the NodeCreator with XMLs

You can create a node or cluster using the config-topology-standalone-template.xml and config-topology-cluster-template.xml located at <install_dir>/server/templates/.

Using these XML templates, you can convert existing node types, create nodes for a cluster, and configure persistence properties for the database, EJB or Web.

Converting a node type is useful if you want to convert an existing, say Standalone J2EE Server to an EJB cluster type. To do this, open the config-topology-cluster-template.xml:

```
<configuration-topology>
  <!-- Valid Types - a J2EE (EJB/BOTH/WEB) Cluster or a JMS Cluster-->
  <!-- SAMPLE CONFIGURATION, change node definitions as required -->
  <cluster name="MyCluster" type="J2EE">
    <convert-nodes>
      <!-- These are nodes for conversion, the server "J2EESAS1" of the given ip in
      the <source-node> tag would be converted to the cluster node type given in the
      node definition. The target nodenames can be different in which case the
      directories will be copied.-->
      <node name="EJB1" type="EJB1-cluster" naming-port="9000" http-
      port="8000">
        <source-node ip="localhost" name="J2EESAS1" />
      </node>
    </convert-nodes>
  </cluster>
</configuration-topology>
```

Table 6: Description of attributes for <convert-nodes> tag

| Attributes | Description |
|------------|---|
| node name | Name(s) of the target node. That is the node that is to be created. |

Table 6: Description of attributes for <convert-nodes> tag

| Attributes | Description |
|----------------|--|
| type | The type of the target node that is to be created. |
| naming-port | Naming port(s) of the target node. |
| http-port | HTTP port(s) of the target node. |
| source-node ip | The IP of the source machine on which this node is started. The default value is local host. |
| name | Name of the source node. |

If you wish to create a J2EE Cluster, locate the <create-nodes> tag in the config-topology-cluster-template.xml:

```

<create-nodes>
  <node name="nodename1" type="both">
    <ip>localhost</ip>
    <naming-port>9292</naming-port>
    <http-port> 8282</http-port>
  </node>
  <node name="nodename2" type="both">
    <ip>localhost</ip>
    <naming-port>9293</naming-port>
    <http-port> 8283</http-port>
  </node>
</create-nodes>

```

Table 7: Description of attributes for <create-nodes> tag to create a cluster

| Attributes | Description |
|-------------|---|
| node name | Name of the cluster node. |
| type | The type of the cluster - EJB, Web, or BOTH. |
| ip | The IP of the machine on which this node is being created. The default value is local host. |
| naming-port | Naming port of the node. |
| http-port | HTTP port of the node. |

To define the properties for persistence, modify the <configuration-persistence> tag in the config-topology-cluster-template.xml:

```

<!-- use "db" for DB persistence -->
<configuration-persistence type="file">
  <property name="driver" value=""/>
  <property name="url" value=""/>
  <property name="user" value=""/>

```

```

        <property name="password" value=""/>
        <property name="tablename" value=""/>
    </configuration-persistence>
    <!--the default type here is "in-memory", use "db" for DB persistence -->
    <ejb-cluster-persistence >
        <property name="driver" value=""/>
        <property name="url" value=""/>
        <property name="user" value=""/>
        <property name="password" value=""/>
        <property name="table-name" value=""/>
    </ejb-cluster-persistence>
    <!--the default type here is "in-memory", use "db" for DB persistence -->
    <web-cluster-persistence >
        <property name="url" value=""/>
        <property name="url" value=""/>
        <property name="user" value=""/>
        <property name="password" value=""/>
        <property name="table-name" value=""/>
    </web-cluster-persistence>
    <!-- the upload-config allows pre-configuring the created cluster.e.g.The
    configuration for nodename1 would be uploaded
    to the DB and this configuration would be synced up to the other nodes.-
    ->
    <upload-config-source-node name="nodename1" />
</cluster>
</configuration-topology>

```

Custom node configuration

You can use the `-configuration` flag with the `nodecreator` utility and pass the name of a configuration file as a parameter. For example: `com.pramati.NodeCreator -configuration config.xml`

The `config.xml` file can have the following structure:

```

<server-configuration>
  <configuration service-name="ResourceService">
    <datasource name="myDS" connection-factory="myCF" transaction-participa-
    tion="true" description="No Description">
      <pool-properties min-pool-size="10" max-pool-size="20" initial-pool-
      size="2" refresh-interval-seconds="" connection-request-timeout-sec-
      onds="" idle-timeout-seconds="" cache-size="" />
      <connection-validation sql="" class="" />
    </datasource>
  </configuration>
</server-configuration>

```

```

<connection-factory name="myCF" description="" classname="COM.cloud-
scape.core.RmiJdbcDriver" url="jdbc:cloud-
scape:rmi:D:\CTS\j2eects\db_cs40\DB1" authorized-by="Container">
  <login-parameters user="" password="" mask-password="false"/>
</connection-factory>

<-jms-adapter name="myAdapter" description="No Description" interface-
class="com.pramati.jms.client.JMSProviderImpl">
  <properties>
    <property name="com.pramati.naming.cacheLookups" value="false"/>
    <property name="java.naming.factory.initial" value="com.pramati.nam-
ing.PramatiLocalContextFactory"/>
    <property name="java.naming.provider.url" value="rmi://localhost:9090"/>
    <property name="com.pramati.force.standalone.ctx" value="false"/>
  </properties>
</jms-adapter-->
</configuration>
</server-configuration>

```

As an example, the above will add the mentioned resources in the newly created nodes - the relevant entries will be added to the `resource-config.xml` files.

