TomEE Embedded

TomEE Embedded is based on Tomcat embedded and starts a real TomEE in the launching JVM. It is also able to deploy the classpath as a webapp and to use either `META-INF/resources` or a folder as web resources.

Here is a basic programmatic usage based on `org.apache.tomee.embedded.Container` class:

```
try (final Container container = new Container(new Configuration())
.deployClasspathAsWebApp()) {
    System.out.println("Started on http://localhost:" + container.getConfiguration()
.getHttpPort());

    // do something or wait until the end of the application
}
```

All EE features are then accessible directly in the same JVM.

TomEE Embedded Configuration

The default configuration allows to start tomee without issue but you can desire to customize some of them.

Name	Default	Description
httpPort	8080	http port
stopPort	8005	shutdown port
host	localhost	host
dir	-	where to create a file hierarchy for tomee (conf, temp, ...)
serverXml	-	which server.xml to use
keepServerXmlAsThis	false	don't adjust ports/host from the configuration and keep the ones in server.xml
properties	-	container properties
quickSession	true	use Random instead of SecureRandom (for dev)
skipHttp	false	don't use the http connector
httpsPort	8443	https potr
ssl	false	activate https
withEjbRemote	false	use EJBd

Name	Default	Description
keystoreFile	-	https keystore location
keystorePass	-	https keystore password
keystoreType	JKS	https keystore type
clientAuth	-	https client auth
keyAlias	-	https alias
sslProtocol	-	SSL protocol for https connector
webXml	-	default web.xml to use
loginConfig	-	which LoginConfig to use, relies on `org.apache.tomee.embedded.LoginConfigBuilder` to create it
securityConstraints	-	add some security constraints, use `org.apache.tomee.embedded.SecurityConstaintBuilder` to build them
realm	-	which realm to use (useful to switch to `JAASRealm` for instance) without modifying the application
deployOpenEjbApp	false	should internal openejb application be delpoyed
users	-	a map of user/password
roles	-	a map of role/users
tempDir	${java.io.tmpdir}/tomee-embedded_${timestamp}	tomcat needs a docBase, in case you don't provide one one will be created there
webResourceCached	true	should web resources be cached by tomcat (set false in frontend dev)
configuration-location	-	location (classpath or file) to a .properties to configure the server [pre-task
-	Runnable or org.apache.tomee.embedded.LifecycleTask implementations to execute before the container starts	classes-filter

Name	Default	Description
-	implementation of a custom xbean Filter to ignore not desired classes during scanning	basic

Note: passing to `Container` constructor a `Configuration` it will start the container automatically but using `setup(Configuration)` to initialize the configuration you will need to call `start()`.

You can also pass through the properties `connector.xxx` and `connector.attributes.xxx` to customize connector(s) configuration directly.

Standalone applications or TomEE Embedded provided main(String[])

Deploying an application in a server is very nice cause the application is generally small and it allows to update the container without touching the application (typically insanely important in case of security issues for instance).

However sometimes you don't have the choice so TomEE Embedded provides a built-in `main(String[])`. Here are its options:

NOTE	this is still a TomEE so all system properties work (for instance to create a resource).

Name	Default	Description
--path	-	location of application(s) to deploy
--context	-	Context name for applications (same order than paths)
-p or --port	8080	http port
-s or --shutdown	8005	shutdown port
-d or --directory	./.apache-tomee	tomee work directory
-c or --as-war	-	deploy classpath as a war
-b or --doc-base	-	where web resources are for classpath deployment
--renaming	-	for fat war only, is renaming of the context supported
--serverxml	-	the server.xml location
--tomeexml	-	the server.xml location
--property	-	a list of container properties (values follow the format x=y)

Note that since 7.0.0 TomEE provides 3 flavors (qualifier) of tomee-embedded as fat jars:

- uber (where we put all request features by users, this is likely the most complete and the biggest)
- jaxrs: webprofile minus JSF
- jaxws: webprofile plus JAX-WS

These different uber jars are interesting in mainly 2 cases:

- you do a war shade (it avoids to list a bunch of dependencies but still get a customized version)
- you run your application using `--path` option

NOTE	if you already do a custom shade/fatjar this is not really impacting since you can depend on `tomee-embedded` and exclude/include what you want.

