Overview of the Liberty profile

Unit objectives

After completing this unit, you should be able to:

- Describe the characteristics and architecture of the Liberty profile
- Install the Liberty profile runtime environment
- Create a Liberty profile server by using developer tools and command-line utilities
- Describe the configuration features for a Liberty profile server
- Use flexible configuration and shared libraries
- Deploy applications by using a monitored directory
- Deploy applications by using developer tools
- Package an application and Liberty profile runtime
- Describe the process for enabling security for a Liberty profile server
- Use the job manager to manage Liberty profile servers
- Describe the characteristics of a Liberty collective
- Describe the characteristics of Liberty profile server clusters

Topics

- Introduction to the Liberty profile
- Tools, run time, and installation
- Configurations
- Security
- Using the job manager
- Liberty collectives and clusters

Introduction to the Liberty profile

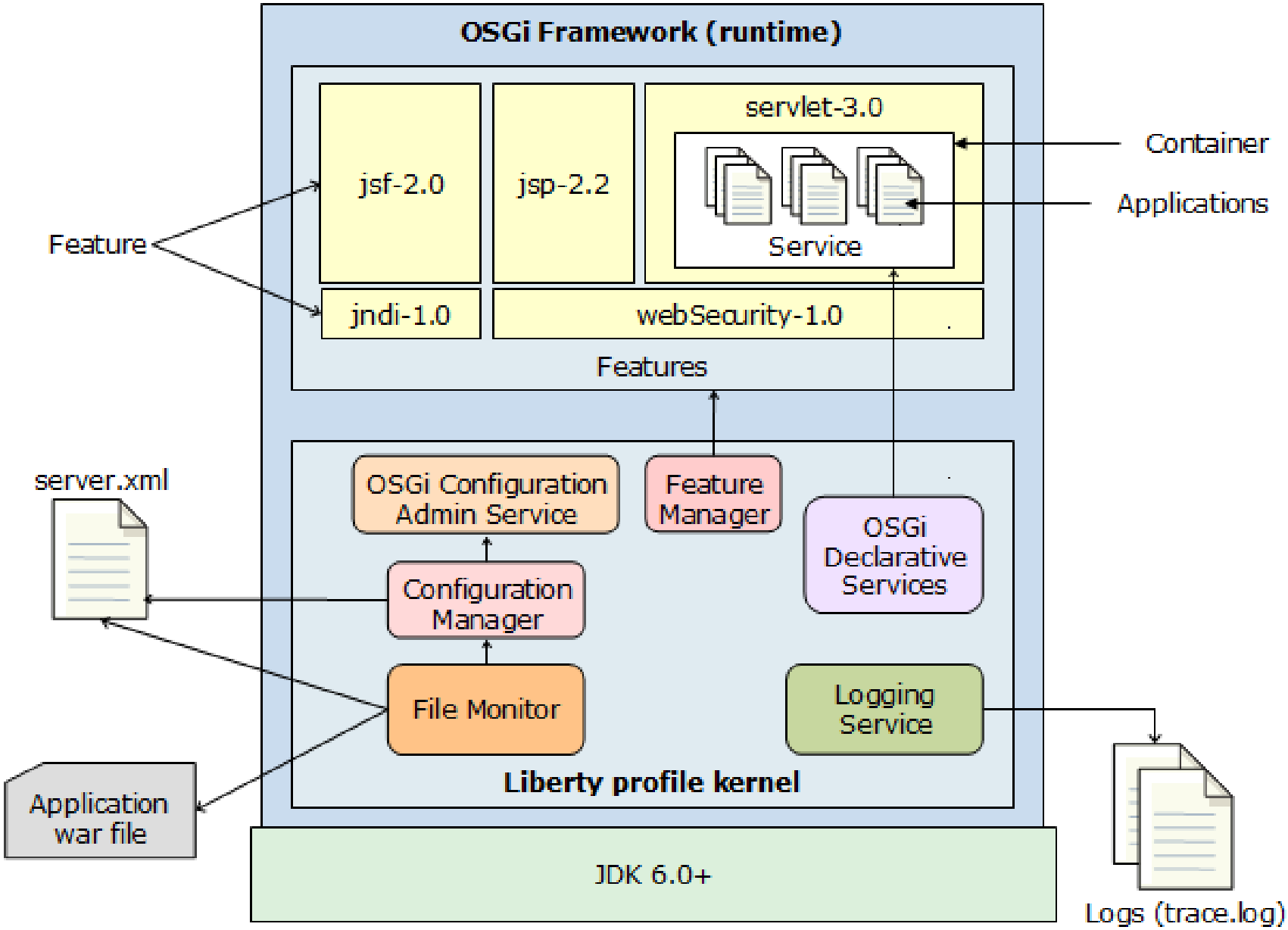
What is the Liberty profile?

- The Liberty profile is an application server runtime environment
  - Highly composable by using feature elements
  - Configuration changes are dynamic on a running server
  - Server starts quickly
- You can install a Liberty profile runtime environment by extracting a JAR file
  - The Liberty profile does not include a Java runtime environment (JRE), so you must install an IBM or Oracle JRE separately
- Liberty profile servers support two models of application deployment:
  - Deploy an application by dropping it into the `dropins` directory
  - Deploy an application by adding it to the server configuration
- The Liberty profile supports a subset of the full WebSphere Application Server programming model:
  - Web applications
  - OSGi (Open Service Gateway initiative) applications
  - Java Persistence API (JPA)
- Can be easily configured to support the full Java EE 6 Web Profile

What does the Liberty profile provide?

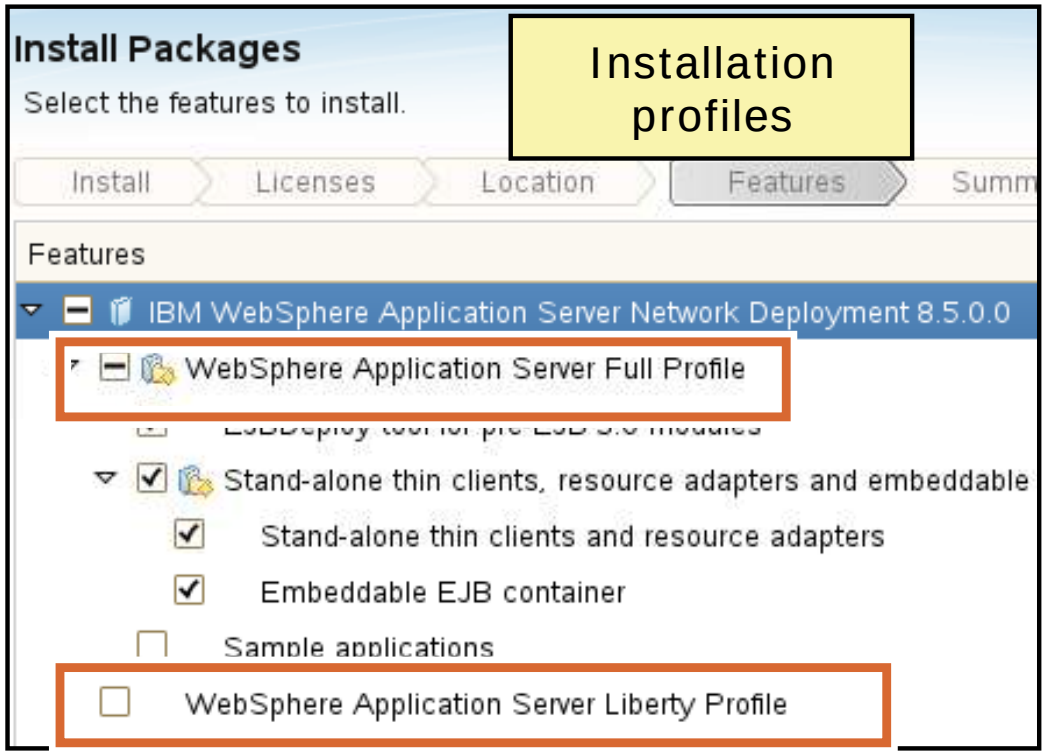
- The Liberty profile provides a lightweight development and application-serving environment that is optimized for developer and operational productivity
- The Liberty profile includes the following key features:
  - Dynamic and flexible runtime (loads only what the application needs)
  - Quick server start time (under 5 seconds with simple web applications)
  - Simplified configuration that uses a single configuration file or modular configuration
  - Support for deploying applications that are developed in the Liberty profile to run in the full profile
  - Support for LDAP user registries
  - Ability to deploy an application and configured server as a package
  - Managed, centralized deployment to multiple nodes of a packaged application and server by using the job manager
  - Availability of WebSphere Application Server Developer Tools as Eclipse plug-ins for broad tool support
  - Support for z/OS platform native features like System Authorization Facility (SAF), Resource Recovery Service (RRS), and z/OS workload management (WLM)

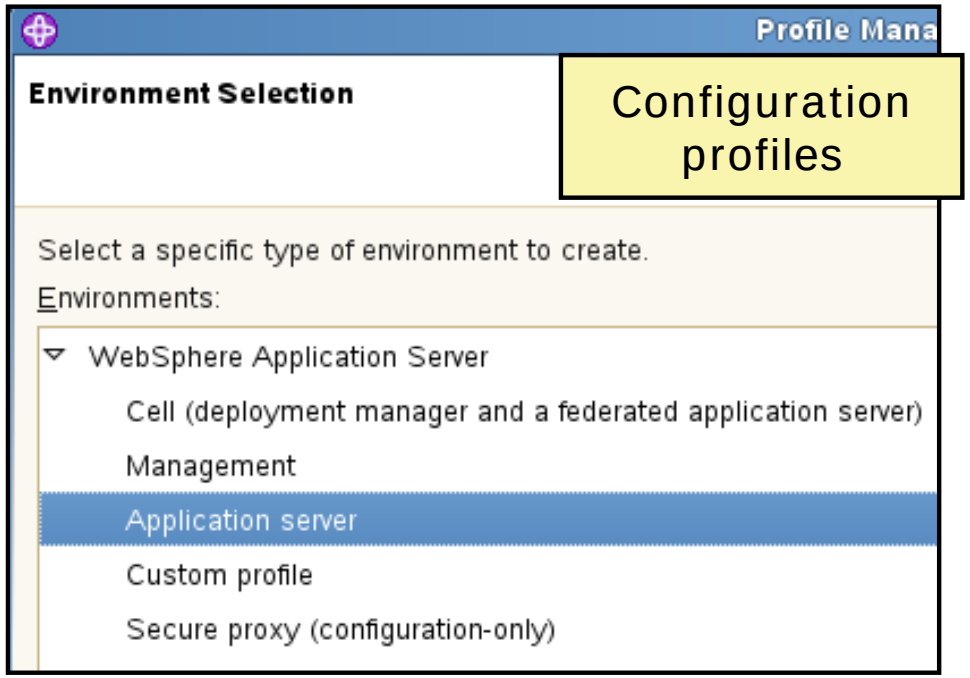
Liberty profile architecture



Clarification: Profile versus profile

- How does the Liberty profile relate to the application server and custom profiles?
- The term “profile” has another meaning in WebSphere V8.5
  - **Installation profile** (full or traditional profile versus Liberty profile): Refers to what runtime is being installed
  - **Configuration profile** (deployment manager, application server, custom): Refers to which configuration is being used within the full installation





WebSphere Application Server family

WebSphere Application Server for Developers

Liberty profile

WebSphere Application Server Hypervisor Edition

Liberty profile

WebSphere Application Server Network Deployment

Liberty profile

WebSphere Application Server for z/OS

Liberty profile

WebSphere Application Server

Liberty profile

WebSphere Application Server Liberty Core

A lightweight and low-cost Liberty-profile-based offering, providing capabilities to rapidly build and deliver web apps that do not require the full Java EE stack

WebSphere Application Server Express

Liberty profile

Features available in the Liberty editions

Liberty Core	Base, Express	Network Deployment	z/OS
servlet jsp jsf jpa jndi jdbc json jaxrs wab ssl osgi.jpa monitor sessionDatabase appSecurity blueprint restConnector localConnector beanValidation ejblite cdi managedBeans oauth ldapRegistry webCache concurrent collectiveMember	was.JmsClient was.JmsServer wsSecurity wmq.JmsClient was.JmsSecurity mongodb jaxb jaxws	collectiveController clusterMember	zosSecurity zosTransaction zosWlm