The supported resources for this tag are:

1. JDBC Connection Factories
2. Datasources
3. JMS Adapters

Creating a Standalone Server

General Instructions:

- Under the nodes directory, create a sub-directory for the required instance.
- Name of the sub-directory will be the name of the instance.
- Create the following sub-directory under this instance's directory
- *config*
- the other directories described in the '*Structure of Nodes Directory*' above will be automatically created after the instance starts up.
- Depending on the type of instance needed, copy the required config files from the `INSTALL-
DIR/server/templates` to the configuration directory created above. The specific files needed are listed in Table 2 provided in the following section.

- Edit the configuration XML to setup the information as described in the following sections.
- Make sure that the cluster service is disabled in `server-config.xml`

```
<service name="ClusterService" enabled="false"
        class-name="com.pramati.cluster.PramatiClusterService">
```
- The instance can be started using `-node` option.

An Instance can be created by just creating a directory with an unique name under `server/nodes` directory. Once the directory is created, create another directory called `config` and copy `server-config.xml` and other configuration files required for the Standalone Server (as described in Table2) from the templates directory to `config`. Once the node directory is created, you can activate the Instance using `-node` command.

The same process can be followed for all the instance types including Message Server instances.

Once the Standalone J2EE Server is setup, based on the type selected, setup the basic attributes of the Web and EJB Containers. In the `server-config.xml`, change the values as needed:

```
<service name="WebContainer" enabled="true"
        class-name="com.pramati.web.WebServer">
    <config-file>web-config.xml</config-file>
    <requires always="NamingService,DeployService" />
    <property name="http-port" value="8181" />
    <property name="https-port" value="443" />
    <property name="ssl-enabled" value="false" />
</service>
```

Default value of HTTP port, and HTTPS port can be set now.

For EJB, in `server-config.xml`, the default pool size and session timeout value can be modified.

```
<service name="EJBContainer" enabled="true"
        class-name="com.pramati.ejb.EJBContainer">
    <requires always="NamingService,TransactionService,
        DeployService" />
    <property name="default-entity-min-pool-size" value="40" />
    <property name="default-entity-max-pool-size" value="1000" />
    <property name="default-session-min-pool-size" value="40" />
    <property name="default-session-max-pool-size" value="1000" />
    <property name="default-mdb-min-pool-size" value="40" />
    <property name="default-mdb-max-pool-size" value="1000" />
    <property name="default-low-activity-interval" value="60" />
    <property name="default-session-timeout" value="1000" />
    <property name="smart-code-generation" value="true" />
</service>
```

Enabling embedded Message Server in a Standalone Server

For enabling embedded Message Server for a Standalone Server follow the general instructions as given above for creating a Standalone Server. Add the following:

```
<service name="JMSService" enabled="true"
    class-name="com.pramati.jms.server.PramatiMessageService">
    <config-file>jms-config.xml</config-file>
    <requires always="NamingService,ResourceService"
    if-enabled="TransactionService" />
</service>
```

If it is not already present in your `server-config.xml` file. By default the standalone server is configured to start embedded Message Server service on services framework startup. '`enabled=true`' lets the server start the Message Server Service. For an embedded Message Server, both EJB and Web container can be enabled in `server-config.xml`.

```
<service name="WebContainer" enabled="true"
    class-name="com.pramati.web.WebServer">
    <service name="EJBContainer" enabled="true"
    class-name="com.pramati.ejb.EJBContainer">
```

Restricting services in a Standalone J2EE Server

Based on the Pramati services framework, the services can be disabled or enabled. The services include the core containers (Web and EJB) and other services such as Resource and Transactions. A Standalone J2EE Server by default has both Web and EJB Container services enabled:

```
<service name="WebContainer" enabled="true"
    class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
    class-name="com.pramati.ejb.EJBContainer">
```

For a Web only J2EE Server, enable the Web Container Service and disable the EJB Service.

```
<service name="WebContainer" enabled="true"
    class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="false"
    class-name="com.pramati.ejb.EJBContainer">
```

For an EJB only J2EE Server, enable the Web Container Service and disable the EJB and Message Server Services.

```
<service name="WebContainer" enabled="false"
    class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
    class-name="com.pramati.ejb.EJBContainer">
```

