Tests

Cette page n'est plus actualisée. À partir de BlueMind 4.8, veuillez consulter la **nouvelle documentation BlueMind**

Présentation

Cette page présente l'exécution et le fonctionnement des tests via mavin, eclipse, docker ou une machine virtuelle.

Sur cette page :

En rapport :

Configuration

Le fichier services.json

Le fichier `services.json` du projet de tests énumère les container/services (docker) nécessaire à l'exécution des tests.

Maven lancera les containers présent dans le fichier avant de lancer les tests et les arrêtera après.

Par exemple pour deux images docker, une pour la bd postgres et une pour un elasticsearch, le fichier contient :

```
[{
    "name":"bluemind/postgres-tests"
    },{
    "name":"bluemind/elasticsearch-tests"
}]
```

Le fragment net.bluemind.pool.dockerconfig

Ce fragment permet de surcharger (remplacer) la classe `BmConfIni.`

Si le fragment est actif (non fermé) et que le projet de tests comporte un fichier `services.json`, le fragment contactera l'instance docker pour trouver les adresses IP des containers correspondant aux images du fichier `services.json` et remplacera/surchargera les divers paramètres que fournit `BmConfIni` (l'hôte de la base de données par exemple).

Quand on fait tourner les tests dans Eclipse et qu'on veut utiliser les services fournis par une machine virtuelle BlueMind (postgres, elasticsearch etc..), il suffit de fermer le fragment. Les fichiers de configuration traditionnels prendront alors le relai (`/etc/bm/bm.ini`, `/etc/bm/mcast.id`, etc.)

Configuration maven

La configuration maven des projets de tests répertorie les differents plugins nécessaires au lancement des tests :

```
<plugin>
    <groupId>org.eclipse.tycho</groupId>
    <artifactId>tycho-surefire-plugin</artifactId>
    <configuration>
        <showEclipseLog>true</showEclipseLog>
        <useUIHarness>false</useUIHarness>
        <useUIThread>false</useUIThread>
        <product>org.eclipse.platform.ide</product>
    </configuration>
</plugin>
<plugin>
    <groupId>org.eclipse.tycho</groupId>
    <artifactId>target-platform-configuration</artifactId>
    <configuration>
        <dependency-resolution>
            <extraRequirements>
                <requirement>
                    <type>eclipse-feature</type>
                    <id>net.bluemind.tests.feature</id>
                    <versionRange>0.0.0</versionRange>
                </requirement>
            </extraRequirements>
        </dependency-resolution>
    </configuration>
</plugin>
```

Jouer les tests depuis Docker

Installer docker (préférer les paquets téléchargeables sur docker.com plutôt que les paquets fournis avec les distributions).

Mettre docker en écoute sur un socket, par exemple sur Debian/Ubuntu :

1. éditer `/etc/default/docker`
2. définir la variable suivante :
   `DOCKER_OPTS="-H tcp://<ip hote>:10000 -H unix:///var/run/docker.sock"`
3. relancer le service Docker

Sur la machine de développement, celle où se trouve Eclipse et les sources BlueMind, il faut ensuite indiquer quel service Docker utiliser en créant le fichier `~/.docker.io.properties` avec le contenu suivant :

```
docker.io.url=http://<ip hote docker>:10000
```

Par la suite, ce sont les containers Docker qui devront-être accessibles depuis Eclipse. Il sera peut-être nécessaire d'ajouter une route sur la machine Eclipse pour accéder au réseau des containers.

D'autres méthodes possibles sont consultable sur le git docker-java.

Les images Docker

Il est actuellement nécessaire d'installer les images en local (sur le docker qui est utilisé pour les tests). Les images se trouvent dans le projet `bm/bm-docker-devenv` :

```
$ git clone https://forge.blue-mind.net/stash/scm/bm/bm-docker-devenv.git
$ ./build
```


Pensez à reconstruire vos images régulièrement afin d'utiliser des versions récentes des binaires BlueMind.

Connecter des répertoires locaux en tant que volumes Docker

Cela se fait via un tableau (*Array*) "volumes".

Par exemple pour connecter le répertoire local "`/tmp/backups/bluemind`" avec le répertoire "`/var/backups/bluemind`" dans le container docker, on utilisera le code suivant :

```
[{
    "name":"bluemind/postgres-tests",
    "volumes":[ { "localPath":"/tmp/backups/bluemind", "containerPath":"/var/backups/bluemind" } ]
}]
```

Dans le DockerFile d'images docker, il faut en plus ajouter le VOLUME :

```
VOLUME ["/var/backups/bluemind"]
```

Jouer les tests depuis Eclipse

Construire une première fois le projet BlueMind :

```
$ cd <sourcesBM>/open
$ mvn -Dmaven.test.skip=true clean install
```

Démarrer les containers Docker nécessaires :

```
$ cd <repertoire plugin test>
$ mvn pre-integration-test
```

Enfin, lancer les tests dans Eclipse.

Attention: Pour windows et MacOS, il est nécessaire d'ajouter une route sur l'hôte de la VM (windows ou MacOS) afin d'atteindre le réseau des containers Docker.
Il faut aussi activer le routage sur la VM Docker :

```
$ sysctl -w net.ipv4.ip_forward=1
$ echo 'net.ipv4.ip_forward=1' > /etc/sysctl.conf
```

Tester sans container OSGI (Junit)

Les classes nommées `*Test.java` (différentes de `*Test`**`s`**`.java`) sont exécutées sans container OSGI.

Cela revient à lancer des tests dans Eclipse avec «*Run as Junit Test*»(au contraire de «*Run as Junit* ***Plugin*** *Test*») et résout des problèmes de Classpath avec des Frameworks de Mocking proxy based, comme Mockito. Les tests s'exécutent aussi plus vite.