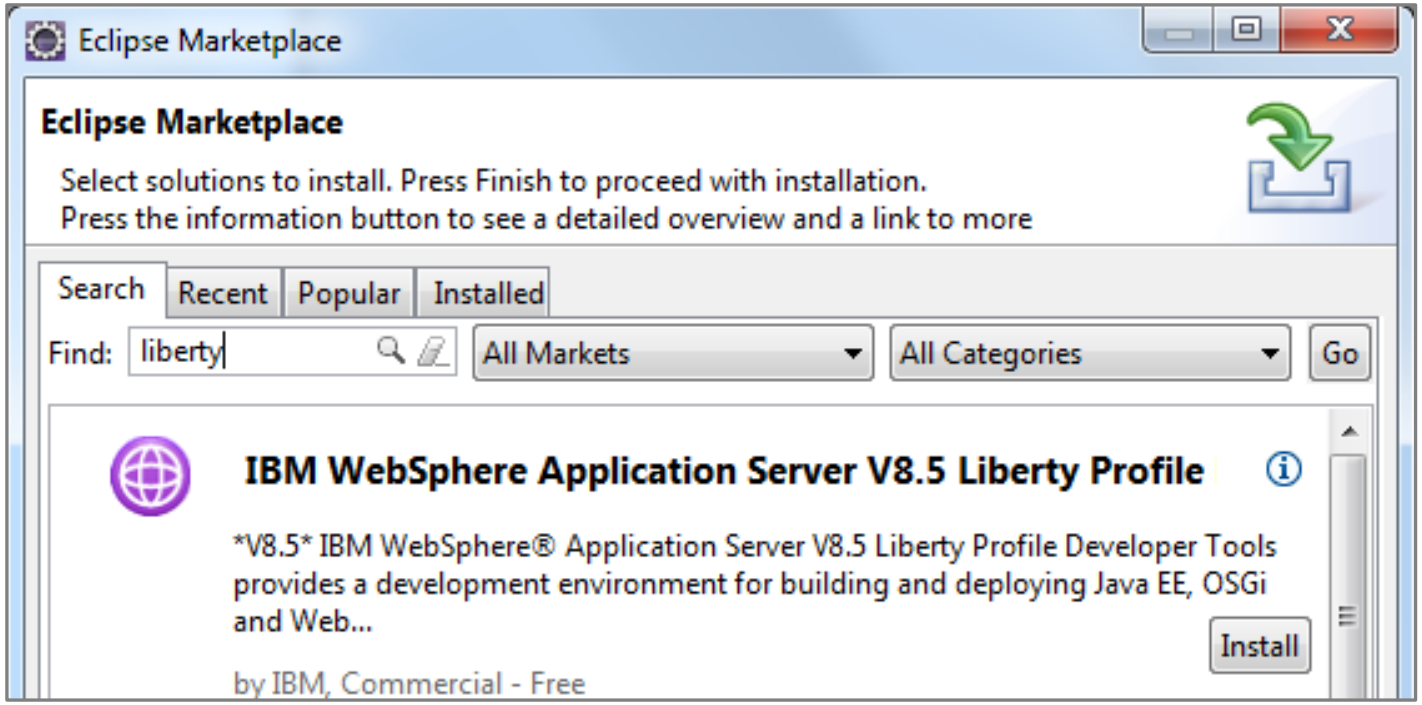
Tools, runtime, and installation

Liberty profile developer tools

- Applications can be developed and tested on a Liberty profile server by using the following developer tools
  - WebSphere Application Server Developer Tools for Eclipse
  - IBM Assembly and Deploy Tools for WebSphere Administration
  - Rational Application Developer

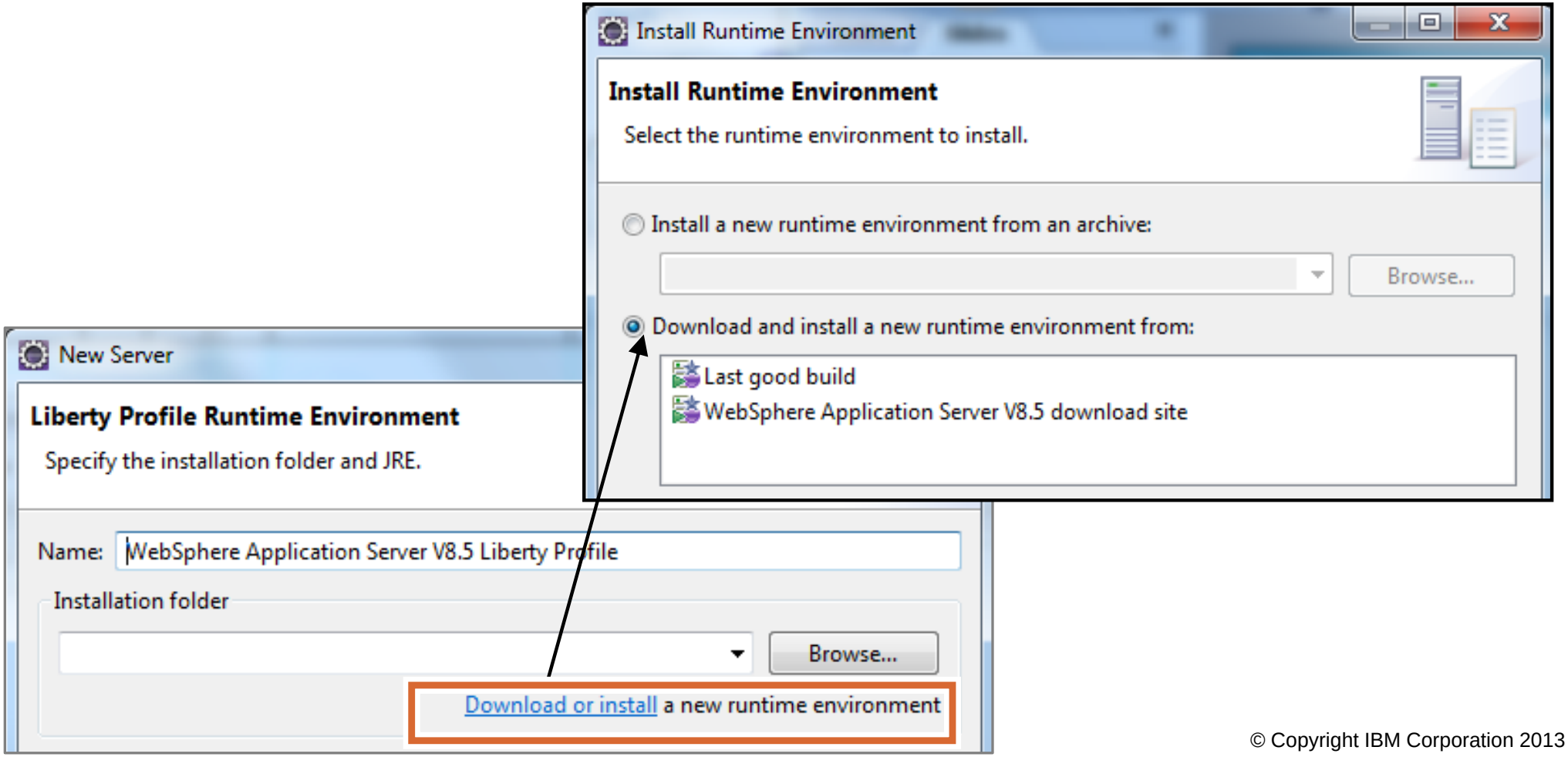
Where to get WebSphere Application Server Developer Tools for Eclipse

- WebSphere Application Server Developer Tools for Eclipse is available for free:
  - From `http://www.wasdev.net`
  - Through the Eclipse Marketplace (**Help > Eclipse Marketplace**)
  - Through the IBM Installation Manager



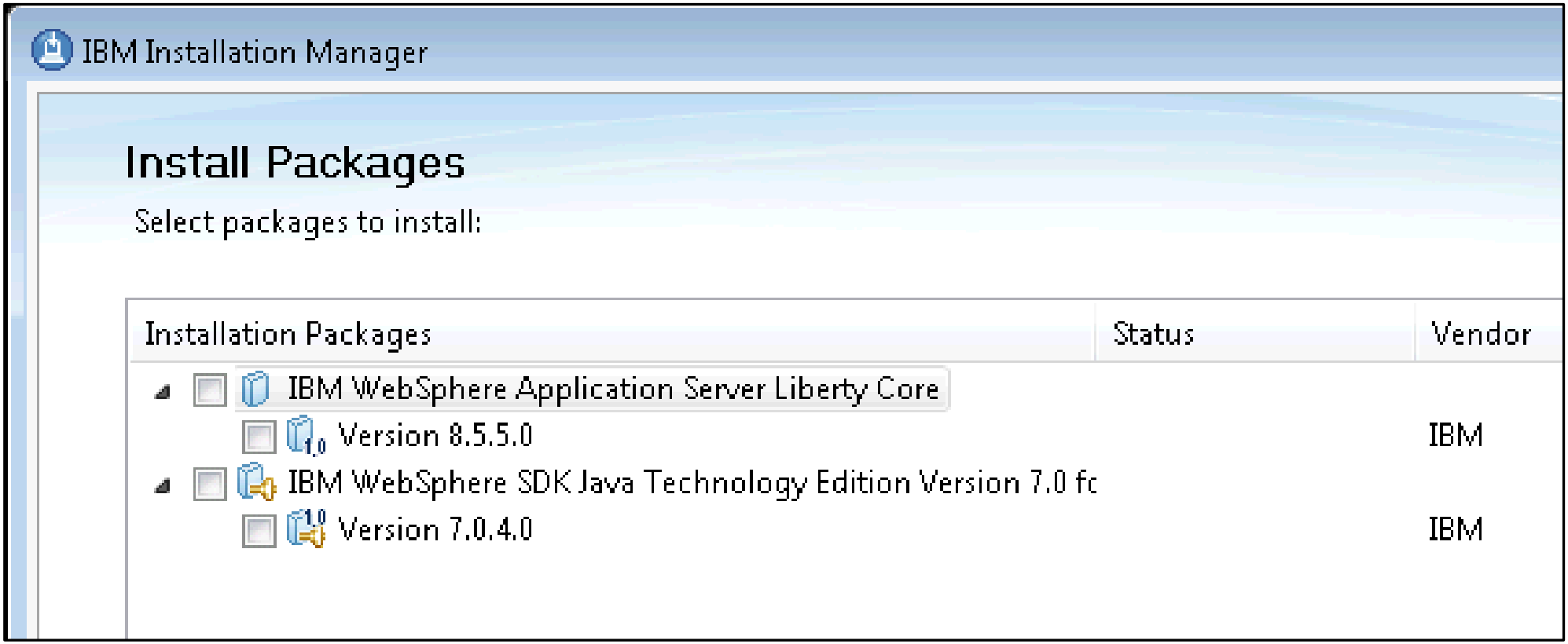
Where to get the Liberty profile runtime environment

- The runtime is available through:
  - IBM Installation Manager (as part of WebSphere V8.5)
  - Developer tools when creating a server
  - Download from `http://www.wasdev.net`



Installing the Liberty profile

- Installing the Liberty profile runtime environment
  - Use developer tools such as WebSphere Application Server Developer Tools, IBM Assembly and DeployTools, or Rational Application Developer
  - Use the IBM Installation Manager
  - Extract from a compressed archive (`java –jar wlp-8500.jar`)