Creating Standalone Message Server

For a Standalone Message Server, enable the JMS Service in the `server-config.xml`:

```
<service name="JMSService" enabled="true"
  class-name="com.pramati.jms.server.PramatiMessageService">
  <config-file>jms-config.xml</config-file>
  <requires always="NamingService,ResourceService"
    if-enabled="TransactionService" />
</service>
```

Disable the Web Container and EJB Container services from `server-config.xml`:

```
<service name="WebContainer" enabled="false"
  class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="false"
  class-name="com.pramati.ejb.EJBContainer">
```

Understanding Pramati Cluster Configuration

Pramati Server features a powerful clustering solution. A cluster is defined as a group of nodes. Pramati Clustering solution offers transparent, self-organizing, homogenous web based configuration and management allowing easy EJB and Web clustering. When an application is deployed on any one of the cluster nodes, it is automatically replicated across all running cluster nodes. For a client, this translates to a single server view of the cluster.

Server clustering is self-organizing. Adding or removing nodes does not require restarting of the cluster. When a node is added, the load is automatically distributed across all the nodes based on their weight.

All the J2EE Services like Naming, Resource, Security and Transaction are automatically replicated across all the nodes across the cluster. Non-J2EE services like locking, application synchronization, load balancing, and failover are designed such that all complex cluster mechanics are hidden ensuring a single server view of the client.

Server clustering provides both EJB and Web clustering. Each node has its own set of replicated and non-replicated services. Pramati Cluster supports the Cluster-node types: Web, EJB and BOTH.

Types of Clusters supported are:

- J2EE Clustering
- Message Server Clustering

Cluster Definition

Cluster definition involves:

- Defining a logical cluster (this information is available with all the nodes of the cluster)
- Defining the nodes of the cluster

The initial definition of the cluster is made in the XML of the first node of the cluster. Once the basic cluster attributes are defined in the `cluster-config.xml`.

The cluster type is defined by the Cluster-node definition in the `cluster-config.xml`

```
<node>
  <name>node</name>
  <computer-name>127.0.0.1</computer-name>
  <naming-port>9191</naming-port>
  <http-port>8181</http-port>
  <node-type>BOTH</node-type>
  <node-id>001</node-id>
</node>
```

Here, when `node-type` is J2EE, enabling the cluster service would make it a J2EE Cluster. A J2EE Cluster can be of type BOTH (both `ejb` and `web`) or just `EJB/WEB`.

A J2EE Cluster cannot start with Message Server in an embedded mode and a Message Server cluster cannot have EJB or Web containers enabled. If `node-type` is "JMS", then it becomes a HA Message Server cluster.

Once cluster is enabled in `server-config.xml`, the cluster topology and properties are defined in the `cluster-config.xml`. Cluster nodes, topology, persistence (EJB/Web) and loadbalancer will be specified here.

J2EE clustering can involve both EJB and Web clustering. However Web and Message Server instances cannot exist in a single cluster. The Cluster solution is based on a common Cluster backbone (called Cluster service, which also runs on the Pramati Services framework).

Setting up a cluster involves a cluster definition and defining cluster nodes. The cluster definition essentially comprises defining the common configuration and state persistence information. Pramati Cluster being a self-organizing non master-slave homogenous cluster, does not have a separate entity for cluster housekeeping or processing. Based on the shared persistence store, the cluster runtime (cluster service) in each cluster node jointly perform all the housekeeping and cluster operational tasks.

Uploading the configuration to the DB (When DB is used for configuration persistence)

In case of db persistence (for cluster configuration), after initial setup of cluster node, to upload the configuration into the database, start the node for the first time with the following commandline option:

```
com.pramati.Server -node <NODENAME> -uploadconfig
```

The rest of the nodes would automatically pick up the configuration and do not need to be started with this option. After any change in the configuration, to update the database, start the cluster node with the following startup option:

```
com.pramati.Server -node <NODENAME> -updatelocalconfig
```

This option would update all of the configuration files. Only this node needs this option and the rest would automatically use updated configuration. To update specific configuration changed manually, start cluster node with the following startup option:

```
com.pramati.Server -node <NODENAME> -updateLocalConfigfor
<SERVICENAME>[, <SERVICENAME>]
```

Only this node needs this option and the rest would automatically use updated configuration.

Creating a Cluster Instance (steps common to all types of clusters)

The cluster definition is essentially 'defined' along with the first cluster node, and the same common persistence information and cluster details (including name) is to be setup in all the other nodes that will form a part of this cluster.

The steps involved in creating a cluster are:

- The DIR and the config files have to be setup just as one would do for a Standalone instance
- Define the name of the cluster

Open the `server-config.xml` file. Add a declaration as follows:

```
<belongs-to-cluster> MyCluster </belongs-to-cluster>
```

- Enable the cluster service

This cluster name is a logical representation for a group of nodes enabling them to communicate with each other. Though at present the nodes interact with each other through a repository of IP and port information, it is important that the nodes identify themselves with a common name. Once the node is identified as part of a cluster, the service definition for the cluster can be added in the `server-config.xml` file as follows:

```
<service name="ClusterService" enabled="true"
  class-name="com.pramati.cluster.PramatiClusterService">
  <config-file>cluster-config.xml</config-file>
  <requires always="NamingService" />
</service>
```

Set the 'enabled' attribute to true, so the cluster service is started when the services framework is started. As you can infer from the above code snippet, the actual configuration file for the cluster is stored by every node in `cluster-config.xml` file in the config directory.

