Taverna 2.3 Server: Installation and Administration Guide

This document relates to the release of Taverna Server that is based on the Taverna 2.3 Platform, from the myGrid team at the University of Manchester.

About

This release is a feature-complete version of the Taverna 2 Server that is provided as a basis for deployments of server-ized Taverna in a multi-user environment.

In addition to its improved performance, this release supports a number of new key features:

- Based on **Taverna 2.3**
- **Multiple users**, with strong separation between them.
- Limited **persistence over service restarts**, depending on exact deployment.
- **Workflow run introspection** capabilities; clients can ask the server what inputs they should supply and what outputs were provided.
- **Workflow run termination notifications** through multiple mechanisms (RSS feed, email, SMS, twitter, etc. depending on deployment).
- **Security**
- **Administrative REST interface** including resource accounting
- Streaming of **large files** both for download and upload.

This is in addition to the features supported by the previous version of Taverna 2 Server:

- **Upload and Execution of arbitrary Taverna 2 workflows**
- **Access to Workflow's Interim Output Files**; no need to wait for the workflow to finish if the results are available sooner
    - **Safe File Management** for handling results; workflows cannot interfere with each others files
- Simple mechanism for **Removal of Expired Workflows**
- Support for both **RESTful and SOAP APIs**, for easier tooling
- **JMX-based Management API**

There remain a number of known-missing features; notably these include access to:

- Provenance Database
- Workflow Execution Model

Installation

Prerequisites

You will need Unix of some kind. (The system security integration does not work with the Windows security model, and the restrictions on command-line lengths on that platform are another significant issue. If you wish to run the server on Windows, see the notes below in §Windows Mode.)

You will need a **Java 6** installation.

You will need a suitable **servlet container**.

> This software was developed using Tomcat 6.0.26 as the servlet container, but other versions of Tomcat are known to work (back to at least 6.0.20) and other containers may also function correctly as no Tomcat-specific APIs are used in the deployable code. We welcome feedback on which containers work, as well as on how to configure them (if they are not Tomcat versions).

Unlike in the previous version of Taverna Server, you do not need a separate installation of the Taverna Workbench or Command-line Tool with this release.

Installation into Tomcat

Note that these instructions are Tomcat-specific.

Step 1. Configure Tomcat for JMX

If you're going to use JMX to administer the server (good for demos; *jvisualvm* is recommended if you've got the JMX support plugin, and *jconsole* is acceptable) then you need to edit Tomcat's *<TOMCATDIR>/bin/startup.sh* script to include the setting:

```
export CATALINA_OPTS=-Dcom.sun.management.jmxremote
```

This works around a minor bug in Spring which prevents correct registration of management beans in the default internal management service. You should also add additional options there to ensure that the JMX management layer is secure; see the Java JMX documentation for a discussion of how to do this.

Step 2. Configure Tomcat for General Management

Add a user entry in *<TOMCATDIR>/conf/tomcat-users.xml* so that the manager webapp can know who you are and that you have permission to deploy webapps (i.e., the "manager" role).

You also *need* to configure Tomcat to support HTTPS if you are planning to use the default secure configuration; to do this, follow the instructions on the Tomcat site (http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html).

Now start Tomcat (or restart it).

Step 3. Prepare for T2Server WebApp Installation

Save the text below as *context.xml* on the machine where you are going to install the server.

```
<Context path="/taverna-server">
</Context>
```

Additional configuration properties can be set in the *context.xml* file; see the detailed deployment description section of this document for more information.

Step 4. Download the Webapp ARchive

Make sure that the *.war* file is also saved to the machine on which you will be installing the server.

Step 5. Install the WebApp

Navigate to *http://<SERVER:PORT>/manager/html* and go to the Deploy box. Fill in the form there with:

Field	Value
Context Path (required):	/taverna-server
XML Configuration File URL:	file:/path/to/context.xml
WAR or Directory URL:	file:/path/to/TavernaServer.war

Press the Deploy button; after a few seconds, Tomcat should respond with OK (at the top of the reloaded page) and you'll have the Taverna Server webapp installed at *http://<SERVER:PORT>/taverna-server*.

Details of Configuration

Deployment of web applications into Tomcat can be done through multiple mechanisms, notably through command line tools and through Tomcat's online administration interface. This document describes the latter mechanism. Note also that we currently only support the use of Taverna Server within a Unix environment (e.g., Linux, Mac OS X); there is no reason in principle why the code should not be adaptable to Microsoft Windows, but there is currently no impersonation module written to integrate Taverna Server with that operating platform.

