OSGi – An easy way to create dynamic, live architectures and applications

Open Services Gateway Initiative – OSGI, provides architecture for developing and deploying modular applications and libraries. Geeks do refer it as Dynamic Module System as well for Java. The lightweight and customizable framework allows decomposing a complex application into multiple modules. In this section we will cover:

- *What is OSGi*
- *History of OSGi*
- *Benefits of OSGi*
- *OSGi Architecture Overview*

What is OSGi

As per its most prevalent definition, **OSGi** (Open Service Gateway Initiative) is a Java framework for developing and deploying modular software programs and libraries. Each bundle is a tightly coupled, dynamically loadable collection of classes, jars, and configuration files that explicitly declare their external dependencies (if any). The OSGi is a technology being considered as set of specifications that actually define a dynamic component system for Java.

History of OSGi

OSGi Alliance was founded in 1999 by Ericsson, IBM, Motorola, Sun Microsystems and others for Open Services Gateway initiative. OSGi specification release 1 was the first release doe by OSGi in 2000. The basic objective of OSGi specification was to define Java-based service platform to be a complete dynamic component model.

JMC does not support natively dynamic module system to get started, stopped or updated at runtime. Hence, the basic intention for developing OSGi specification was starting, stopping, updating application at runtime. At the same time, it was expected to manage JAR dependencies management missing as way to restrict exported packages, enforce imports was lacking earlier.

Benefits of OSGi

- Reduced Complexity

  It is said that, while you are developing with OSGi technology, you are developing bundles: the OSGi components. These bundles are modules. They can easily hide their internals from other bundles and communicate through well-defined services.

Hiding internals means more freedom to change later. This not only reduces the number of bugs, it also makes bundles simpler to develop because correctly sized bundles implement a piece of functionality through well-defined interfaces.

- Reuse

  With the prominent usage of the OSGi component model, it is extremely easy to use many third party components in an application. As per latest scenario, rising numbers of open source projects provide their JARs ready made for OSGi. Even commercial libraries are also being offered as readymade bundles.

- Easy Deployment

  Not being limited as just a standard for components, the OSGi technology specifies how components are installed and managed and many bundles utilize its API to furnish a potential management agent. This management agent can be as simple as a command shell, a TR-69 management protocol driver, OMA DM protocol driver, a cloud computing interface for Amazon's EC2, or an IBM Tivoli management system. Due to its standardized management API, it is very plain and simple to integrate OSGi technology in existing and future systems.

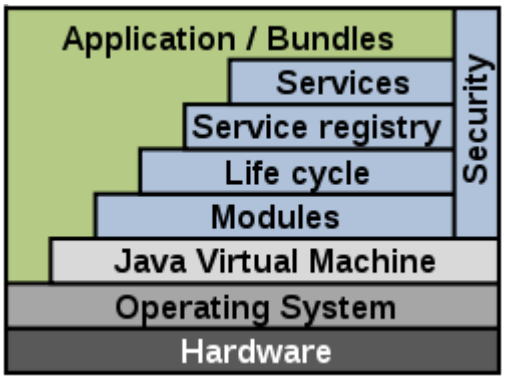
- Dynamic Updates

  The OSGi component model is a dynamic model. You don't have to bring whole system down to install, start, stop, update and uninstall the bundles. Sometimes, developers do not prefer to use this in production as they find it difficult to swallow the fact. But once they use it, they about its capabilities especially useful in reduction of time while deployment.

- Transparency

  In the world of technologies, transparency is being stated as major advantage of OSG. Bundles and services are considered as first class citizens in the OSGi environment. The management API provides access to the internal state of a bundle as well as how it is connected to other bundles. For example, most frameworks provide a command shell that shows this internal state. Parts of the applications can be stopped to debug a certain problem, or diagnostic bundles can be brought in. Instead of staring at millions of lines of logging output and long reboot times, OSGi applications can often be debugged with a live command shell.

OSGi Architecture Overview



Please find an industry prevalent short definition of the terms described in figure.

- Bundles – Bundles are the OSGi components made by the developers.
- Services – The services layer connects bundles in a dynamic way by offering a publish-find-bind model for plain old Java objects.
  In this layer, service providers publish services to service registry, while service clients search the registry to find available services to use.
  This is like a service-oriented architecture (SOA) which has been largely used in web services. Here OSGi services are local to a single VM, so it is sometimes called *SOA in a VM*.

- Life-Cycle – The API to install, start, stop, update, and uninstall bundles.

  This layer defines how bundles are dynamically installed and managed in the OSGi framework. It provides a way for a bundle to gain access to the underlying OSGi framework.