- Setup persistence

The services configuration in a node can be stored **either** in a **file** or a **database**. These information can be specified in the `server-config.xml` file's `<configuration-persistence>` option. By default all the configuration information are persisted to the file.

```
<configuration-persistence use="file">
  <persistence type="file" />
  <persistence type="db">
    <property name="driver" value="" />
```

```

    <property name="url" value="" />
    <property name="user" value="" />
    <property name="password" value="" />
    <property name="tablename" value="" />
  </persistence>
</configuration-persistence>

```

Persistence can be configured either at the file level or DB level. If you are planning to have a DB level persistence, connection parameters like driver, connection URL, username, password and table name can be specified in the configuration file which are persisted. A driver can be any valid JDBC driver. The driver varies for different databases. The 'tablename' attribute is used to create a specific table where the configuration details for all the services are stored.

If you are setting up a new cluster, you can opt for a file level persistence, where the DB is not used for aiding a sync up. The new version of the server supports an 'On Demand sync up', wherein the configuration information for each node are synced up with every other node and persisted in the DB.

- Setup the cluster configuration file that would define all nodes that form part of this cluster. `cluster-config.xml` file captures the information on the nodes that forms a cluster. Initially when creating the cluster this information is to be provided in this file. This file will be identical in ALL the nodes that will form the cluster- with information on each one of them.
- After the first startup of the server, the information gets persisted in the DB/File persistence store of the cluster, and a read-only version of the file is always available (synced up from the DB on every 'start' of the cluster node). Copy the `cluster-config.xml` file from the templates directory to the `nodes/<node-name>/config` directory. Now add the node related information or topology for each node.

```

<node>
  <name>N1</name>
  <computer-name>128.128.77.77</computer-name>
  <naming-port>9191</naming-port>
  <http-port>8181</http-port>
  <node-type>BOTH</node-type>
  <node-id>001</node-id>
</node>
<node>
  <name> N2 </name>
  <computer-name> 128.128.77.78 </computer-name>
  <naming-port> 9190 </naming-port>
  <http-port> 8180 </http-port>
  <node-type> BOTH </node-type>
  <node-id> 002 </node-id>
</node>

```

Setting up a J2EE Cluster

Follow the General instruction for setting up a Cluster node (Refer to ‘*Creating a Cluster Instance—common to all cluster types*’), summarized here:

- Create the Node Directory and copy the configuration files required for the J2EE Node as mentioned in Table 2.
- Add cluster declaration in `server-config.xml`. Make sure that EJB and Web or EJB/Web are enabled and Message Server is disabled.

```
<service name="WebContainer" enabled="true"
  class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
  class-name="com.pramati.ejb.EJBContainer">
<service name="JMSService" enabled="false"
  class-name="com.pramati.jms.server.PramatiMessageService">
```

- Enable Cluster Service.

```
<service name="ClusterService" enabled="true"
  class-name="com.pramati.cluster.PramatiClusterService">
```

- Setup Persistence.
- Setup Cluster topology in `cluster-config.xml`.

Note: Make sure the node types mentioned in the node definitions in cluster-config.xml are of the type BOTH, Web or EJB

J2EE Cluster node types

The Cluster node types for both the nodes are indicated as BOTH, which means both the Web and EJB Services are running on that node. If you specify BOTH, it is required that you enable both EJB and Web services from the `server-config.xml` file. Other valid options are EJB and Web.

- *BOTH* – If a cluster node is defined as BOTH, both EJB and Web containers are started. Select this option when the application has both enterprise and web components.

A snippet from `server-config.xml`:

```
<service name="WebContainer" enabled="true"
  class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
  class-name="com.pramati.ejb.EJBContainer">
```

- *Web* – If a cluster node is defined as Web, only the Web container is started. EJB components are not available on this node. Select this option when the application has only JSP pages and other web components. A Standalone web node can act as an EJB Load Balancer.

A snippet from `server-config.xml`:

```
<service name="WebContainer" enabled="true"
  class-name="com.pramati.web.WebServer">
```

```
<service name="EJBContainer" enabled="false"
  class-name="com.pramati.ejb.EJBContainer">
```

- *EJB* – If a cluster node is defined as EJB, only the EJB container is started. Web components are not available on this node. Select this option when the application consists of only enterprise beans.