This assumes that you installing into Tomcat 6 running on top of Java 6; this was tested with Tomcat 6.0.26 over the Sun JRE 1.6.0_14. Later patch versions from the same major version are also recommended (if available).

The configuration of the Taverna Server installation is done by writing a context descriptor document, only some parts of which can be configured afterwards via the management interface. An example of that XML document is below:

```
<Context path="/taverna-server">
    <!-- Sample logging configuration. -->
    <Valve className="org.apache.catalina.valves.AccessLogValve" />

    <Parameter name="default.localusername"
        value="localtavernauser" />

    <!-- For email-dispatched notifications. -->
    <Parameter name="email.host" value="localhost" />
</Context>
```

The context descriptor is typically in a file called *context.xml* and there is a sample context descriptor with this distribution, in the *context.sample.xml* file. There are a substantial number of properties that may be tuned during installation (see below).

The actual deployment is done by giving the actual context location (i.e., the base URL of the webapp relative to the whole Tomcat container) as a separate field, together with URLs (it is useful to use file: URLs for this) to the context descriptor document and the distributed WAR file.

Configuration Property List

This is a list of all the properties that are set by default in the Server (NB: the properites in red are actually unset by default, but they are all understood). They may be overridden by the use of configuration parameters as described above; for example, to override the default local user name, the *default.localusername* configuration parameter would be set.

```
# Script in Taverna installation to run to actually execute workflows
executeWorkflowScript:      /usr/taverna/executeworkflow.sh

# User name to use by default for impersonation if nothing else
# specified
default.localusername:      taverna

# How to pick a user name out of a global identity
localusernameregexp:    ^TAVERNAUSER=(.*)$
# Whether to log incoming workflows; noisy if enabled
default.logworkflows:   false
# Whether to log outgoing exceptions; noisy if enabled
default.logexceptions:  false
# Whether to allow workflows to be submitted; adjustable via admin
# interfaces
default.permitsubmit:   true
# How long a workflow run should live by default, in seconds
default.lifetime:       1440
# Maximum number of simultaneous workflow runs, in any state
default.runlimit:       100

# Location of impersonation credentials
secureForkPasswordFile:     /usr/local/tomcat6.0/conf/sudopass.txt

# URI to server's REST interface
taverna.preferredUserUri:
    https://some.host:8443/taverna-server/rest/

# Delays used in the task executor, both in milliseconds
purge.interval:             30000
finish.interval:            10000

# Thread pool sizing
pool.size:                  2

# Stylesheet to apply to generated XML - EXPERIMENTAL!
xml.stylesheet.url:
        http://www.linkwerk.com/pub/xml/xml2html/xml2html.xslt

### Usage Record handling
usage.logFile:              none
usage.disableDB:            no

### General configuration of messaging
# cooldown in seconds
message.cooldown:           300
message.termination.subject:  Taverna workflow run finished
message.termination.body:   Your job with ID={0} has finished with
                            exit code {1,number,integer}.

### Email-specific options
email.from:                 taverna.server@localhost
email.type:                 text/plain
```

```
email.host:                  localhost

### Jabber-specific options
xmpp.server:                 xmpp://some.host:5222
xmpp.resource:               TavernaServer
xmpp.user:                   taverna
xmpp.password:               *******

### Atom/RSS feed; lifespan in days, cleaninterval in milliseconds
atom.language:               en
atom.lifespan:               7
atom.cleaninterval:          3600000

### SMS-specific options
sms.service:
        https://www.intellisoftware.co.uk/smsgateway/sendmsg.aspx
sms.userfield:               username
sms.passfield:               password
sms.destfield:               to
sms.msgfield:                text
sms.user:                    taverna
sms.pass:                    *******

### Twitter-specific options
twitter.oauth.accessToken:         *******
twitter.oauth.accessTokenSecret:   *******
```

User Accounts

Once you have deployed the server, you can use either JMX or the */admin* interface to create and manage accounts. Only accounts with administrative permission can do such management. The initial set of users is loaded into the database from the *WEB-INF/security/users.properties* file in the deployment package; see the comments in that file for a more complete description of its contents; the file is only used if the user database is empty.

By default, two enabled users are created. One is a normal user (*taverna*, with password *taverna*) and the other is an administrative user (*admin*, password *admin*). These defaults should be changed after installation, as they are not considered secure by default. More information about the mapping process is in the security summary document.

Configuration of Impersonation

If it is desired to separate each user of Taverna Server from the others, it is necessary to configure impersonation of users. That is, the user account that is running the servlet container (Tomcat, etc.) must have permission somehow to execute code as other users. (If this is not desired, the service should be configured to use the simpler non-impersonating worker factory — see the *backEndFactory* property above — or the fallback user identity to use for impersonation should be set in the *default.localusername* to the same identity as the user account used for running the server.)

