Resin Cookbook - Article: **Deployment**

Resin's .war application refers to it's ability to deploy Java web applications, connecting devices, people, and applications around the world. Deployment can be as simple as copying a .war file to a /webapps directory on a local machine, and as powerful as cloud deployment, archiving, staging, & rollback.

- cloud deployment: Resin's cloud deployment will distribute a new web-application to all servers in the cloud, using a transactional store to ensure consistency.
- activation control: deployment and activation can be controlled independently, allowing applications to be deployed and verified on all servers, and then activated at once or on a rolling-server basis.
- command-line: web-applications can be deployed to the cloud from the command-line for scriptable deployment.
- browser: for convenience, applications can also be deployed from a browser using the /resin-admin site.

Resin deploys a web-application to all servers in the cluster when you deploy using the command-line or the browser. This process happens automatically and does not require any configuration beyond Resin's normal cluster configuration.

This cloud deployment does not occur when you deploy using the filesystem by placing a .war in a /webapps directory. Cloud deployment only occurs on the command-line or browser.

Example:

unix> resinctl deploy test.war

Resin replicates the deployed application to all three triad servers, or to all available server if you have less than three servers. Any server beyond the triad will copy the deployed application from the triad. Normally, you don't need to be aware of the triad, except to make sure at least one of the first three servers is always available.

When you add a new dynamic server or restart a server, the server will check with the triad and update to the most recent copy of the application. The system is reliable if servers crash or even if a server crash or network outage occurs during deployment. Deployment is transactional meaning the new version is only activated when every file has been copied and verified on the server.

The internal data-directory layout of Resin is as follows:

```
resin-data/
  app-0/                    # each named server gets its own section
    .git/                   # web-app deployment .git repository
    distcache/              # distributed cache directory
      data.db               # distcache value data
      mnode.db              # distcache key/value binding
    log/                    # health system log
      log_data.db           # log entries
      log_name.db           # log names
    stat_data.db            # health statistics data
    stat_name.db            # health statistics names
    tmp/                    # temp swap directory
      temp_file
    xa.log.a                # XA log
```

When deployed, resin-data/ will receive a copy of the .war file. It uses a hash of this .war file to determine when a web-app should be re-deployed. webapps/ will receive the actual files from the .war

Basic Deployment Methods

For a simple deployment, you can copy a .war archive file containing your application to a webapps/ directory. Resin will detect the .war archive, expand it, and start serving requests.

The webapps directory is configured and enabled by the <web-app-deploy/> tag in the resin.xml.

Example: web-app-deploy in resin.xml

```
<resin xmlns="http://caucho.com/ns/resin">

  <cluster id="">
    ...
    <host id="">

      <web-app-deploy path="webapps"
                      expand-preserve-fileset="WEB-INF/work/**"/>

    </host>
  </cluster>
</resin>
```

The <expand-preserve-fileset> attribute lets you keep files for a redeployment, which can improve restart times. Normally, Resin will delete all files for an update, including the compiled files for JSP, which forces Resin to recompile JSP files, even if they haven't changed in the update. Expand-preserve-fileset let’s Resin only recompile the JSP files that have changed.

Command-Line Deployment

Command-line deployment uses the same resin.xml <web-app-deploy> configuration as a webapps/ deployment and expands the archive to the same directory. Instead of looking for the .war file in a directory, Resin will look in an internal repository using the web-app's identifier.

The default deployment is to the default host with the war's name as a prefix. Both can be changed with deploy options.

> usage: resinctl deploy [--options] <war-file>

deploys an application

where command options include:

```
  --host <host>        : virtual host to make application available on
  --stage <stage>      : stage to deploy application to, defaults to production
  --version <version>  : version of application formatted as <major.minor.micro.qualifier>
  --m <message>        : commit message
  --timeout <timeout>  : timeout for long deploys
  --name <name>        : name of the deployment context
```

Deployment Expansion

The <web-app-deploy> controls the expansion based on web-app ids like "production/webapps/default/test" which is the same as the web-app identifier. The deployment expansion process looks like the following:

1. Look in the internal repository for "production/webapps/default/test" (uploaded by command-line). If an archive exists in the internal repository, skip to step #4.
2. If that fails, look for a webapps/test.war
3. If the webapps/test.war exists, copy the test.war into the internal repository as "server/[server-id]/webapps/default/test"
4. Use the archive in "production/webapps/default/test" (or "server/[server-id]/webapps/default/test") as the repository source archive.
5. Delete the old webapps/test directory (saving some directories when expand-preserve-fileset is configured.)
6. Expand the repository source archive files into the webapps/test directory.
7. Restart the webapp

The deployment identifier matches the web-app id that Resin logs at startup time. The identifier a repository tag that lets Resin have a general cloud repository system and also handle web-app deployments, versioning, and staging.

For the example web-app tag "production/webapp/default/test", the "production" is the deployment stage, "webapp" is because it's a webapp deployment, "default" is the virtual-host name and "test" is the web-app name.

web-app deployment options

archive-path	path to the .war file which contains the web-app's contents
dependency-check-interval	how often Resin should check for changes in the web-app for a redeployment
expand-preserve-fileset	a set of files/directories Resin should preserve on a redeploy when it deletes the expanded directory
id	unique identifier for the web-app and the default context-path value
redeploy-check-interval	how often Resin should check the .war file for changes
redeploy-mode	how Resin should handle redeployment: automatic, lazy, or manual
root-directory	path to the expanded web-app directory

Resin checks the following resources to see if the web-app or host should be restarted:

- Any classes in WEB-INF/classes
- Any jars in WEB-INF/lib
- Any additional classes or jars specified in a <class-loader> tag
- Any WEB-INF/resin-web.xml
- Any WEB-INF/web.xml
- resin.conf
- Any other configuration file read by a <resin:import>
- A source .war file configured by <archive-path> or a <web-app-deploy> or <host-deploy>.
- Any file added in a <dependency> tag

Checks can be controlled with the resin.properties line dependency_check_interval. A value of -1 disables the check.

The restart behavior of the web-app is determined by the <redeploy-mode> tag of the web-app. (Defaults to "automatic" can be set to "manual" to be turned off)

Example:
<web-app id="/foo" root-directory="/var/resin/foo" expand-preserve-fileset="WEB-INF/work/**" archive-path="/usr/local/stage/foo.war" redeploy="manual"/>