A snippet from `server-config.xml`:

```
<service name="WebContainer" enabled="false"
  class-name="com.pramati.web.WebServer">
<service name="EJBContainer" enabled="true"
  class-name="com.pramati.ejb.EJBContainer">
```

To summarize:

Table 8: J2EE Cluster Node Type and required services

| J2EE Node Type | Web Container | EJB Container |
|----------------|---------------|---------------|
| BOTH | Required | Required |
| EJB | - | Required |
| Web | Required | - |

Advanced Configuration

Typically a J2EE Cluster node means a ‘BOTH’ node. But a J2EE Cluster can also be configured as a Web only cluster, EJB only cluster or a combination of Web and EJB nodes.

Setting up Web Replication and Persistence

You can set up a Web cluster following the same procedure as shown in ‘*Creating a Cluster Instance-common to all cluster types*’. Make sure that you enable the Web container service for all the nodes. This can be done from the individual `server-config.xml` files.

Additionally, you need to add these information in your `cluster_config.xml` file.

```
<web-cluster>
  <enable>true</enable>
  <web-session-persistence>
    in-memory
  </web-session-persistence>
  <replication-on-all-nodes>
    true
  </replication-on-all-nodes>
  <web-cluster-persist-info>
    <url />
    <driver />
    <user />
```

```

        <password />
        <table-name />
    </web-cluster-persist-info>
</web-cluster>

```

Backup nodes can be setup for a Web cluster only when the web session persistence is set to In-Memory and replication on all nodes is set to false. Enable the Web cluster by setting '*enabled*' as '*true*'. There can be two kinds of Web session persistence:

- In-memory Persistence
- Database Persistence

When you select your Web session persistence to be in-memory, the HTTP session objects are persisted in memory locally for nodes in the same VM and persisted through RMI calls for nodes in different VM.

Whenever there is any state change for Web sessions, the information is replicated across all other nodes. This may prove expensive on memory and hence it is better to set '*replication-on-all-nodes*' to '*false*'. When the '*replication-on-all-nodes*' are set to '*false*', only the backup nodes are notified for replication. The Web cluster persistence information can be stored in a DB.

Setting up EJB Persistence

EJB session persistence for J2EE Cluster could be in-memory or database based. In case of database-based persistence, provide all the database information like URL, table name, etc. The following snippet is taken from `cluster_config.xml`:

```

<ejb-session-persistence>in-memory</ejb-session-persistence>
  <persist-info>
    <url />
    <driver />
    <user />
    <password />
    <table-name />
  </persist-info>

```

The above example uses In-memory EJB session persistence.

Setting up an HA Message Server Cluster

General instructions for setting up a cluster node (Refer to '*Creating a Cluster Instance*')

- 1 Create the Node Directory and copy the configuration files required for a Message Server node as mentioned in Table 2.
- 2 Add cluster declaration in `server-config.xml`. Make sure the JMS Service is also enabled and Web and EJB are disabled.

```

<service name="WebContainer" enabled="false"
  class-name="com.pramati.web.WebServer">

```

```
<service name="EJBContainer" enabled="false"
  class-name="com.pramati.ejb.EJBContainer">
<service name="JMSService" enabled="true"
  class-name="com.pramati.jms.server.PramatiMessageService">
```

3 Enable cluster service.

```
<service name="ClusterService" enabled="true"
  class-name="com.pramati.cluster.PramatiClusterService">
```

4 Setup persistence.

5 Setup cluster topology in cluster-config.xml.

Note: Make sure the node types mentioned in the node definitions in cluster-config.xml are only of the type jms-cluster.

A sample node configuration for a Message Server cluster:

```
<node>
  <name>node</name>
  <computer-name>jms1</computer-name>
  <naming-port>2099</naming-port>
  <node-type>jms-cluster</node-type>
  <node-id>001</node-id>
</node>
```

The above snippet is from cluster-config.xml.

Note: A Message Server Cluster cannot have a Web/EJB node. Hence make sure that the Web/EJB container services are disabled for all the Message Server nodes.

Updating cluster configuration using XML (when DB is used for configuration)

Once cluster is enabled in server-config.xml, the cluster topology and properties are defined in the cluster_config.xml. Cluster nodes, topology, persistence (EJB/Web) and Load Balancer will be specified here. After the initial configuration of the cluster, the cluster configuration may be modified using the Console or the XMLs.

In a file based persistence, once the config XMLs are modified in any of the nodes, and the cluster is restarted, the new configuration takes effect and is synched-up with all other nodes.

When DB is used for config persistence, the following steps need to be followed to update the configuration in the DB, using local XMLs to provide the information:

- The cluster must be stopped
- In any of the nodes, update the required config XML
- Start the node, where the XMLs were modified:
 - After any change in the configuration, to update the database, start cluster node with the following option: `com.pramati.Server -node <NODENAME> -updateLocalconfig`
This updates all of the configuration files.

- To update specific configuration changed manually, start cluster node with the following **startup option**: `com.pramati.Server -node <NODENAME> -updateLocalConfigfor <SERVICENAME> [<SERVICENAME>]`

Only this node needs this option and the rest would automatically use updated configuration.

Configuring JMS Adapters in J2EE Servers

JMS Adapters are setup in a J2EE Server instance determining the connectivity to a Message Server (Embedded, Standalone or HA-Message Server). JMS Adapters can be configured in `resource-config.xml` file of the J2EE Server (Standalone or cluster-node).