This is done by either instructing the service what password is to be used with *sudo* (typically the password for the account that is invoking the *sudo* command) or by configuring *sudo* itself so that the service account is more highly authorized than a normal account. The first style of impersonation, which requires that the

service account have a password at all, is enabled by creating a file (in a suitably secured directory) that contains the password as its only content, and telling Taverna Server about it during deployment by giving the full pathname of the file in the *secureForkPasswordFile* deployment parameter. The second style of impersonation is done by leaving that parameter unset and instead adding some extra configuration to the system's */etc/sudoers* file, as seen below (typically set with the *visudo* command). Note that conventionally the three parts of the configuration are in separate sections of the file, and that care should be taken during configuration as mistakes can result in a system that is broken. In the example below, we assume that the servlet container is running as the Unix user *tavserv* and that local user accounts that may be targets for impersonation are all members of the *taverna* UNIX group.

```
# Flags for the tavserv user (keep things quiet)
Defaults:tavserv   !lecture, timestamp_timeout=0, passwd_tries=1

# Who can we impersonate? Manage via Unix group called 'taverna'
Runas_Alias        TAV = %taverna

# The actual permission to impersonate, with permission to run
# anything
tavserv            ALL=(TAV) NOPASSWD: ALL
```

Care should be taken as without a password specified and without permission to execute as another user, an attempt to create a workflow run will hang instead of failing.

Security in Taverna 2 Server

General

Taverna Server normally operates in a mode where it executes each user's workflow runs under a user-id that is specific to that user. This keeps the users from seeing each other's workflow runs by back-door mechanisms, and makes it far easier to apply standard server resource accounting.

In order to do this, it needs to be able to run code (specifically, a Java program) as effectively-arbitrary other users. On Unix (currently the only supported hosting platform) this is implemented through the use of *sudo* with a special configuration, which allows the user hosting the Java container special access. Because of this, it is strongly **recommended** that other web applications be not run in the same container, or that if it is necessary to share webapps that way, the subprocess execution module be instructed where to find a password for use with the *sudo* thunk.

It is **recommended** that Taverna Server always be operated in secure mode, with all connections normally being made via HTTPS. Given that this requires that the container be configured with an SSL certificate, it should be noted that a single-host certificate is available from many certificate authorities for extremely limited cost (even free in some cases). Please consult your container's documentation on how to install the SSL certificate and configure the container for HTTPS operation.

If JMX is used for the management interface (depends on the container) it is **recommended** that it be configured to only accept authenticated connections

over SSL. There is also an */admin* REST interface to the server, which allows access to the same capabilities; it is only accessible to users which have the *ROLE_tavernasuperuser* authority granted to them. Not all parts of the configuration can be managed in this way though; some properties are sufficiently fundamental that they can only be set through the configuration of the deployment descriptor.

Architecture

The communication between the back end workflow executor managers and the front-end webapp is done via RMI, which has been configured to not accept connections from off the local host or class definitions that it does not already know about.

> *You **should** configure your firewall to not permit incoming connections to port 1099 (the default RMI registry port).*

Authorisation of users is done through the use of Spring Security to assign them *on each connection* a set of security authorities. In particular, the key authorities are:

- *ROLE_tavernauser* — allows the user to access the main operational parts of the server.
- *ROLE_tavernasuperuser* — allows the user to read and write all non-security characteristics of all workflows, and also grants access to the */admin* pages.
- *LOCALUSER_** (where the * is replaced with a local user name) — specifies what local user name the user should be executing workflows as; the prefix (*LOCALUSER_*) is simply stripped and the remainder is used as a user name. If absent, the default user name (*taverna* in the default configuration) will be used; two users mapped to the same user name will be able to see each others workflows if they can work out where the job working directories are located, but will not be able to see them inside Taverna Server itself (unless one user grants the other authority to do so, of course).

The default source of authorities is the file *WEB-INF/security/users.properties* (relative to the directory which is the expanded webapp) that is used to populate the database if that is empty.

Insecure Mode

The server can be switched into insecure mode by editing its *WEB-INF/web.xml* file so that it pulls its Spring configuration from *insecure.xml* instead of from *secure.xml* (the default) via the *contextConfigLocation* parameter. This alternate configuration disabled URI rewriting, restricts the set of users to a single one (*taverna* with a password *taverna*) and arranges for execution of workflow runs to be done in the same local userid as is running the host servlet container (Tomcat, etc.)

If you are using this, it is *strongly* recommended that you place the server behind a strong firewall and portal, and only permit vetted workflows to be used.

