Removing unpackWARs

This page is intended to collect the reasons for and against the removal of this feature. The main purpose is to review use cases for using unpackWARs to ensure that a suitable alternative would be available if the feature was removed but all arguments for and against the feature are welcome.

Background

Prior to Tomcat 8 unpackWARs=false did not result in the web application running entirely from the WAR. For performance reasons, any JARs in WEB-INF/lib were unpacked to the work directory and accessed from there. Tomcat 8 introduced a new resources implementation that did not perform this unpacking. This had a significant (e.g. 3x to 10x slower application start) performance impact - Bug 57251. One of the options suggested in the discussion on that bug was the removal of the unpackWARs feature.

Use Cases for unpackWARs=false

Simpler deployment while Tomcat is stopped

If unpackWARs=true and a WAR is updated while Tomcat is stopped, Tomcat will not realise that the unpacked directory structure is from an older version of the WAR and will continue to use it. The current work-arounds are:

- ensure that the exploded directory is deleted when the WAR is replaced
- deploy an exploded directory rather than a WAR
- use unpackWARs=false

The last of these work-arounds is often viewed as the simplest and would not be available if the unpackWARs option was removed. Work is in progress to handle this use case within the existing automatic deployment code so WARs updated while Tomcat is shut-down result in the old directory being removed and the new WAR expanded in its place.

Security - make appBase read-only

Note that this discussions assumes a pre-existing application or Tomcat vulnerability that permits writing files into Tomcat's appBase.

If the appBase is writeable by the Tomcat user and automatic deployment is enabled then it makes it easier for an attacker to deploy a malicious application by placing it in the appBase. Using unpackWARs=false and a read-only appBase defeats this particular attack vector in Tomcat 8. In Tomcat 7 the protection is provides is less than complete since the location where the JARs are copied to remains writable.

The removal of unpackWARs would effectively require the deployment of web applications as exploded directories if the appBase was to remain read-only to the Tomcat user.

An alternative to removing uppackWARs in this case might be to move the location of the unpacked WAR files: the appBase is still the *source* of all WAR data, but unpacked WAR files would be unpacked elsewhere -- such as into the work directory, etc. This would allow a read-only appBase and still allow unpackWARs=true. From a security point of view, the work directory itself is still vulnerable, but it would not be possible (given a pre-existing application or container vulnerability) for an attacker to deploy a completely new WAR onto the container.

Actually read-only filesystem

If the filesystem is actually read-only (or effectively so, where the effective Tomcat user has no file-write rights whatsoever), then unpackWARs must be false in order to deploy. If all JSPs are pre-compiled and logs are not written to disk, Tomcat should be able to run on a read-only filesystem.

Deploying an exploded WAR file with pre-compiled JSPs would also meet this requirement.

Testing - configuration with getRealPath() returning null

When unpackWARs="false" the SevletContext.getRealPath() method always returns null. This configuration can be used to test that a web application is programmed correctly and can function without relying on getRealPath() method.

Other use cases

Additional use cases welcome. Either directly on this page or via the users mailing list.