Goals

Application Versioning allows multiple versions of the same application to exist in a GlassFish domain. It provides the developer and the administrator extensions to the relevant existing tools and commands to deploy, view, and manage multiple versions of applications.

At most one version of an application may be enabled on any target.

To the extent possible, implement the changes to the exposed interfaces so that the same changes could be used as-is or with minimal revision to support runtime versioning.

Any published interface that supports the application operations that are affected by app versioning (e.g., deploy, undeploy, enable, disable, create/delete-app-ref, get-client-stubs) will support app versioning. This includes the CLI, the GUI, ant tasks, and the callable DeploymentFacility.

Provide explicit versioning, in which users specify a version for an app.

Non-goals

Automatic versioning, in which users specify no version information but GlassFish automatically supplies successive automatically-generated version identifiers in response to successive deployments of an application.

Runtime versioning, in which multiple versions of an app can be enabled on a single target. This would require added runtime support in the containers and in naming which is not needed for app versioning alone.

Functionality

1. A *version identifier* is [A-Z|a-z|0-9]+ with dot, underscore, and hyphen permitted when used with at least one other character.

2. A *version expression* is either a version identifier or a wildcard expression that contains one or more * characters as the wildcard. GF applies the associated operation to all versions of the app which match the expression.

3. Version identifiers and expressions appear in public interfaces (commands, API arguments, etc.) as suffixes to application names, introduced by a semi-colon (;). For example, myApp;beta or inventory;RC*.

This approach preserves existing APIs (e.g. JSR-77, JSR-88, AMX) while allowing them to take advantage of the versioning feature. Places where an application name appears in the v2 APIs will accept a name with version expression in v3.

4. These version-aware operations accept an optional version *identifier*: deploy, enable, create-app-ref, and get-client-stubs.

5. These version-aware operations accept an optional version *expression*: undeploy, disable, delete app ref.

6. App versioning is, from the user's perspective, optional. The user will experience the v2 behavior for all app operations he or she invokes without specifying a version.

7. GlassFish imposes no limit on the number of versions of an application the user may create, aside from limitations in disk space for storing the application and its configuration information.

8. The *default* version of an application on a target is the version that GlassFish creates if the user does not specify a version as he or she deploys the app. Any version-aware operation that does not specify a version operates on the default version. This preserves the v2 user experience.

9. The *current* version of an app on a target is the version most recently specified in any operation that can change the configuration. In this context the configuration operations are any of the version-aware operations listed above except get-client-stubs.

   Operations can refer to the current version by using a “naked” semi-colon – a ; with no following version expression. This distinguishes the reference to the current version from a reference to the default version (which is the complete absence of the version expression, including the absence of the introducing ; character). Note that the current version can be the default version.

10. GlassFish does not support versioning for in-place directory deployed apps *when deployed from the same directory*. GlassFish does allow the user to use in-place directory deployment and versioning together so long he or she deploys different versions of the app from different directories. If a user has in-place deployed an app with a version from a directory GlassFish reports an error if he or she attempts to in-place redeploy that app from the same directory specifying a different version.

    This restriction exists primarily because GlassFish does not maintain copies of directory-deployed applications in its own directory structure. When the user changes a directory deployed app and redeploys it, the contents corresponding to the older version would no longer be accessible to GlassFish so GF cannot support versioning behavior. If GF were to make a copy of successive versions of the user's directory contents in its own area in order to support versioning then much of the performance advantage of in-place deployment would be lost.

11. When the user deploys a versioned app with --enable=true (the default) to a target or enables a versioned app on a target, GlassFish automatically disables any currently-enabled version of that app on that target.

12. When the user creates an app ref with --enable=true for a versioned app for a target, GlassFish disables any app ref involving that same target for all other versions of the same app.

13. When a user identifies a specific version of an app, no other versions of the app are affected with the exception that deploying with --enable=true or enabling a specific version or app ref automatically disables other versions or app refs for other versions on the affected target.

14. In v2 GlassFish enforces some restrictions among different applications, such as requiring each application to have a unique context root and to use distinct JNDI names. These restrictions remain among different applications in v3 but do not apply to different versions of the same application. GlassFish v3 permits different versions of the same application to use the same context root and the same JNDI names. This presents no runtime conflict because at most one of the versions can be enabled at a time.

    Open issue: Are there other similar restrictions currently imposed among applications that should be relaxed among versions of the same app?