FatApp a shortcut main

`FatApp` main (same package as tomee embedded `Main`) just wraps the default main ensuring:

- `--as-war` is used
- `--single-classloader` is used
- `--configuration-location=tomee-embedded.properties` is set if `tomee-embedded.properties` is found in the classpath

configuration-location

`--configuration-location` option allows to simplify the configuration of tomee embedded through properties.

Here are the recognized entries (they match the configuration, see org.apache.tomee.embedded.Configuration for the detail):

Name	
http	
https	
stop	
host	
dir	
serverXml	
keepServerXmlAsThis	
quickSession	

skipHttp	
ssl	
http2	
webResourceCached	
withEjbRemote	
deployOpenEjbApp	
keystoreFile	
keystorePass	
keystoreType	
clientAuth	
keyAlias	
sslProtocol	
webXml	
tempDir	
classesFilter	
conf	
properties.x (set container properties x with the associated value)	
users.x (for default in memory realm add the user x with its password - the value)	
roles.x (for default in memory realm add the role x with its comma separated users - the value)	
connector.x (set the property x on the connector)	
realm=fullyqualifiedname,realm.prop=xxx (define a custom realm with its configuration)	
login=,login.prop=xxx (define a org.apache.tomee.embedded.LoginConfigBuilder == define a LoginConfig)	
securityConstraint=,securityConstraint.prop=xxx (define a org.apache.tomee.embedded.SecurityConstaintBuilder == define webapp security)	

configurationCustomizer.alias=,configurationCustomizer.alias.class=class,configurationCustomizer.alias.prop=xxx (define a ConfigurationCustomizer)	

Here is a sample to add BASIC security on `/api/*`:

```
# security configuration
securityConstraint =
securityConstraint.authConstraint = true
securityConstraint.authRole = **
securityConstraint.collection = api:/api/*

login =
login.realmName = app
login.authMethod = BASIC

realm = org.apache.catalina.realm.JAASRealm
realm.appName = app

properties.java.security.auth.login.config = configuration/login.jaas
```

And here a configuration to exclude jackson packages from scanning and use log4j2 as main logger (needs it as dependency):

```
properties.openejb.log.factory = log4j2
properties.openejb.container.additional.include =
com.fasterxml.jackson,org.apache.logging.log4j
```

Application Runner

SInce TomEE 7.0.2, TomEE provide a light ApplicationComposer integration for TomEE Embedded (all features are not yet supported but the main ones are): `org.apache.tomee.embedded.TomEEEmbeddedApplicationRunner`. It relies on the definition of an `@Application`:

```
@Application
@Classes(context = "app")
@ContainerProperties(@ContainerProperties.Property(name = "t", value = "set"))
@TomEEEmbeddedApplicationRunner.LifecycleTasks(MyTask.class) // can start a
ftp/sftp/elasticsearch/mongo/... server before tomee
@TomEEEmbeddedApplicationRunner.Configurers(SetMyProperty.class)
public class TheApp {
    @RandomPort("http")
    private int port;

    @RandomPort("http")
    private URL base;

    @org.apache.openejb.testing.Configuration
    public Properties add() {
        return new PropertiesBuilder().p("programmatic", "property").build();
    }

    @PostConstruct
    public void appStarted() {
        // ...
    }
}
```

Then just start it with:

```
TomEEEmbeddedApplicationRunner.run(TheApp.class, "some arg1", "other arg");
```

TIP `@Classes(values)` and `@Jars` are supported too which can avoid a huge scanning if you run with a lot of not CDI dependencies which would boost the startup of your application.