In case of non-embedded Message Server, that is when the JMS Service is not running in the same VM as the EJBService, specify `com.pramati.naming.client.PramatiClientContextFactory` for `java.naming.factory.initial`. Also specify the IP and port of the VM where the Message Server is running in `resource-config.xml`. In case of HA Message Server, provide the comma separated list of URLs of HA Message Server nodes.

Sample configuration for the local JMS Adapter is shown below. The sample configuration is valid only for a Message Server running in embedded mode:

```
<jms-adapters>
  <jms-adapter name="default" description="nodesc" interface-
    class="com.pramati.jms.client.JMSProviderImpl">
    <properties>
      <property name="java.naming.factory.initial" value="com.pra-
        mati.naming.PramatiLocalContextFactory"/>
      <property name="java.naming.provider.url" value="rmi://local-
        host:9191"/>
    </properties>
  </jms-adapter>
</jms-adapters>
```

Sample configuration for the external JMS Adapter (non-HA) is shown below. The sample configuration is valid only for a Message Server not running in embedded mode. `jmshost` is the host where the Standalone Message Server is running on Port 2099:

```
<jms-adapters>
  <jms-adapter name="default" description="nodesc" interface-
    class="com.pramati.jms.client.JMSProviderImpl">
    <properties>
      <property name="java.naming.factory.initial" value="com.pramati.nam-
        ing.PramatiClientContextFactory"/>
      <property name="java.naming.provider.url" value="rmi://jmshost:2099"/>
    </properties>
  </jms-adapter>
```

```
</jms-adapters>
```

Sample configuration for the external JMS Adapter (HA) is shown below. The sample configuration is valid only for an HA-Message Server cluster. *jms2*, *jms1* are part of HA Message Server configuration.

```
<jms-adapters>
  <jms-adapter name="default" description="nodesc" interface-
class="com.pramati.jms.client.JMSProviderImpl">
  <properties>
    <property name="java.naming.factory.initial" value="com.pramati.nam-
ing.PramatiClientContextFactory"/>
    <property name="java.naming.provider.url" value="rmi://
jms2:9191,rmi://jms1:9191"/>
  </properties>
</jms-adapter>
</jms-adapters>
```

Note: For HA Message Server node or a Standalone Message Server node, the resource-config.xml should not contain any JMS Adapters.

Web Load Balancer

A Load Balancer is a type of Server instance that works as a web-request dispatcher, distributing the web requests across a set of Web/J2EE Cluster nodes. In advanced configurations, it distributes the web requests even across non-cluster server instances and multiple vendor's servers. Load Balancing for EJB clusters is done at the container level and does not need any external configuration.

A Load Balancer is a variant of Web node. The steps involved in setting up a Load Balancer are:

- 1 The DIR and the config files have to be setup just as one would do for a Standalone instance.
- 2 As defined in Table 2 above, copy the `server-config.xml` and `web-lbconfig.xml` into the config DIR

- 3 Enable the Load Balancer in `web-config.xml`

```
<request-dispatcher enabled="true" name="web-lbconfig.xml"/>
```

- 4 Configure the Load Balancer properties in `web-lbconfig.xml`. The nodes are referenced by their name:

```
<node name="first" type="pramati" host="localhost" web-port="8484">
  <naming-port>9494</naming-port>
  <socket-pool min="20" max="50" idle-time-out="1000"/>
</node>
```

You can also configure node chooser and node group information from `web-lbconfig.xml`. For more information on node chooser, node group and failover refer to the *Technical Reference* document.

Customizing Server Footprint

The Server framework allows customizing a server instance to the extent of choosing the specific set of services required by the application. This will result in just those services being started. The binary files (jar) for the remaining services can even be removed from the Server install directory. Services can be enabled and disabled by changing configuration in `server-config.xml`. More details are dealt in the following section.

Server Architecture

Overview

The Server framework is a system composed of a set of independent services to provide extensibility and enable assembling of server flavors such as Web-only server or EJB-only server. The Integration Service allows developers to decompose an application and enables reusable design. To address the need to separate the implementation of Services and eliminate crossdependencies resulting in compilation issues, the framework has been designed to enable parallel development, pluggability and modularity to suit custom requirements.

Pramati Services framework is the core services sandbox for Pramati Server, which enables services to be plugged in, and removed seamlessly with minor configuration changes. Pramati Services framework was conceived as a flexible component framework designed to facilitate the development of sophisticated enterprise solutions from a bunch of independent and interdependent services. The services framework helps greatly in reuse of services and thus enabling the decomposition of the application into a set of manageable services.