Windows Mode

For Windows, Taverna Server needs to be configured to be either in fully insecure mode (see above) or to be in a mixed security mode where HTTPS is supported but impersonation is not used. (This is due to a mismatch in the way that impersonation is handled across operating systems.) This has the consequence of ensuring that communication with the server is secure, but it reduces the server's internal security since it will be possible for carefully crafted workflows to gain access to both the workflows of other users and also the internal database used by Taverna Server itself.

Under those restrictions, you can enable this "windows mode" by using *windows.xml* as the value of the *contextConfigLocation* parameter in *WEB-INF/web.xml* after installation. Note that because HTTPS is being used, you will still need to configure the servlet container with an SSL keypair for this to work.

Managing the Server

The server is designed to be managed via JMX. This allows the use of tools such as *jconsole* or *jvisualvm* (with appropriate plugin) to connect to the server so that they can view, chart, and manipulate properties of the server. The exact list of properties is liable to change, but is as follows in this release:

Component: Taverna/Server/Webapp

This is the component that interfaces with the external world.

Property	Type	Description
AllowNewWorkflowRuns	Writable	Whether to permit any new workflow runs to be created; has no effect on existing runs.
CurrentRunCount	Read-Only	Count of currently existing runs.
InvocationCount	Read-Only	Count of SOAP and REST calls made to the Webapp.
LogIncomingWorkflows	Writable	Whether to put submitted workflows in the log.
LogOutgoingExceptions	Writable	Whether outgoing exceptions should be extensively logged.

Component: Taverna/Server/IdAwareForkRunFactory

This component is responsible for manufacturing workflow runs and maintaining connections to existing runs. Note that most of these properties have a sensible value by default.

Property	Type	Description
CurrentRunNames	Read-Only	The names of the currently existing runs.
DefaultLifetime	Writable	How many minutes should a workflow live by default?
ExecuteWorkflowScript	Writable	The full pathname of the script to run to start running a workflow. Must be *readable* by any user of the system.
ExtraArguments	Writable	The list of additional arguments used to make a worker process.
FactoryProcessMapping	Read-Only	The mapping of user names to RMI factory

		IDs.
JavaBinary	Writable	The full pathname of the Java executable to run.
LastExitCode	Read-Only	What was the exit code from the last time the factory subprocess was killed?
LastStartupCheckCount	Read-Only	How many checks were done for the worker process the last time a spawn was tried. (Larger values indicate problems with system loading.)
MaxRuns	Writable	The maximum number of simultaneous runs supported by the server. Note that this includes runs that have finished executing but have not yet been deleted.
PasswordFile	Writable	The full pathname of a file containing a password to use when running a program as another user (e.g., with *sudo*).
RegistryHost	Writable	The host holding the RMI registry to communicate via.
RegistryPort	Writable	The port number of the RMI registry. Should not normally be set.
ServerForkerJar	Writable	The full pathname of the JAR implementing the secure-fork process.
ServerWorkerJar	Writable	The full pathname of the JAR implementing the server worker processes.
SleepTime	Writable	How many milliseconds to wait between checks to see if a worker process has registered.
TotalRuns	Read-Only	How many times has a workflow run been spawned by this engine.
WaitSeconds	Writable	How many seconds to wait for a worker process to register itself before causing the creation operation to fail.

Component: Taverna/Server/Users

This is an interface for adding, deleting and otherwise managing user accounts on the server. It does not manage the underlying system accounts, but does allow control over the mapping of users to those accounts. Note that newly created accounts are disabled by default. More information about the mapping process is in the security summary document.

Property	Type	Description
UserNames	Read-Only	The list of server accounts known about.

Operation	Description
addUser(*nm,pw,cpl*)	Adds the user called *nm* to the database, with password *pw*. If *cpl* is true, set the local user account to be the same as the user name, otherwise use a default set at system configuration time. The user will be a non-admin and disabled by default.
deleteUser(*nm*)	Remove the user called *nm* from the database.

getUserInfo(*nm*)	Get a description of the user called *nm* from the database.
setUserAdmin(*nm*,*ad*)	Set whether the user called *nm* is an admin or not (according to the boolean, *ad*).
setUserEnabled(*nm*,*en*)	Set whether the user called *nm* is an admin or not (according to the boolean, *en*).
setUserLocalUser(*nm*,*lu*)	Set what the user called *nm* will be mapped to as a local user to *lu* (which must be the name of an account understood by the local system).
setUserPassword(*nm*,*pw*)	Set the password for the user *nm* to be *pw*. This implementation stores the value directly in the database.