Creating a Liberty profile server

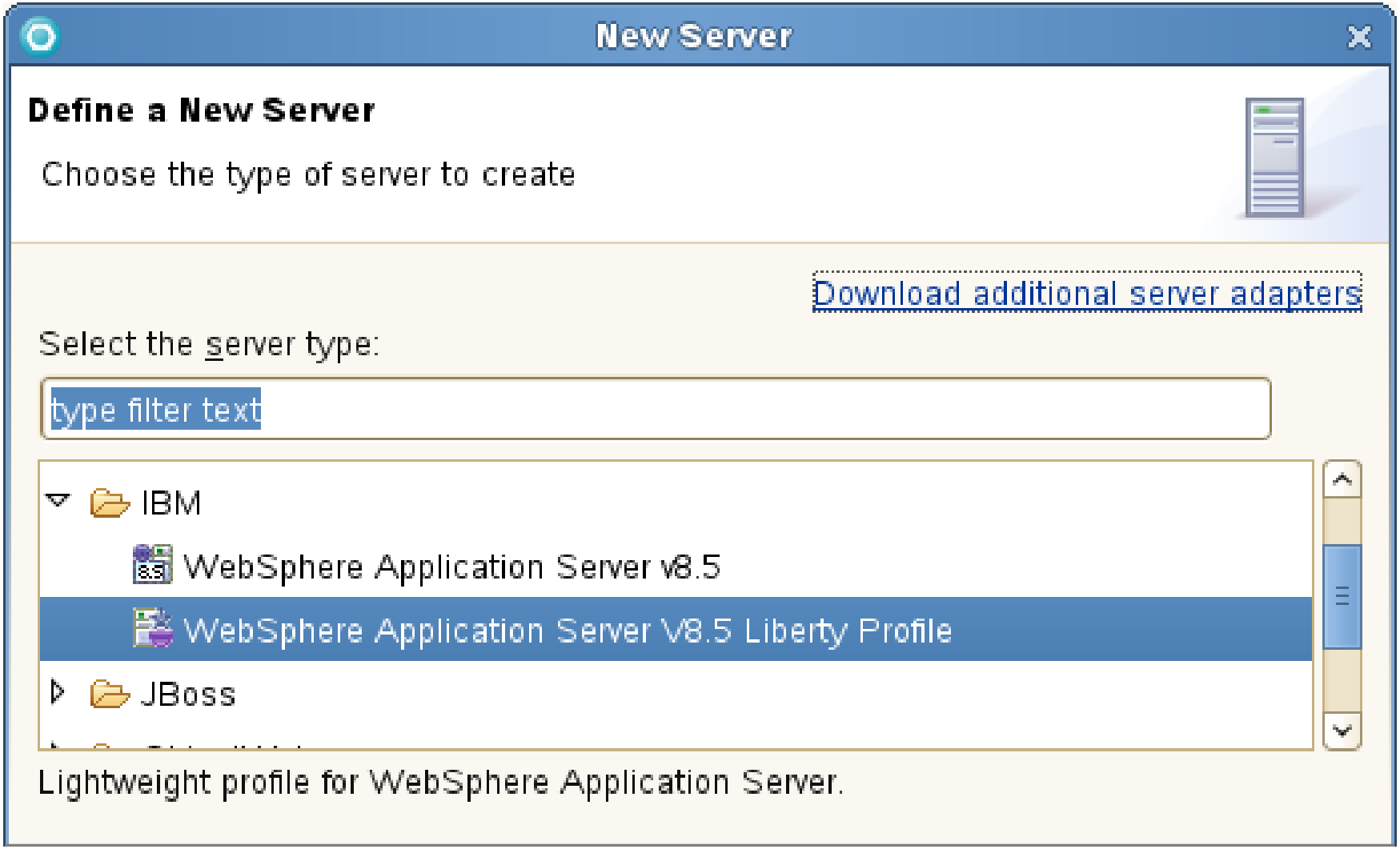
- Creating a server can be done quickly
  - Using the command line
  - Using developer tools (WebSphere Application Server Developer Tools, IBM Assembly and DeployTools, Rational Application Developer)

- From the command line:

```
Terminal
File  Edit  View  Terminal  Tabs  Help
was85host:/opt/IBM/wlp/bin # ./server create server2
Server server2 created.
was85host:/opt/IBM/wlp/bin #
```

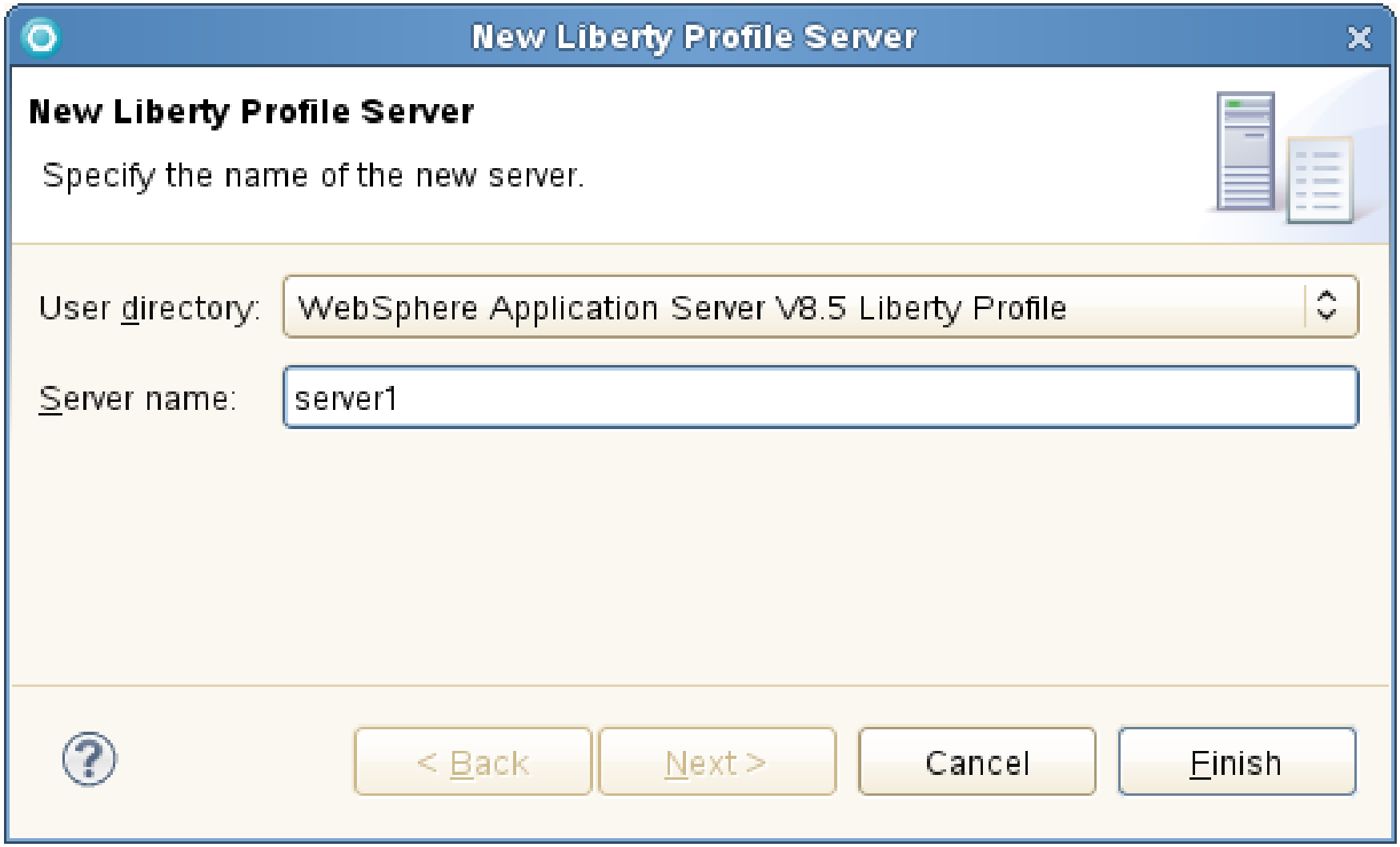
Creating a server with developer tools (1 of 3)

- Select the type of server



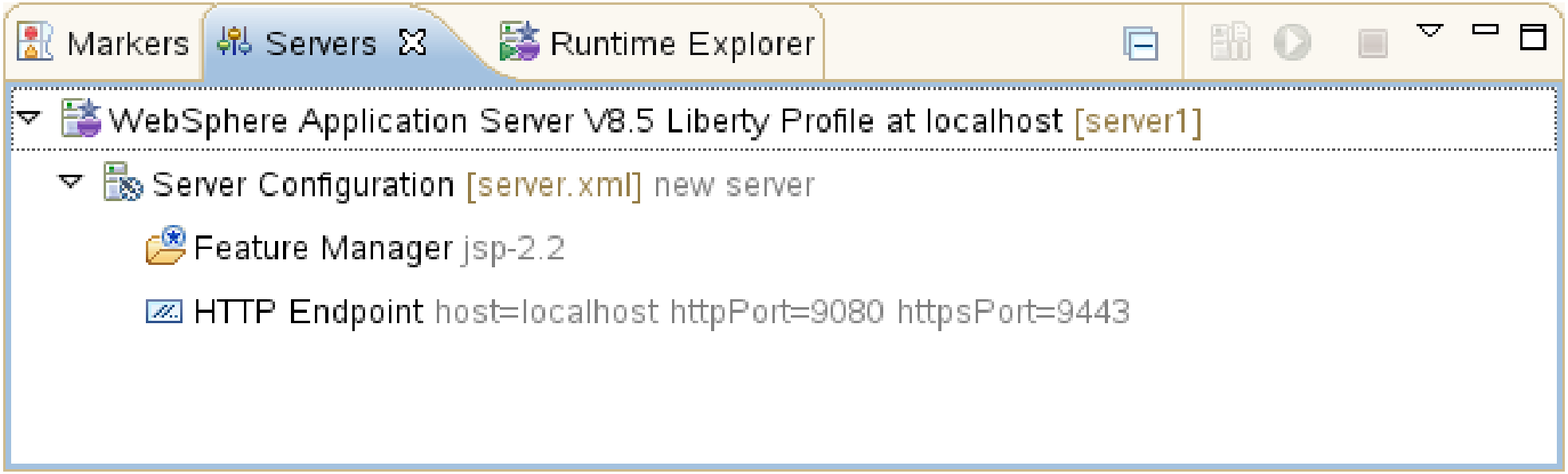
Creating a server with developer tools (2 of 3)

- Specify a server name and click Finish



Creating a server with developer tools (3 of 3)

- Server1 is created and is now listed in the Servers view
- Expand the server entry and you see the Server Configuration
- Configuration for a new server consists of:
  - The jsp- 2.2 feature
  - The HTTP Endpoint definition (9080 and 9443 are default ports for all new servers)



Configurations

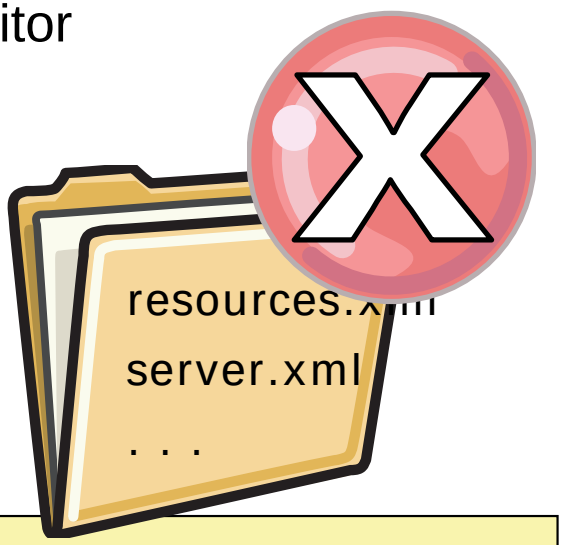
Simplified server configuration

- No need for administrative console, wsadmin, or enhanced EARs
  - These tools are not supported
- Configuration in XML files
  - **Simplest case:** one XML file (`server.xml`) for all server configuration
  - Editable within the developer tool or by using a text editor
  - Exportable, shareable, and versionable

server.xml

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>localConnector-1.0</feature>
        <feature>jdbc-4.0</feature>
    </featureManager>

    <httpEndpoint host="localhost" httpPort="9080"
                  httpsPort="9443" id="defaultHttpEndpoint"/>

    <applicationMonitor updateTrigger="mbean"/>

    <application id="HelloWorld" location="HelloWorld.war"
                 name="HelloWorld" type="war"/>
</server>
```

Liberty server configuration



Traditional WebSphere Application Server

Simplified configuration: server.xml

```
<server>
    <featureManager>
        <feature>jsp- 2.2</feature>
        <feature>jdbc-4.0</feature>
    </featureManager>