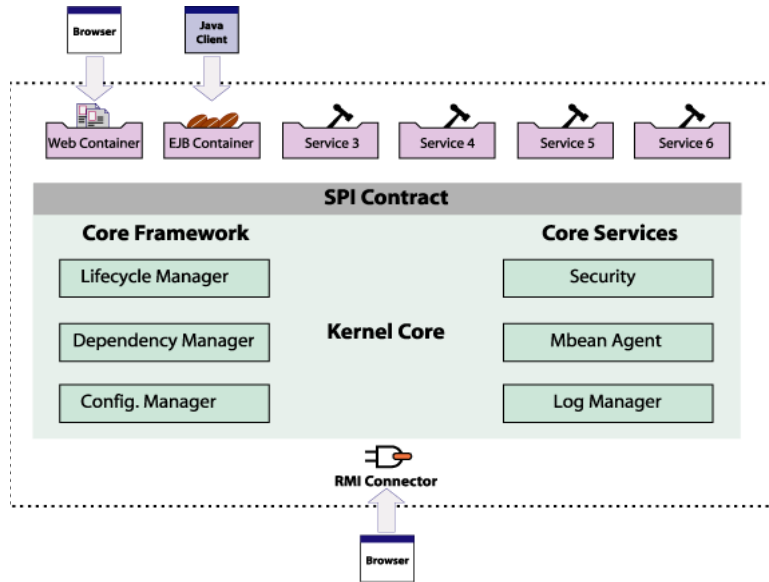


Fig Services Framework

Architecture

Pramati Services framework comprises of the following core components:

- Life Cycle Manager
- Dependency Manager
- Configuration Manager

Pramati core services are

- Security
- JMX
- Logging

The services are classified into:

- *Services (Normal)* - These are services, which cannot be configured.
- *Configurable Services* - These are services, which can be configured using an external configuration file.
- *Re-configurable Services* - These are the services, which can be re-configured after successful deployment in the server framework.

Some of the default service configurations which comes with Pramati Services framework are:

- Naming Service

- Transaction Service
- Deploy Service
- Resource Service
- Web Container
- EJB Container
- JMS Service
- Management Service
- Cluster Service

Kernel

Pramati Services framework (PSF) architecture comprises of the Kernel, which provides arbitration for all the framework components and takes care of all services level contracts. It creates and manages the Management Agent, the services repository and other global components required for the function of the services. PSF Kernel is responsible for starting and managing core framework components like the Lifecycle Manager, Dependency Manager and Configuration Manager. The Kernel provides the management and service registration capabilities and also logging, security, packaging, configuration and deployment functionalities.

Pramati Server creates, maintains and destroys the Kernel instance. Upon Kernel startup, the Dependency Manager invokes the Dependency Analyzer, which reads the list of services and their dependencies and categorizes them according to their types, that is Enabled, Circular, Disabled and Invalid. The types of dependencies will be discussed later in this chapter.

Core Services

PSF Kernel comprises of three core services

- MBean Server
- Security
- Logging

Core Components

Lifecycle Manager

Based on the lifecycle of each service, the functionality of the Lifecycle Manager can be broadly classified as:

Table 9: Functionalities of a Lifecycle Manager

| Function | Description |
|---|---|
| Loading the service class | The service class mentioned in the configuration file is loaded into the Kernel VM and instantiated of type Service. All classes which need to be started up as a service by the Lifecycle Manager should implement the Service Interface – <code>com.pramati.services.framework.Service</code> <pre>Class serviceClass = Class.forName(serviceClassName);</pre> |
| Call the no-args constructor of the service | The service implementation class, which needs to be started, should have a constructor with no argument signature. This is mandatory for starting the service as the Lifecycle Manager does not assume or try to obtain the initialization parameters of the service. |
| Initialize the service data | Every configurable service will have service data associated with them. These are service specific information, which needs to be taken care of when the service is deployed. The service data is initialized for identifying the service specific configuration. |
| Set Service context | Once the service data is initialized, the service context or the runtime environment for each service is setup, so the service can communicate with the kernel through the service context. |
| Configure service (Optional) | The next step after setting the service context is to configure the service. The services are configured only if they are set as 'configurable' or 're-configurable'. |
| Start service | After the service is configured, the service is started by instantiating the service implementation class file. <code>Service serv = (Service) serviceClass.newInstance();</code> |
| Reconfigure service | When a service is started as a re-configurable service, the service can be re-instantiated whenever its configuration file gets modified. This can be achieved by setting up a service specific listener which tarps all the configuration modification events. And in the event of the service modification, the life cycle manager reconfigures the service. |
| Stop service | The service can be stopped or disabled even when it is plugged in and running. |
| Destroy service | The service can be destroyed are unplugged from the services framework at anytime. Note that the service implementation class file remains undeleted but starting the service follows the entire initial lifecycle process including configuration and setting up new service context. |

Configuration Manager

The configuration manager reads and loads the service specific configuration into kernel memory. The configuration manager performs the following actions:

Table 10: Functionalities of a Configuration Manager

| Function | Description |
|----------------------------|--|
| Get Node Information | Retrieves the Server Instance information which includes default login information like username, password and realm, RMI properties like socket time out, export port, SSI export port, SSL protocol state and class file server port number. |
| Get Configuration | Reads the global configuration from the server-config.xml and also reads configuration for individual services. The Configuration object has a timestamp associated with it. |
| Save Service Properties | The properties of the services are saved in kernel memory space for faster processing. This holds true for services like the EJB Container, which has properties like min-pool size, max-pool size, low-activity interval, session-timeout and smart-code-generation. |
| Save Configuration | The default XML manager picks up the configuration DOM and writes to a location. Note that if there is any configuration, which has to be synced up, cluster wide cannot be written in the service config. |
| Add Configuration Listener | Any listener would be monitored at the frequency specified in the config. file. If there is no such entry, the default monitoring frequency of 30 sec is registered. |
| Register Configuration | Registers all external configurations, which are not part of Pramati Services Framework. A third party load balancer might send a configuration handler and a registration key to be registered with the framework. |
| Get Persistent Mode | The configuration can either be persisted in a file or a database. The configuration Manager reads the content of the <configuration-persistence> tag from the server-config.xml file, which includes persistent type, and information like driver name, url, username and password in case of db persistence. |

Dependency Manager

Dependencies can be classified as:

- Regular (Always)
- Optional

If a dependency is stated as 'always' in the configuration file of a service, then the dependent service should be started before starting the service. But if the dependency is optional, the dependent service need not be started before the service. Based on the rules the service can have the following dependency states:

- *Enabled*- In 2 services dependency, if service S1 depends on S2, S2 should be enabled for the dependency to reach this state.
- *Disabled*- In 2 services dependency, if service S1 depends on S2, and if S2 is disabled, the Dependency Manager throws an Exception and stops the server.

- *Invalid*- If a service has specified an unknown or non-configured service as the dependent service, the Dependency Manager throws an Exception and stops the server.
- *Circular*- Circular dependency state happens when two services pointing to each other as their dependent service. Hence the server would not be able to start any of the service.

The dependency information for services are specified in the `server-config.xml` file. The code snippet below indicates the service definition of the cluster service.

```
<service name="ClusterService" enabled="false"
  class-name="com.pramati.cluster.PramatiClusterService">
  <config-file>cluster-config.xml</config-file>
  <requires always="NamingService" />
</service>
```

Here 'name' indicates the service name, 'class-name' points to the service implementation file, 'config-file' holds the cluster specific configuration information, and 'requires' sets the dependency for the service. In the case of the cluster service, naming service should have been started before. This is an instance of regular dependency.

Inter Service Dependency

The inter service dependency of the core services is depicted in the following table:

Table 11: Inter Service Dependency

| Service | Startup Class | Dependency | Configuration |
|---------------|--|---|---------------------|
| EJB Container | com.pramati.ejb.EJBContainer | NamingService, TransactionService, DeployService | None |
| Web Container | com.pramati.web.WebServer | Naming Service, Deploy Service | web-config.xml |
| Naming | com.pramati.naming.PramatiNamingServiceJRMP | None | None |
| Transaction | com.pramati.jta.PramatiJTATransactionService | Naming Service | None |
| Deploy | com.pramati.j2ee.deploy.PramatiDeployService | Naming Service, Cluster Service (If enabled) | deploy-config.xml |
| Resource | com.pramati.resource.PramatiResourceService | Naming Service, Transaction Service, Deploy Service (If enabled). | resource-config.xml |
| JMS | com.pramati.jms.server.PramatiMessageService | NamingService, ResourceService, TransactionService (If enabled) | jms-config.xml |

Table 11: Inter Service Dependency

| Service | Startup Class | Dependency | Configuration |
|------------|--|--|-----------------------|
| Management | com.pramati.admin.base.ManagementServiceImpl | NamingService, WebContainer (If enabled) | management-config.xml |
| Cluster | com.pramati.cluster.PramatiClusterService | NamingService | cluster-config.xml |

Log Manager

The services framework supports an advanced logging mechanism providing an ability to monitor the services running in the framework. The logging itself comprises of the actual mechanism in which the messages are logged and the functionality of the Log Manager. The Log Manager forms the core component of the framework, which provides an interface between the services framework and the logging mechanism. It abstracts the logging mechanism from direct access. The Log Manager performs the following tasks:

- Get Logging properties
- Get Trace Admin
- Get Rotation Type
- Get File Size
- Set Rotation Type
- Set File Size
- Set Rotation time

Rotation is the process of saving the backup of the log in some location (db or file) and refreshing the log file to start as a new process. There can be two kinds of rotation 1. Rotation based on duration 2. Rotation based on size.

Log Manager is capable of setting the rotation type and managing the log files for each node in the cluster.

Services framework features

- Implementation of a service is abstracted from all other services enabling ease of maintenance
- Contracts are defined upfront enabling cleaner separation
- No compile-time dependencies enabling loosely-coupled coexistence of services through SPI contracts
- Subsettable and Extensible, enabling adding and starting services in the same VM
- Modular, enabling for reduction in footprint and overhead
- Modular and Manageable