15. Database tables created or deleted as part of deployment and undeployment are global resources and so cannot be qualified by an application version. The documentation for app versioning will need to describe this.GlassFish supports version rollback through the enable command. If the user deploys a new version he or she can rollback simply by enabling a different version. Because enable automatically disables any enabled version of the app, enable accomplishes rollback.

16. **Open Issue: Implications for sun-resource.xml? Esp. global resources.**

Unversioned vs. versioned apps and backward compatibility

Although the same operations apply to versions and unversioned apps, the effects can be slightly different. So we need to control carefully (and document) these differing effects. This includes if and how an app can change from unversioned to versioned and back again.

We want users to have a consistent experience in v3 when they use a sequence of operations they have used from v2. We also want a user to be able to deploy an app as unversioned and later change his or her mind and deploy versions of the app. Further, we want to provide a convenient and predictable way for users who take advantage of versioning to refer to the current version.

The concepts of the *default* version and the *current* version satisfy these needs.

A user who never specifics a version identifier or expression sees the v2 behavior. Behind the scenes GlassFish operates on the default version of the application, which in this case is the only version of the application that ever exists.

If a user deploys an application one or more times, always using a version identifier, GlassFish will reject attempts to refer to the default version because there will be no default version. The user would be able to refer to the *current* version using “naked ;” notation (; with no string after it).

Wildcarding using only “*” will match the default version, as will the current version notation (if in fact the default version happens to be the current version).

Note that the current version need not actually be in an enabled state on that target. For example, if the user deploys version A (--enable=true) then version B (--enable=false), version A remains the current version. If the user then enables version B, version B become the current version. If the user then disables B, B remains the current version because it was the version most recently operated on by a configuration command. A subsequent operation that specifies the naked semi-colon operates on version B.

This approach gives users the same experience they had with v2, while allowing v3 users who use versioning a convenient and deterministic shorthand for referring to the current version of an app on a target.

Implementation

Domain.xml

This proposal requires no new changes in the layout. The version information is embedded in the module name.

General Implementation Notes

1. It will be essential that containers use the correct APIs to locate resources and classes within the apps and NOT rely on assumptions about where the files will reside. The set of files that should be active for a given app will change over time as the user deploys new versions and/or disables and enables versions.

2. The DOL will eolve so that the getName() method for application and application-ref returns only the name part of the application, excluding the version. To support logic that needs to use the version, the DOL will need to add getVersion and setVersionto application and application-ref. The empty string represents the default version.

3. GlassFish v3 will store application bits in its repository at

.../applications/${app-name}

for the default version and at

.../applications/${app-name}-${version-ID}

for explicit versions.

The semicolon character is valid in file names for non-Windows systems and for the Windows NTFS file system. It is not valid for the Windows FAT file system, and while it is unlikely that many users would be using FAT we may as well avoid the problem entirely. The dash is valid in filenames on all of our supported operating systems and file systems.

Some Open Issues

1. Should GF even provide implicit autoversioning? If so, should a single application be subject to both autoversioning and explicit versioning? **Resolved: do not provide autoversioning.**

2. Should we impose a limit on the number of versions for an app, configured by domain? **Resolved: no.**

3. How should the rollback feature be exposed?

   Should undeploying the enabled version of an app automatically re-enable the preceding version? Should the undeploy, disable, and delete-app-ref operations accept an optional --rollback option?

   Resolved: The enable operation accomplishes rollback.

4. Ideally the versioning logic would enforce that versions of an app are of the same module type. But there is not currently an unambiguous way to use the domain.xml information to deduce a module's "type." This is related to backward compatibility with v2, which rejects redeployment of an app if the module type changes from the earlier deployment of the app, since that logic would need the same information about a module's type.

   Is there a reliable and efficient way to continue this restriction and extend it to versions?

5. Will ws-related tools and functions need to know about version identifiers? **Resolved: Not explicitly. They will need to accept a version identifier and its introducing semicolon as part of the app name.**

6. The list-components command should display all versions of each app. Should the command also accept an optional version expression to restrict the output to versions which match the expression? **Nice to have. Not essential.**

7. What other attributes in <application> should migrate to <application-version>? **Resolved: domain.xml will not include <application-version>.**

8. What does support for OSGi bundles mean with versioning?

9. Are there aspects other than context root and JNDI names that should either be permitted to be the same in different versions of the same app, required to be the same, or required to be different?