    <logging traceSpecification=
       "webcontainer=all=enabled:*=info=enabled" />

    <application name="tradelite" location="tradelite.war" />

    <dataSource jndiName="jdbc/TradeDataSource">
          <properties.derby.embedded
  databaseName="${server.config.dir}/tradedb"/>
    </dataSource>
</server>
```

Features control which capabilities (OSGi bundles) are installed in the server

Singleton configurations specify properties for a runtime service like logging

Instance configurations specify multiple resources like applications and data source definitions

Any of this configuration could be put into a separate XML file and included in this master configuration file by using the **include** element

Flexible configuration

- Shareable configuration snippets

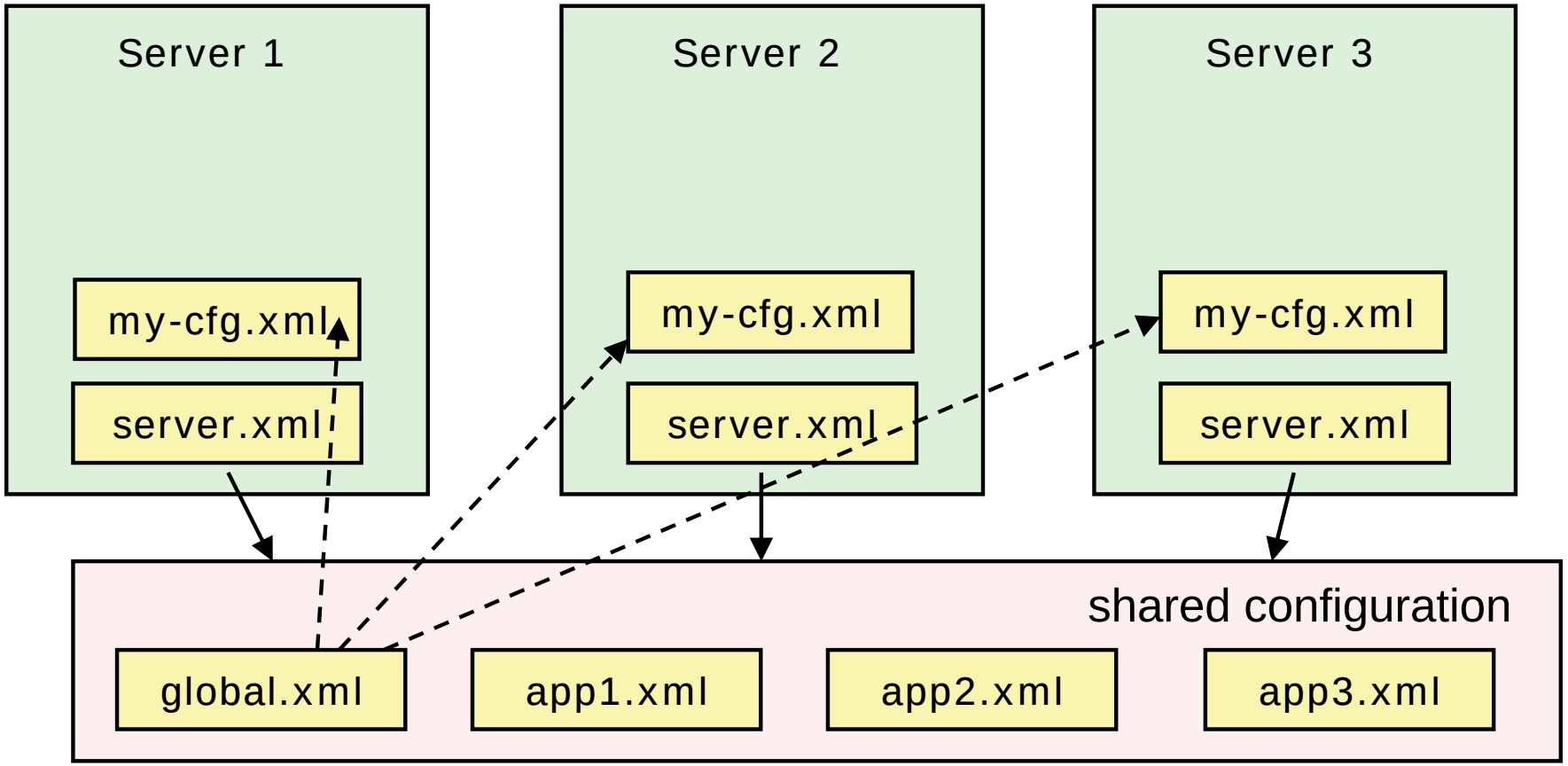
```
server>
    ...
  <include location="http://cfgserver/global.xml/>" />
  <include location="${shared.config.dir}/datasource.xml" />
<server>
```

server.xml

- Configurations can be divided into components at any level of granularity
  - From a single file to several
- Can use developer tools to associate configuration snippets with a server configuration
- Visualization through developer tools provides a single logical view
- **Team development:** keep the application and configuration components together

Shared configurations

- With shared configuration and includes, it is possible to create powerful topologies
  - Configuration files can be places in the server configuration directory or the shared configuration directory (or can be stored anywhere)

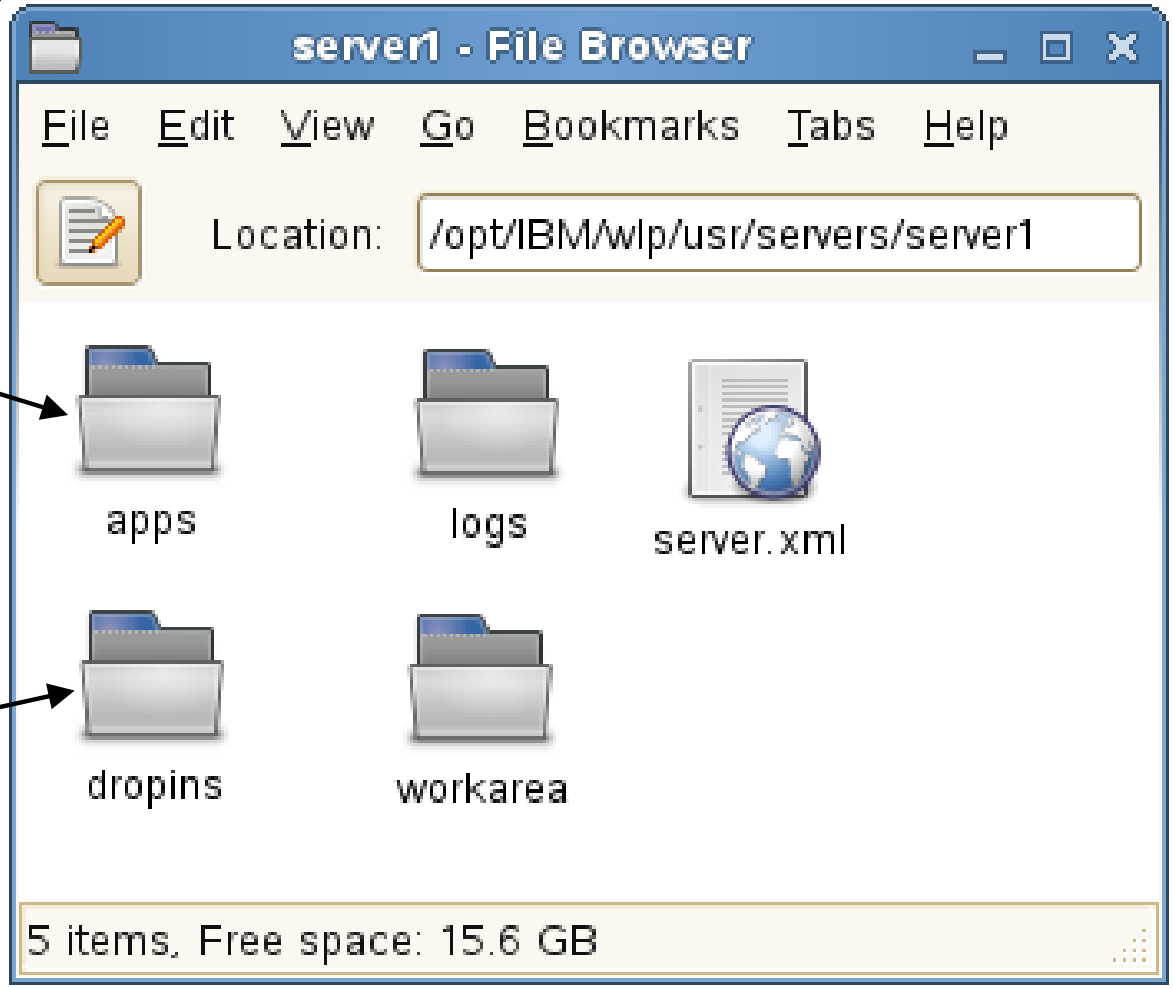


Application deployment

- Applications are deployed by using:
  - Monitored directory (`dropins`)
  - Configuration (`server.xml`)
  - Developer tools

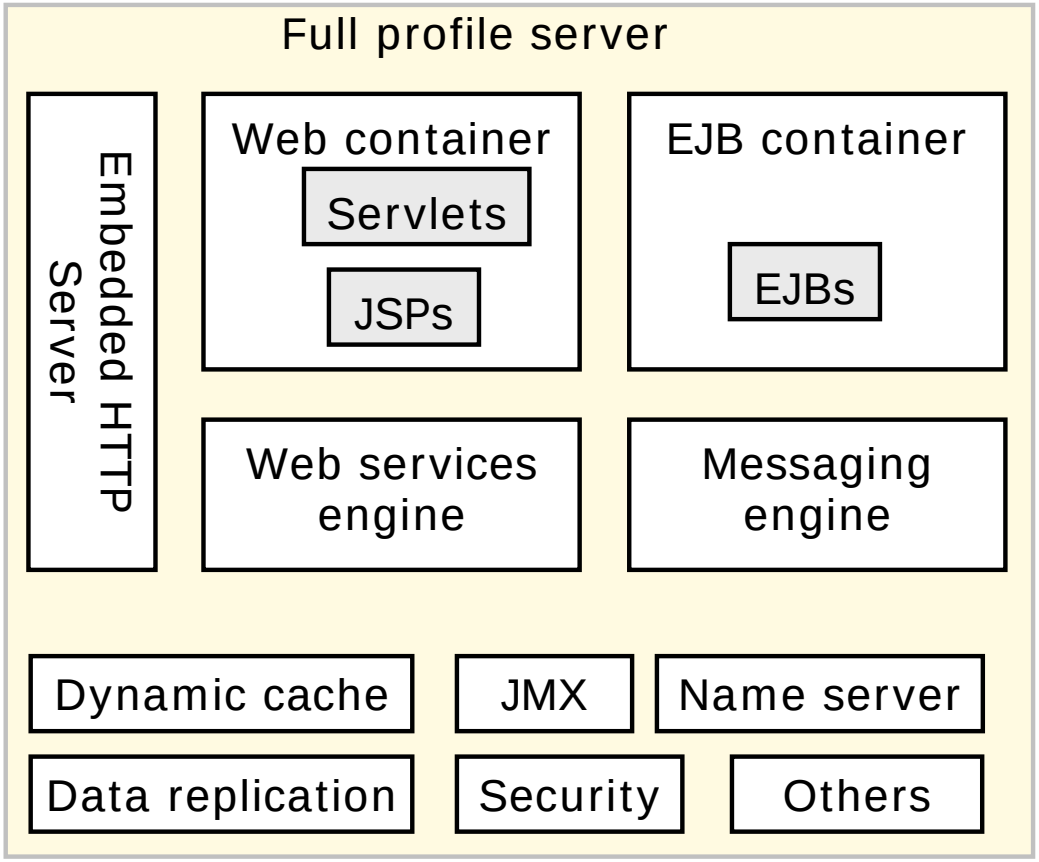
Configured applications go here (location can be configured)

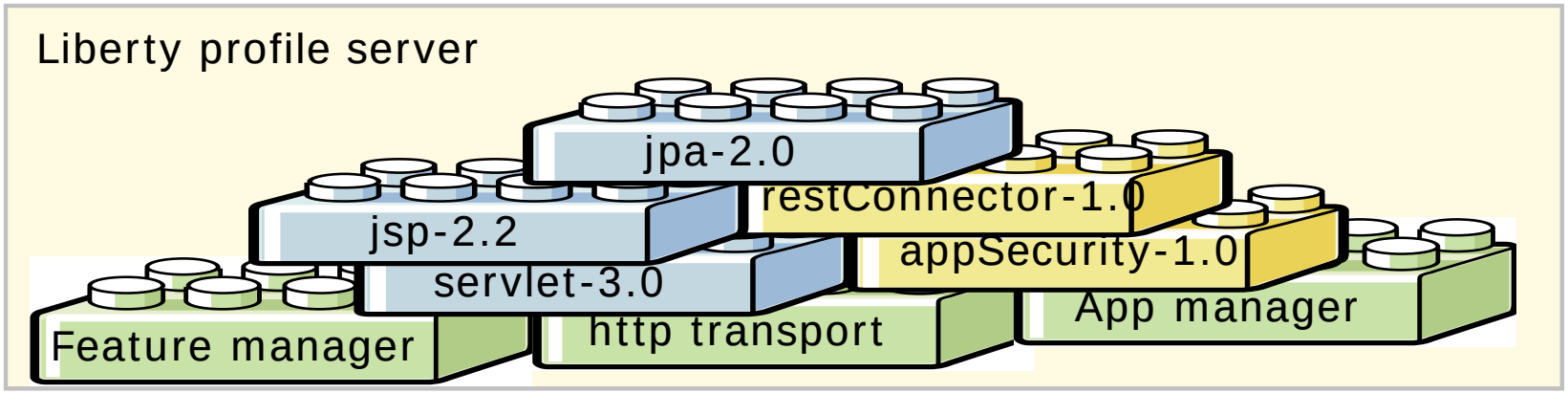
Monitored directory (location can be configured)



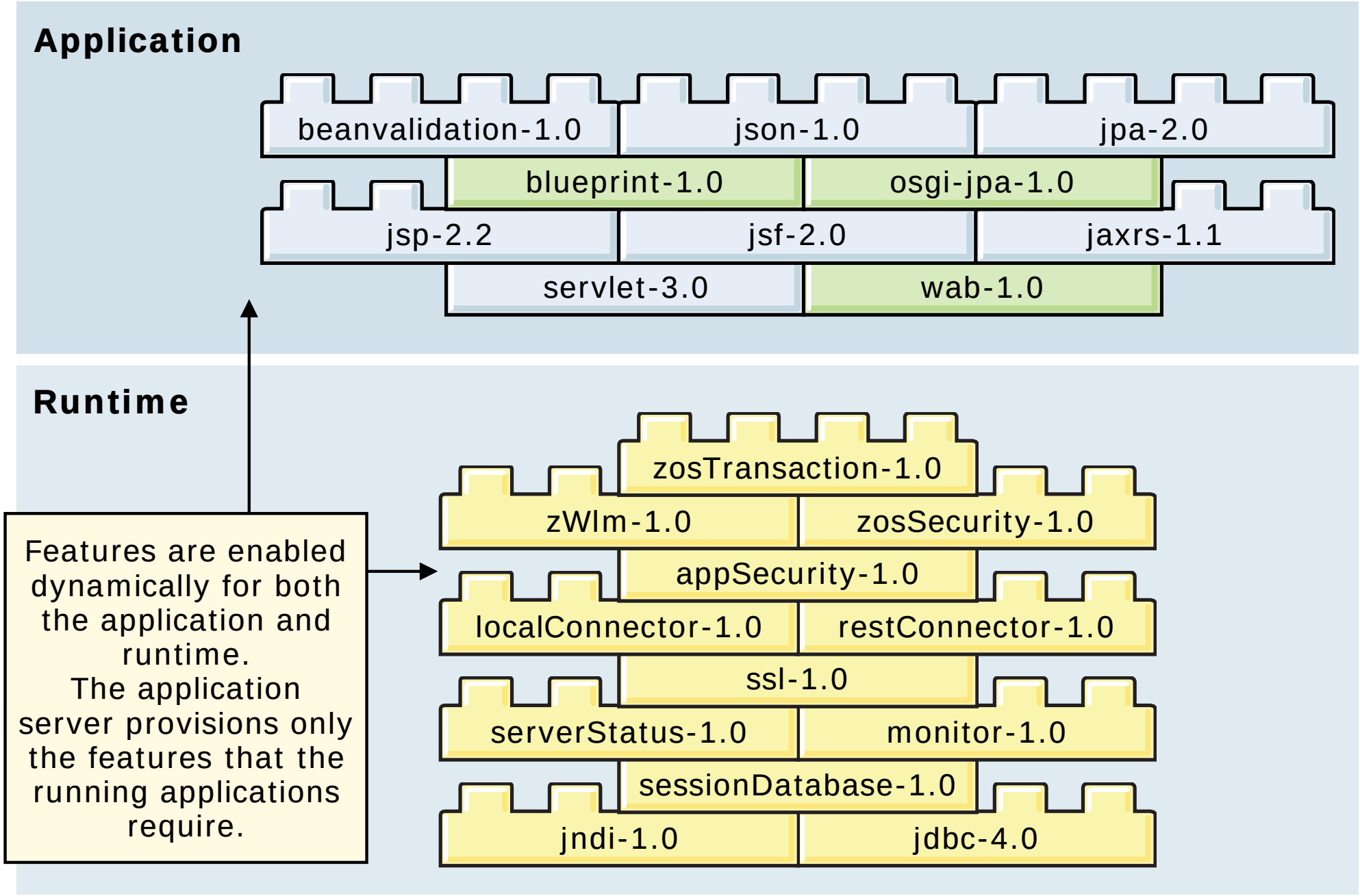
Highly composable runtime that is based on features

- Full profile includes everything
- Liberty profile includes only those features you add
  - Improved performance
  - Faster server starts



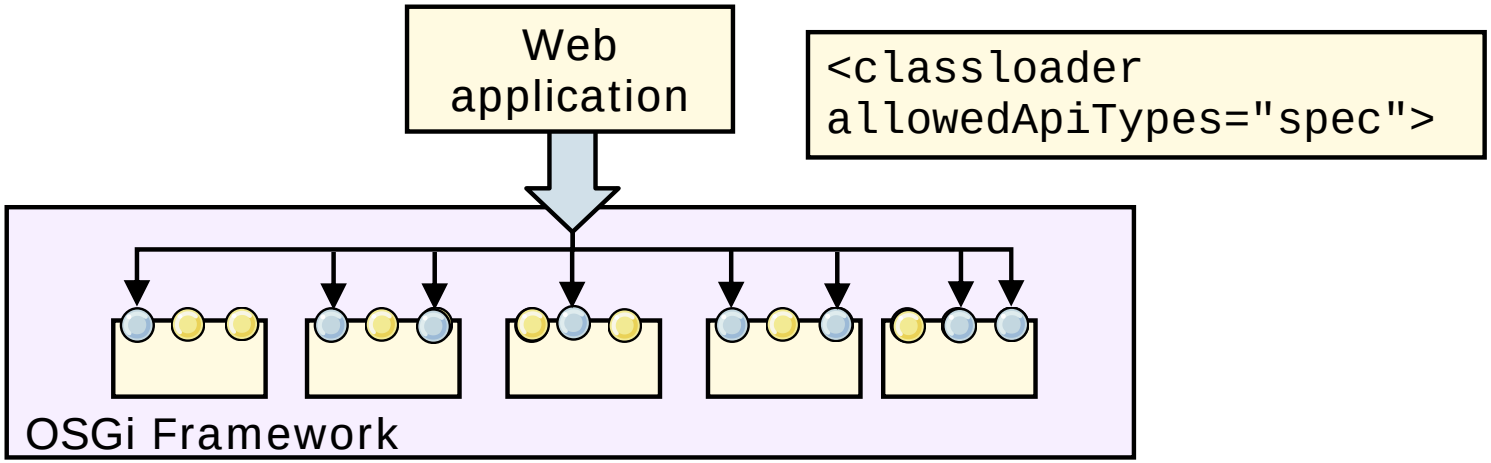


Dynamic enablement of feature sets



Class visibility

- Full profile server makes runtime classes visible to applications
- Liberty profile hides runtime classes from applications
- Applications can use open source APIs without the runtime interfering
- Three types of API
  - **`spec API`**: APIs defined by an external standards group ( )
  - **`ibm-api`**: Value add APIs provided by IBM ( )
  - **`third-party`**: APIs provided by open source projects
- By default only `spec` and `ibm-api` are visible to applications
  - third-party can be added by using the `classloader` element



Shared libraries

- Associated with applications
- Move common libraries out of the WAR files