- Modules –

  The layer that defines how a bundle can import and export code. Module layer defines OSGi module concept - bundle, which is a JAR file with extra metadata.
  A bundle contains class files and related resources such as images, xml files.
  Through manifest.mf metadata file, Module layer declare which contained packages in JAR file are visible to outside, and declare on which external packages the bundle depend.

  Some metadata examples:
  Export-Package: com.azilen.helloexport
  It declares which packages are visible to users.
  Import-Package: com.azilen.helloimport
  It declares when external packages are required.

- Security – The layer that handles the security aspects.

- Execution Environment – Defines what methods and classes are available in a specific platform.

As programmer, you might have faced several situations while starting new projects, where you had to start over from scratch almost every time. That's not just the code you write for the project build, it includes underlying codes as well that support the project. This makes coding a monotonous task rather than a creative logical thinking process.

Liferay Portal can be at your immense help. It proffers a baseline set of features that would help in giving a head start on the entire repetitive code. It offers a platform to quickly build web applications, mobile applications and web services. Its east to use features and framework are specially designed to handle rapid development. Some of its tools such as Blade CLI make programming easy and interesting.

Let us make a start with having an in-depth understanding about:

- What is Blade CLI?
- How it can be used with any IDE or development environment?

What is Blade CLI?

As per its techie definition, Blade CLI is a command line tool that is bootstrapped on to a Gradle based environment used for building Liferay 7.0 modules. It is one of the easiest ways for Liferay developers to create new **Liferay portal solutions** and modules. You can simply download blade tools JAR and once you've obtained the Blade Tools JAR, you can install Blade Tools through the Java Package Manager (JPM).

Using JAVA Package Manager (JPM) To Download Blade Tools:

1) Install JAVA Package Manager (JPM) from here if you don't have installed in your machine yet
2) Once JPM is installed, we need to download com.liferay.blade.cli.jar from here.
3) Next, we can install Blade tools from downloaded JAR with the use of below command.

```
(sudo) jpm install -fl [Downloads Directory]/com.liferay.blade.cli.jar
```

4) Alternate command to install blade tools.

```
jpm install -f https://liferay-test-01.ci.cloudbees.com/job/blade.tools/lastSuccessfulBuild/artifact/com.liferay.blade.cli/generated/com.liferay.blade.cli.jar
```

Verifying **Blade CLI** installation:

1) Verify your installation by writing “blade” command in command prompt. We can see below list of MAIN OPTIONS and available sub-commands for the installed blade tools.

MAIN OPTIONS

```
[ -b, --base <string> ]  - Specify a new base directory(default working
directory).

[ -e, --exceptions ]   -  Print exception stack traces when they occur.

[ -f, --failok <string;> ]  -  Do not return error status for error that matches
this given regular expression.

[ -k, --key ]  -  Wait for a key press, might be useful when you want to see the
result before it is overwritten by a next command

[ -p, --pedantic ] - Be pedantic about all details.

[ -t, --trace ] - Trace on.

[ -w, --width <int> ]  -The output width, used for wrapping diagnostic output

Available sub-commands:

agent  - Install or uninstall remote agent in Liferay

create   -  Creates a new Liferay module project from several available
templates.

deploy - Deploys a bundle to the Liferay module framework.

gw  - Execute gradle command using the gradle wrapper if detected

help - Get help on a specific command

init - Initializes a new Liferay workspace

migrateTheme  - Migrate a plugins sdk theme to new workspace theme project

open  - Opens or imports a file or project in Liferay IDE.

samples - Generate a sample project

server - Start or stop server defined by your Liferay project

sh - Connects to Liferay and executes gogo command and returns output.

update - Update blade to latest version

version   - Show version information about blade
```

2) We have installed Blade tools, now let's understand its 2 important command: blade create and blade deploy

3) To understand how to use blade create, enter blade create command into terminal. Below is output that terminal produces.

4) We need to make sure that our module is built before we execute blade deploy command as this command will be used to deploy application in Liferay.

```
NAME

create   -   Creates a new Liferay module project from several available
templates.

SYNOPSIS

create [options] <name>

OPTIONS

[ -c, --classname <string> ] - If a class is generated in the project, provide
the name of the class to be generated. If not provided defaults to Project name.

[ -d, --dir <file> ] - The directory where to create the new project.

[ -h, --hostbundlebsn <string> ] - If a new jsp hook fragment needs to be
created, provide the name of the host bundle symbolic name.

[ -H, --hostbundleversion <string> ] - If a new jsp hook fragment needs to be
created, provide the name of the host bundle version.

[ -p, --packagename <string> ] - Package name

[ -s, --service <string> ] - If a new DS component needs to be created, provide
the name of the service to be implemented.

[ -t, --template <template> ]  - The project template to use when creating the
project. The following templates are available: activator, jsphook, mvcportlet,
portlet, service, servicebuilder, servicewrapper
```