```
<library id="libs">
   <fileset dir="${shared.resource.dir}/libs"
            includes="*.jar"/>
</library>
```

- Share classes between applications

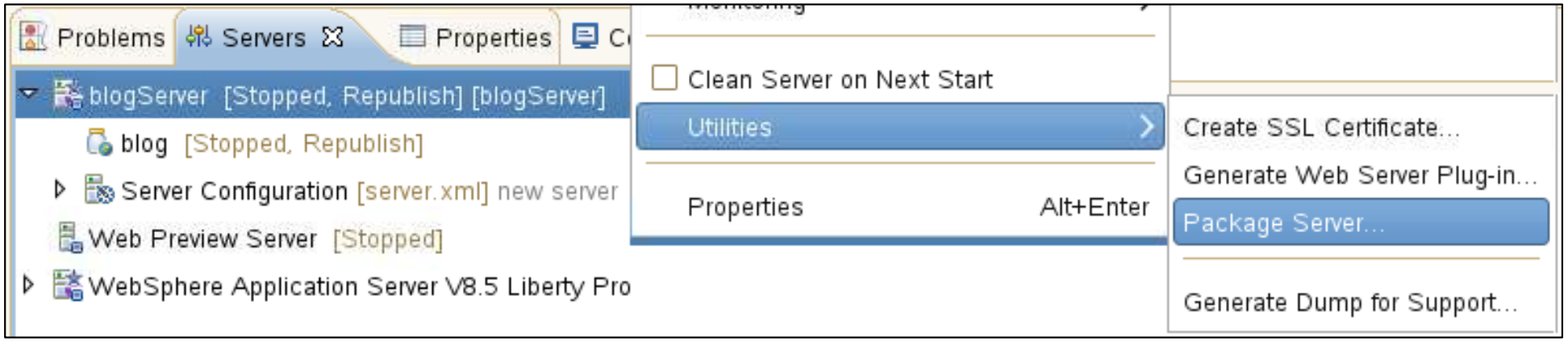
```
<application location="snoop.war">
    <classloader commonLibraryRef="libs" />
</application>
```

- Or have an instance per application

```
<application location="snoop.war">
    <classloader privateLibraryRef="libs" />
</application>
```

Packaging an application for deployment

- Package an archive of a configured Liberty server along with its applications
  - Directly from Eclipse environment
  - Resulting compressed file can be copied to integration or production environment and uncompressed



Liberty profile server command-line tools

- For server operations outside of the developers tools, there is a command-line program to manage the lifecycle of server instances and also package it for deployment:
  - Create <server_name>
  - Start and stop <server_name>
  - Package <server_name>
  - Status <server_name>

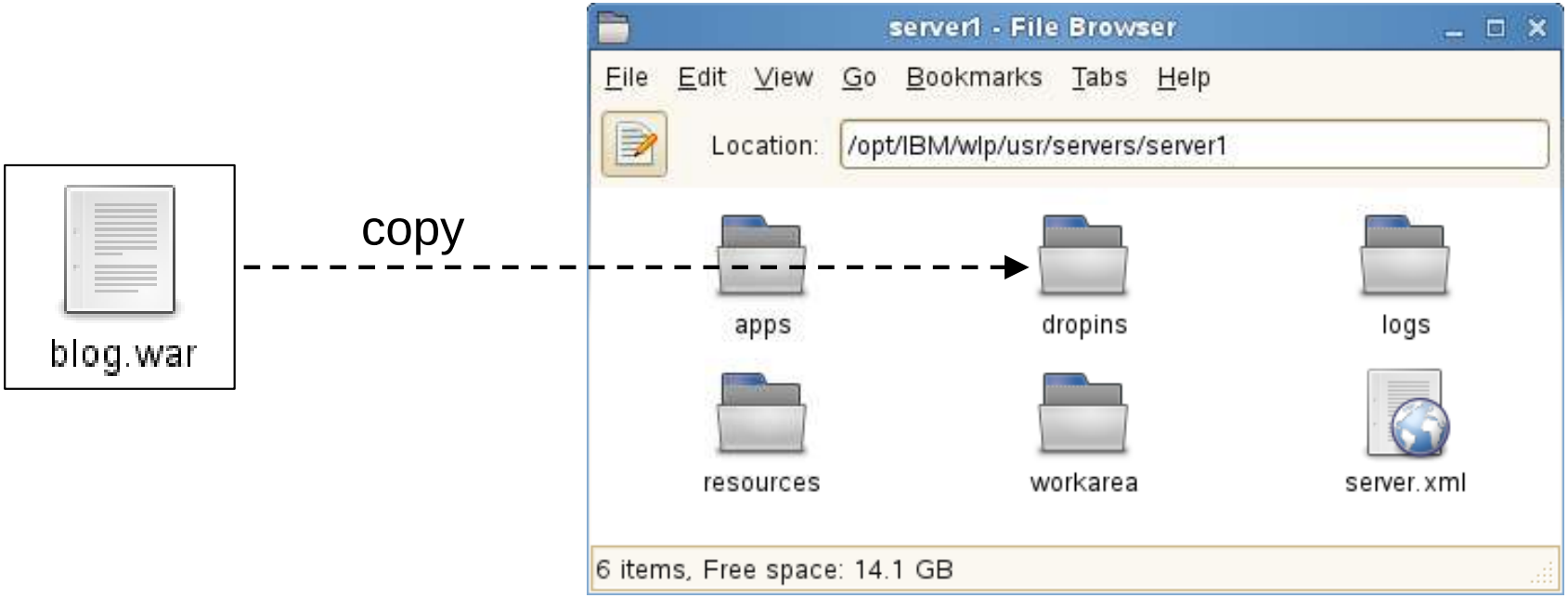
Terminal

File Edit View Terminal Tabs Help

```
was85host:/opt/IBM/wlp/bin # ./server package server2
Server server2 package complete in /opt/IBM/wlp/usr/servers/server2/server2.zip.
was85host:/opt/IBM/wlp/bin #
```

Deploying an application by using the drop-ins directory

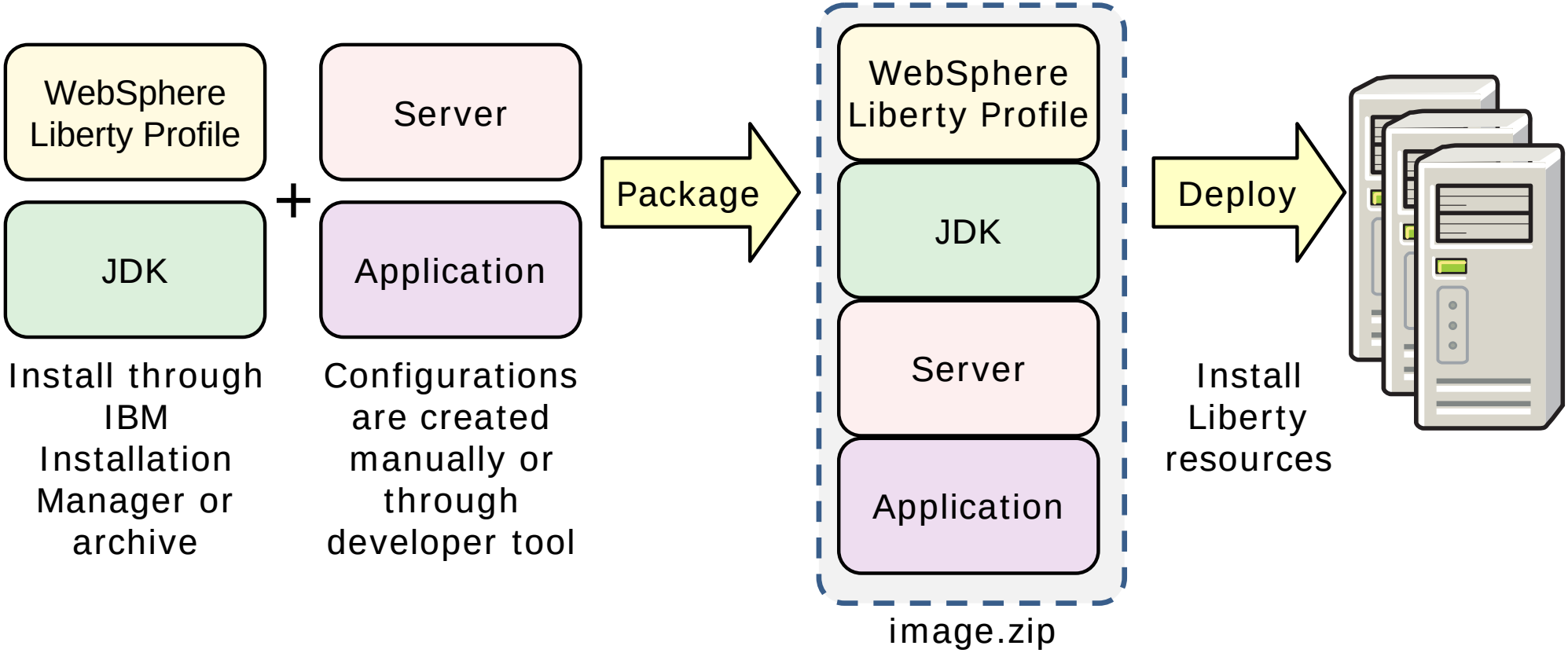
- A monitored directory can be used to deploy applications
  - Copy the application file into the **`dropins`** directory
  - The server installs and starts the application



- Removing the file from the dropins directory automatically uninstalls the application

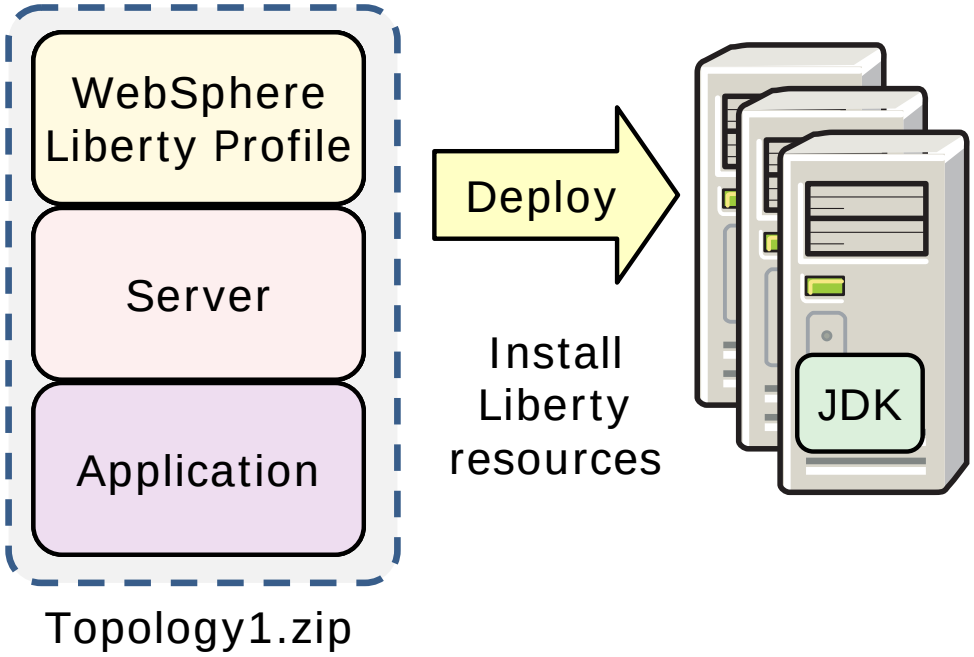
Creating the production image

- Runtime installation by using Installation Manager or archive
- Configurations are created manually or by using developer tools
- Package Liberty profile resources into a compressed file
- Deploy to production environment manually or by using the job manager



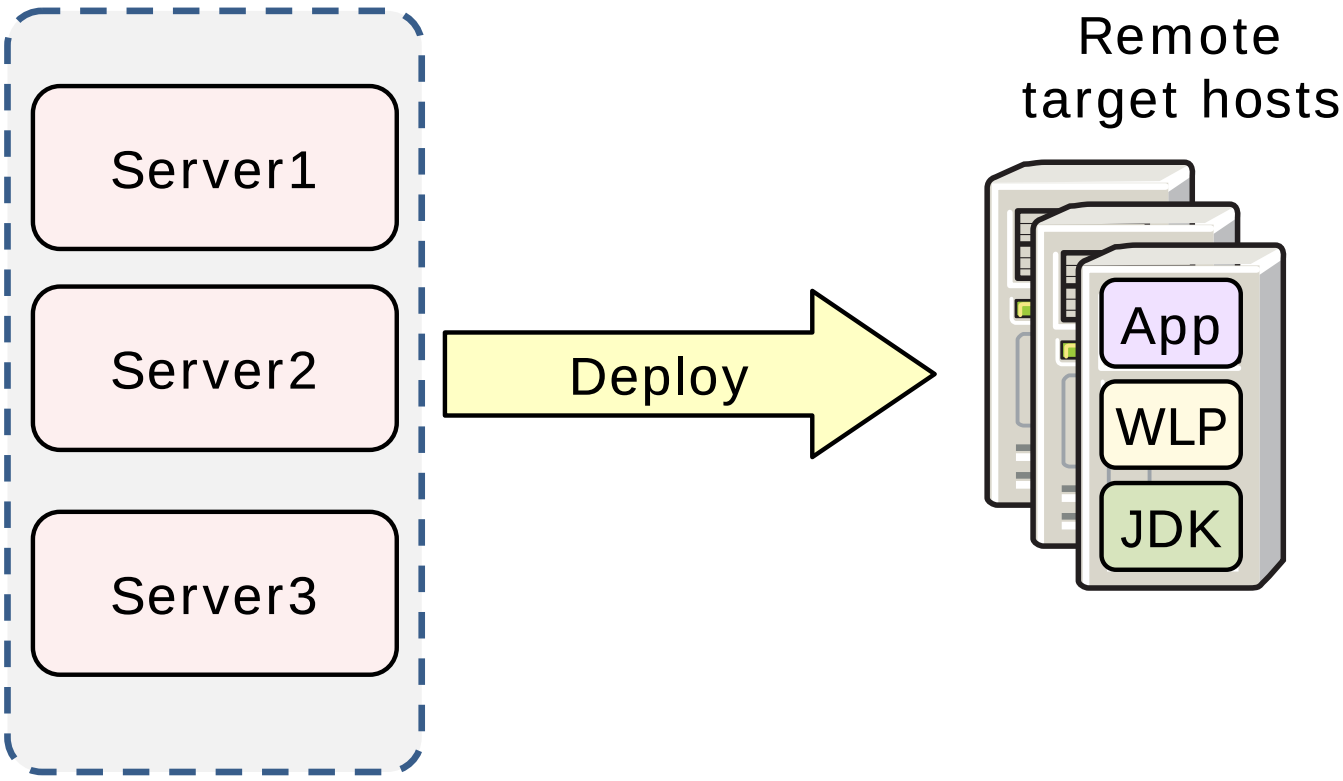
Deploying Liberty topologies

- Choose which parts you must deploy
  - Only the application
  - Only the Liberty profile server
  - The Liberty profile runtime and the server
  - Any combination
- Example: A self-contained topology in which the compressed file contains everything but the JDK
  - The SDK is preinstalled on each host



Example: A shared topology

- A shared topology where each compressed file contains only the Liberty profile server definition
  - Applications are predeployed as read-only and shared across different servers
  - The Liberty profile and JDK are preinstalled and shared by different servers



Topology2.zip

WLP = WebSphere Liberty Profile

Security

Liberty profile security

- All opened ports are local host only
- No remote management by default
- Seamless transition when enabling security
- Three key security-related features

Feature	Description
ssl-1.0	Includes the SSL-specific code
appSecurity-1.0	Includes all the security services (authentication, registry, authorization) and web-specific security code
zosSecurity-1.0	Includes the SAF registry and authorization code

Enable SSL

- Add the SSL feature and provide the keystore password

```
<featureManager>
    <feature>ssl-1.0</feature>
</featureManager>

<keyStore id="defaultKeyStore" password="{xor}DFoKyp="/>
```

- Certificate is generated at server startup
- `securityUtility` can be used to generate a self-signed certificate

Terminal

File Edit View Terminal Tabs Help

```
was85host:/opt/IBM/wlp/bin # ./securityUtility createSSLCertificate
--server=server2 --password=web1sphere
Creating keystore /opt/IBM/wlp/usr/servers/server2/resources/securit
y/key.jks

Created SSL certificate for server server2

Add the following lines to the server.xml to enable SSL:

    <featureManager>
        <feature>ssl-1.0</feature>
    </featureManager>
    <keyStore id="defaultKeyStore" password="{xor}KDo9biwvNzotOg=="
/>
```

Advanced SSL

- Configure per endpoint SSL configuration

```
<featureManager>
    <feature>ssl-1.0</feature>
</featureManager>

<keyStore id="myKeyStore" password="{xor}DFoKyp="
          location="${server.config.dir}/mykeystore.p12"
          type="PKCS12"/>
<keyStore id="myTrustStore" password="{xor}DFoKyp="
          location="${server.config.dir}/mytruststore.p12"
          type="PKCS12"/>
<ssl id="mySSLConfig" keystoreRef="myKeyStore"
                      trustStoreRef="myTrustStore"/>

<httpEndpoint id="defaultHttpConfig">
  <sslOptions sslRef="mySSLConfig"/>
</httpEndpoint>
```

User registries

- Three types of registry
  - Basic XML-based registry (Not for use in production)
  - LDAP registry
  - SAF registry (SAF is available only for z/OS)
- Used for application and JMX security
  - One registry per server

Basic user registry

- Configured in `server.xml`
- Passwords can be encoded
  - The `securityUtilities.sh/bat` command encodes passwords
  - Developer tools encode automatically

```
<server>

    <featureManager>

        <feature>appSecurity-1.0</feature>

    </featureManager>


      <basicRegistry realm="basicRealm">
         <user name="bob" password="{xor}CDo9Hgw=" />
         <group name ="group1">

           <member name = "bob"/>

          </group>

       </basicRegistry>

</server>
```

LDAP user registry

- Authenticate by using an LDAP server
- Supports: Microsoft Active Directory, IBM Lotus Domino, Novell eDirectory, IBM Tivoli Directory Server, Sun Java System Directory Server, Netscape Directory Server, IBM SecureWay Directory Server

```
<server>

    <featureManager>

        <feature>appSecurity-1.0</feature>

    </featureManager>


    <ldapRegistry host="myldapserver.ibm.com"
         port="389" baseDN="o=ibm,c=us"
         ldapType="IBM Tivoli Directory Server" />

</server>
```

SAF user registry

- Authenticate by using an SAF on z/OS

```
<featureManager>
    <feature>zosSecurity-1.0</feature>
</featureManager>

<safRegistry id="saf"/>
```

Authorization

- Security role mappings are defined in
  - Server configuration
  - `ibm-application-bnd.xml`

```
<application location="secureapp.war">
   <application-bnd>
      <security-role name="users">
         <user name="fred"/>
         <group name="userGroup"/>
      </security-role>
   </application-bnd>
</application>
```

- SAF

```
<safAuthorization id="saf" />
```

Liberty administrative security

- One administrator role
- One user registry for applications and administrators
- Simple configuration for a single administrator user

```
<quickStartSecurity userName="bob"
                    userPassword="{xor}Lz4sLCgwLTs"/>

<keystore id="DefaultKeyStore"
                password="{xor}DFoKyp="/>
```

- But still easy for multiple users

```
<administrator-role>
    <user>fred</user>
    <group>administratorsGroup</group>
</administrator-role>
```

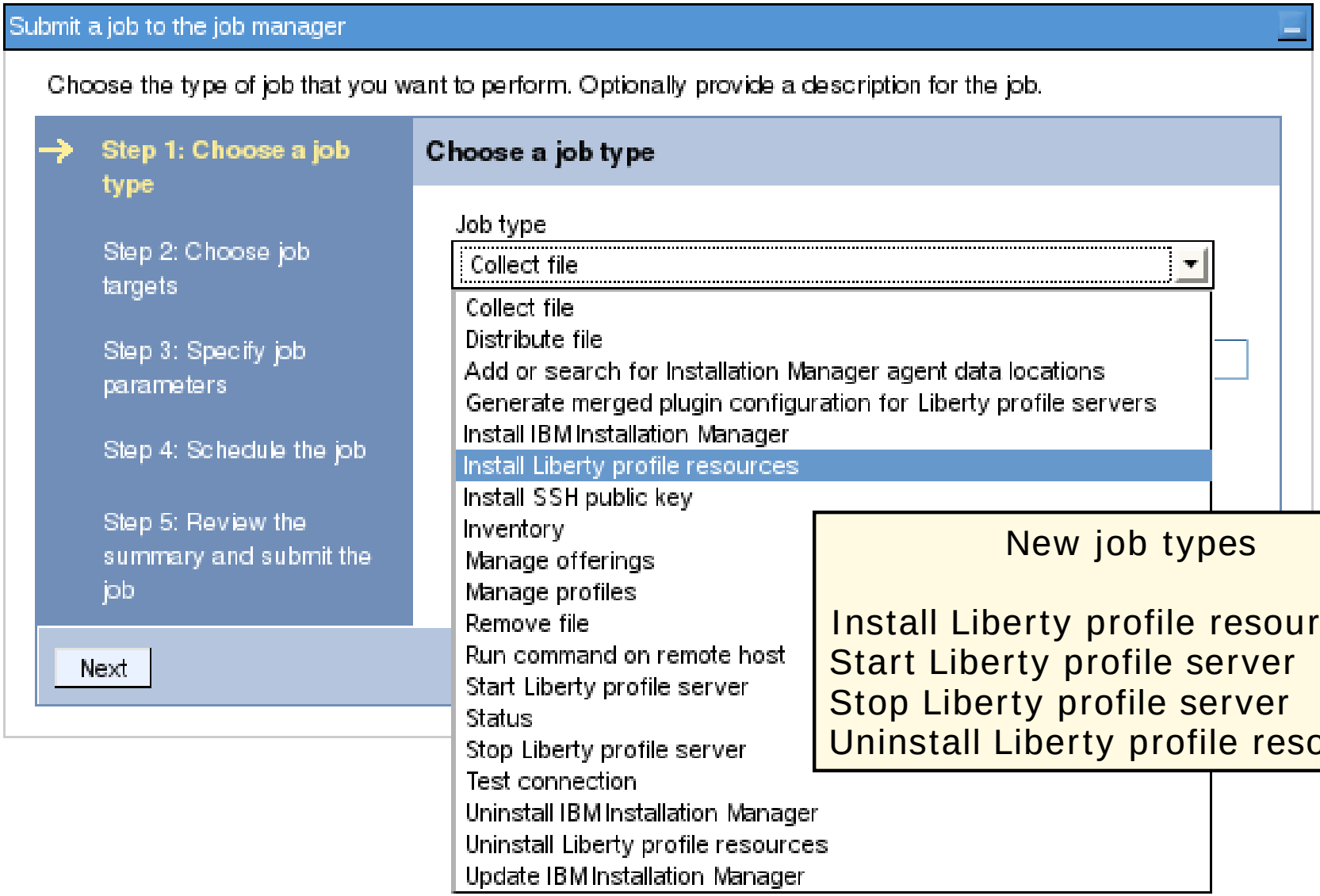
Using the job manager to manage Liberty profile servers

Administering Liberty profiles by using the job manager

- Centralized remote management of Liberty profiles
- Uses remote target host capability of job manager
- Existing job types apply to Liberty profiles
  - Inventory and status
- New job types
  - Install or uninstall Liberty profile resources
  - Start or stop Liberty profile servers
  - Generate merged plug-in configuration
- New resource types
  - Project
  - Runtime
  - Server
  - Application
  - JDK

Job manager Liberty profile jobs

- From the job manager administrative console, click **Jobs > Submit**

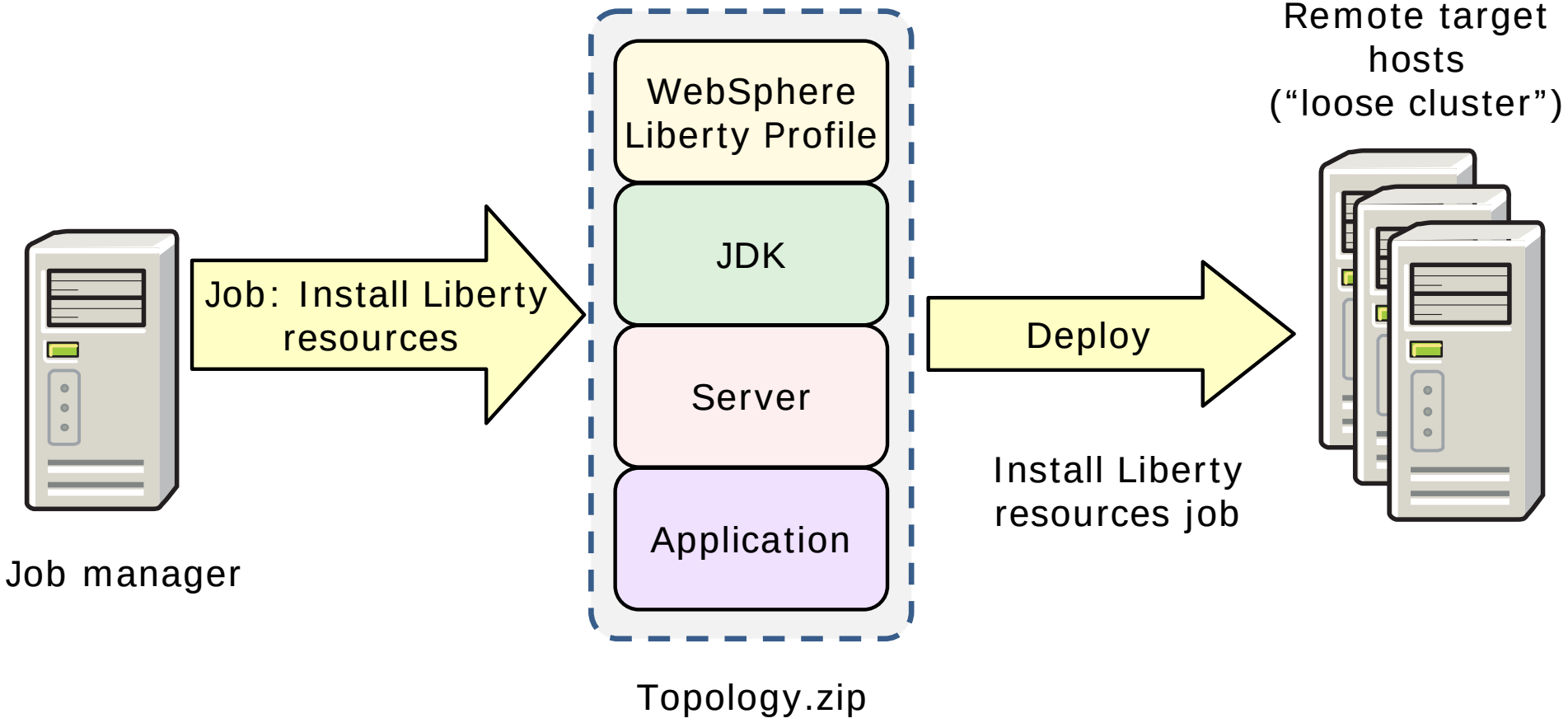


New job types

Install Liberty profile resources
Start Liberty profile server
Stop Liberty profile server
Uninstall Liberty profile resources

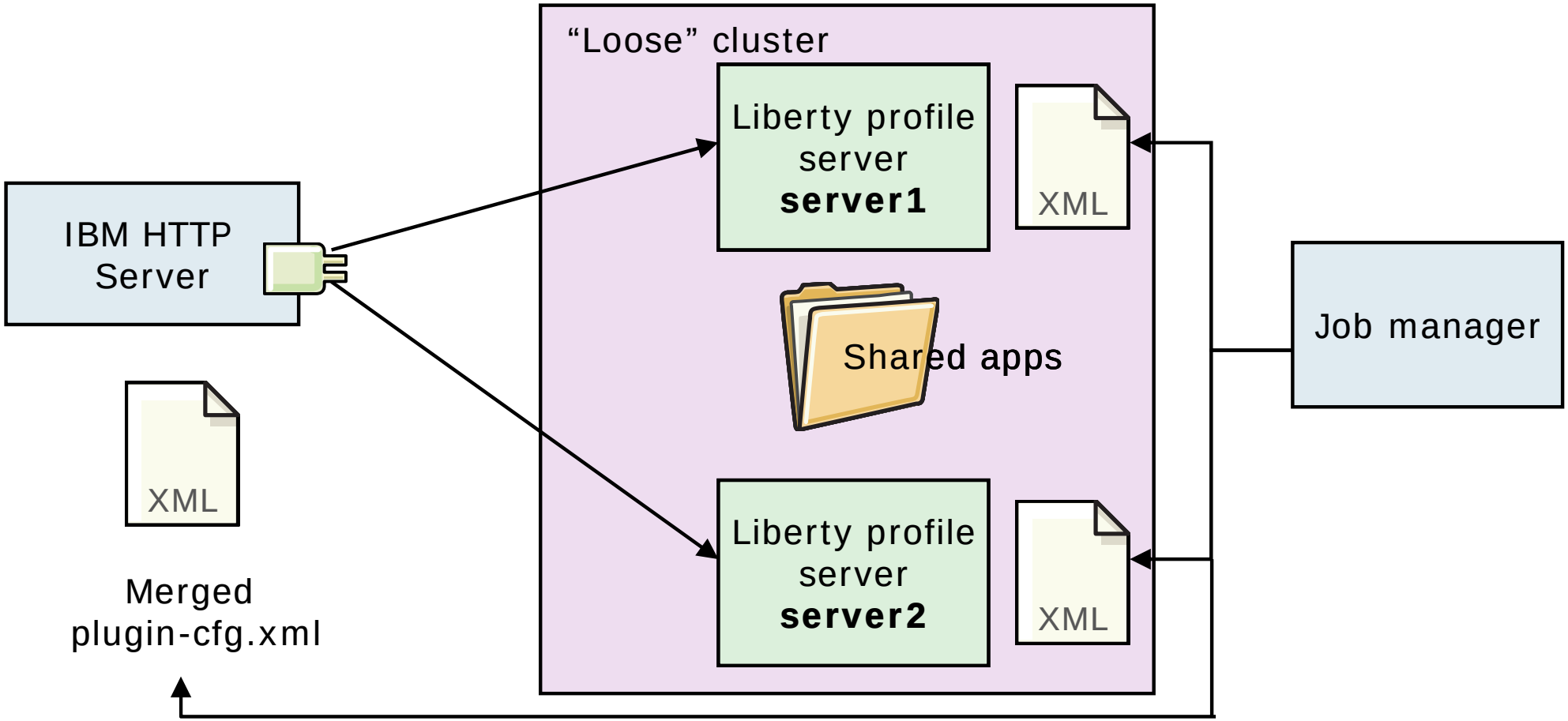
Deploying a self-contained topology

- A self-contained topology is packaged as a compressed file that contains following resources: a Liberty profile runtime, a JDK, a Liberty profile server, an application
- The job manager can deploy the topology to multiple target hosts



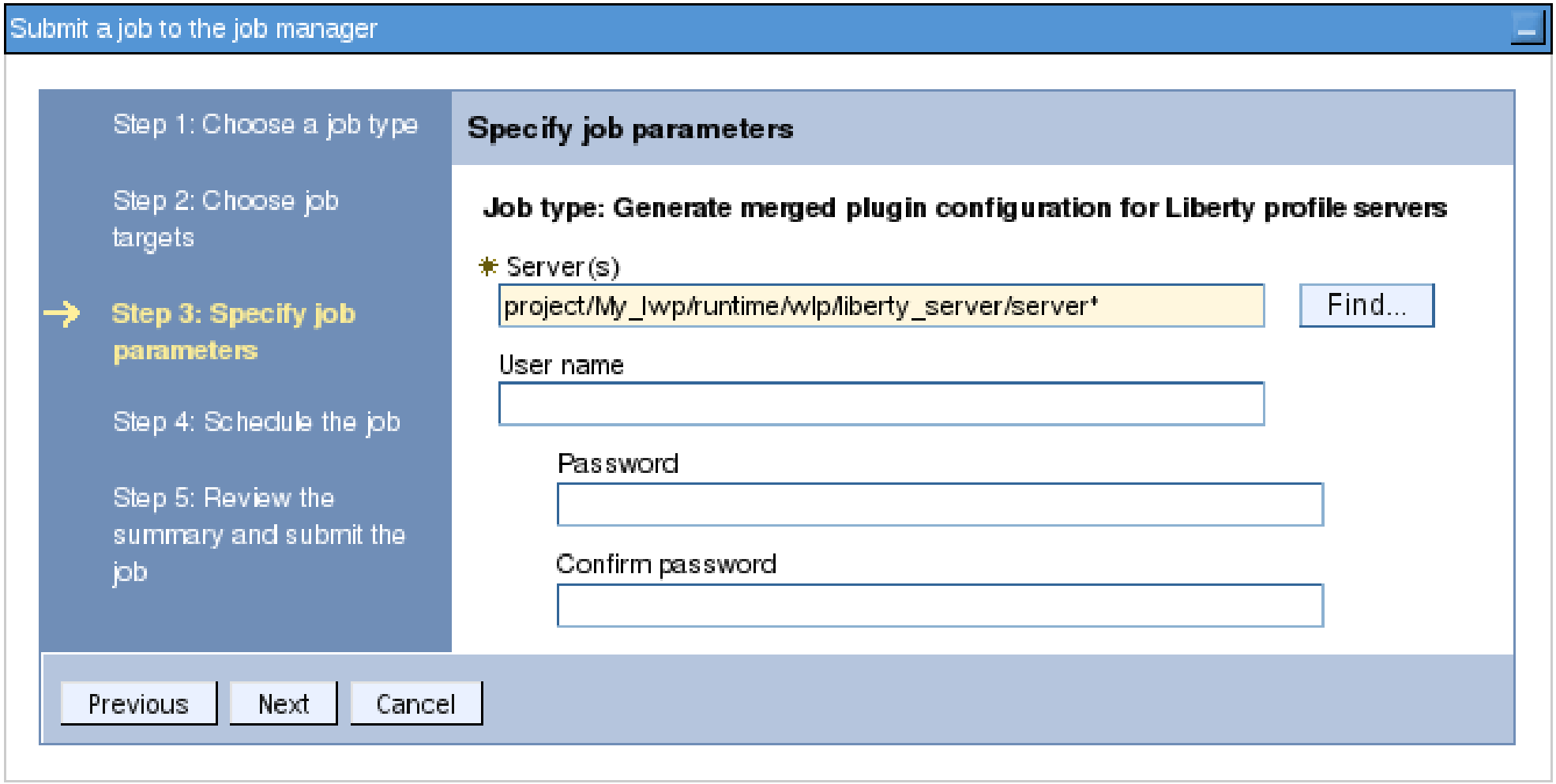
HTTP request routing

- For workload management (WLM), the HTTP server plug-in must be able to read a `plugin-cfg.xml` file that contains plug-in configuration from all the Liberty profile servers
- Job manager can create a merged plug-in configuration file



Job type: Generate merged plug-in configuration

- Use the **Find** facility to specify the resource ID for one server
- Use pattern matching to specify multiple servers
- Servers must be started



Merged plugin-cfg.xml

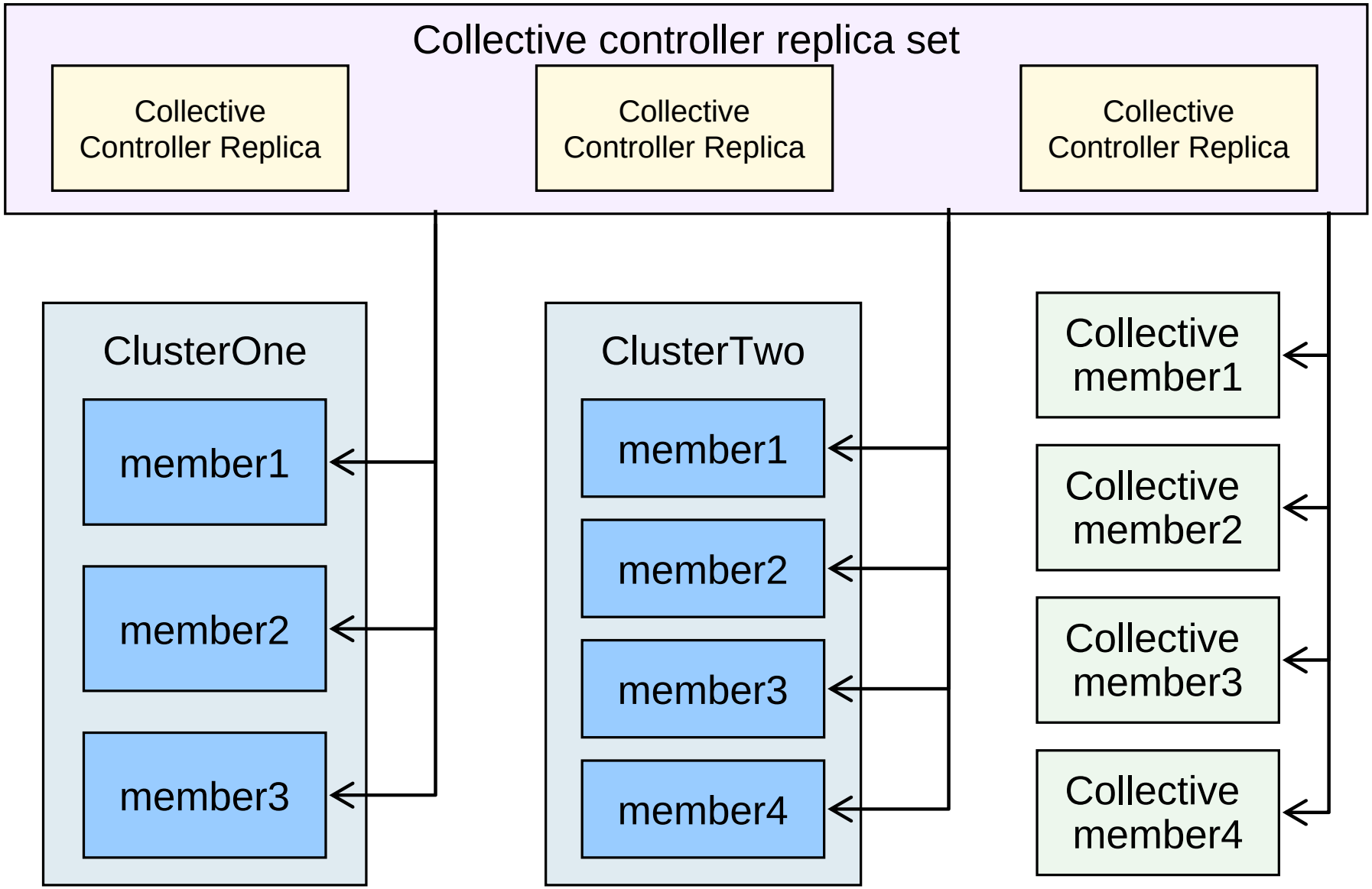
```
<ServerCluster CloneSeparatorChange="false" GetDWLMTable="false"
        IgnoreAffinityRequests="true" LoadBalance="Round Robin"
        Name="Shared_2_Cluster_0" PostBufferSize="64" PostSizeLimit="-1"
        RemoveSpecialHeaders="true" RetryInterval="60">
        <Server CloneID="e9da8378-7892-4993-8ce0-7e838e6bf6d2"
            ConnectTimeout="0" ExtendedHandshake="false"
            MaxConnections="-1" Name="default_node_defaultServer0_1"
            ServerIOTimeout="900" WaitForContinue="false">
            <Transport Hostname="was85host" Port="9041" Protocol="http"/>
        </Server>
        <Server CloneID="306b1428-4119-4b9d-bae0-e7cc0cc5e0a8"
            ConnectTimeout="0" ExtendedHandshake="false"
            MaxConnections="-1" Name="default_node_defaultServer0_0"
            ServerIOTimeout="900" WaitForContinue="false">
            <Transport Hostname="was85host" Port="9040" Protocol="http"/>
        </Server>
  . . .
</ServerCluster>
```

Liberty collectives and clusters

Liberty collectives and clusters

- A Liberty collective is like a traditional WebSphere cell
  - More loosely coupled
- A Liberty collective controller is like a deployment manager
  - Scalable and highly-available
- A Liberty cluster is like a traditional WebSphere cluster
  - Any Liberty profile servers can be added and removed from Liberty clusters

Example of server clusters and a Liberty collective



Administering Liberty profiles in collectives

- A Liberty collective is an administrative domain for Liberty profiles
  - Analogous to a traditional WebSphere cell
- Standards-based administration API
  - Built on JMX (MBeans)
  - Works with common tools (Jconsole, Jython, and so on)
- Loosely-coupled
  - No centralized master configuration
  - No nodeagents
- Management server is called a collective controller
  - Analogous to a deployment manager
  - Application servers cache sparse configuration data and state in the controller
  - Application servers own their own configuration
- Scalable and resilient
  - Can have multiple controller servers for the same collective
  - Replicate member state and configuration data between collective controllers

Liberty profile collective controller and collective members

- Collective controller
  - A Liberty server with the collectiveController feature enabled
    - Network Deployment Liberty Profile only
  - Caches the configuration and state of collective members
  - Send commands to collective members
  - Multiple collective controller servers can manage the same collective

- Collective member
  - A Liberty server with the collectiveMember feature enabled
    - All Liberty Profile editions
  - Sends configuration data and state to one of the collective controllers
  - Accepts commands from collective controller servers

Liberty clusters

- A Liberty profile can be configured into a server cluster for application high availability and scalability.
- A server cluster is a logical grouping of related servers
- The server cluster feature clusterMember-1.0 is only available in the Network Deployment Liberty Profile
- A server cluster can only be defined within a Liberty collective
- All server cluster members must be members of the same collective
- One collective can have multiple server clusters

Unit summary

Having completed this unit, you should be able to:

- Describe the characteristics and architecture of the Liberty profile
- Install the Liberty profile runtime environment
- Create a Liberty profile server by using developer tools and command-line utilities
- Describe the configuration features for a Liberty profile server
- Use flexible configuration and shared libraries
- Deploy applications by using a monitored directory
- Deploy applications by using developer tools
- Package an application and Liberty profile runtime
- Describe the process for enabling security for a Liberty profile server
- Use the job manager to manage Liberty profile servers
- Describe the characteristics of a Liberty collective
- Describe the characteristics of Liberty profile server clusters

Checkpoint questions

1. (True or false) The Liberty profile is freely available.

2. (True or false) Liberty profile servers support everything that the full profile WebSphere Application Server supports.

3. (True or false) Liberty profile servers can be created only with a developer tool such as WebSphere Application Server Developer Tools.

4. (True or false) The job manager can be used to install Liberty profile resources on remote target hosts.

Checkpoint answers

1. True: The Liberty profile is freely available for developers.

2. False: Liberty profile servers support only a subset of what the full WebSphere Application Server supports.

3. False: A server create command can also be used.

4. True

Exercise 15

Working with the Liberty profile

Exercise objectives

After completing this exercise, you should be able to:

- Use IBM Assembly and Deploy Tools to install the Liberty Profile Runtime Environment
- Start and stop a Liberty profile application server by using the command line and through IBM Assembly and Deploy Tools
- Deploy a simple application by using IBM Assembly and Deploy Tools
- Deploy an application by using the dropins directory
- Deploy an application with a data source
- Configure SSL for a Liberty profile application server
- Configure a user registry for a Liberty profile application server
- Configure application security for a Liberty profile application server
- Use flexible configuration to create shared configurations
- Configure a Liberty profile server to generate a plug-in configuration file