JDG + EAP Lab 2 Guide

This explains the steps for lab 2, either follow them step-by-step or if you feel adventurous try to accomplish goals without the help of the step-by-step guide.

Background

In Lab 1 we implemented a side cache using JDG to speed up reads, but master data store is still the database. So far however the data access is only a using the common CRUD (Create, Read, Update and Delete) operations. Since JDG primary are a key/value store these operations are easy to implement.

A competing vendor that has a similar task management solution released a new feature where users can filter their tasks. Something our customers has been requesting for a while. Our marketing director demands that we ASAP add this feature. An external consultant are hired and to implement this feature, but since he wasn't familiar with JDG he implemented the filter solution using JPA query. This is however not responsive enough and we refactor the filter function to query JDG instead.

JDG has very advanced querying capabilities in library mode (client/server quering is tech preview for JDG 6.3)

Use-case

You are tasked to rewrite the filter implementation using queries in JDG instead of JPA queries. However the Task data model is used in the native mobile application and since it will take a while before we can update the mobile application you are not allowed to change the org.jboss.infinspan.demo.model.Task class.

Objectives

Your task in Lab 2 re-implement the filtering method, but using JDG Queries. The UI and REST methods are already implemented.

Basically you should replace the DB Query with a JDG Query and you will have to do this without modifying the org.jboss.infinspan.demo.model.Task class.

To to this we need to do the following:

1. Setup the lab environment
2. Add developer dependencies: Update the pom.xml and add developer dependency to infinispan-query
3. Add runtime dependencies Update jboss-deployment-structure.xml to add runtime dependency to infinispan-query
4. Update configuration Enable indexing in the API Configuration. Hint See the Infinispan Query Index
    - The index should only be persisted in RAM
    - Since we will later deploy this on mulitple EAP instances we need to allow for shared indexes.
    - The index should be based on the `title` field from `org.jboss.infinspan.demo.model.Task`
5. Write the implementation to Query JDG Replace the implementation of `TaskSerivce.filter(String)` to query JDG instead of DB

Step-by-Step

Setup the lab environment

To assist with setting up the lab environment we have provided a shell script that does this.

Note: *If you previously setup up lab 1 using this script there is no need to do this for lab 2_*

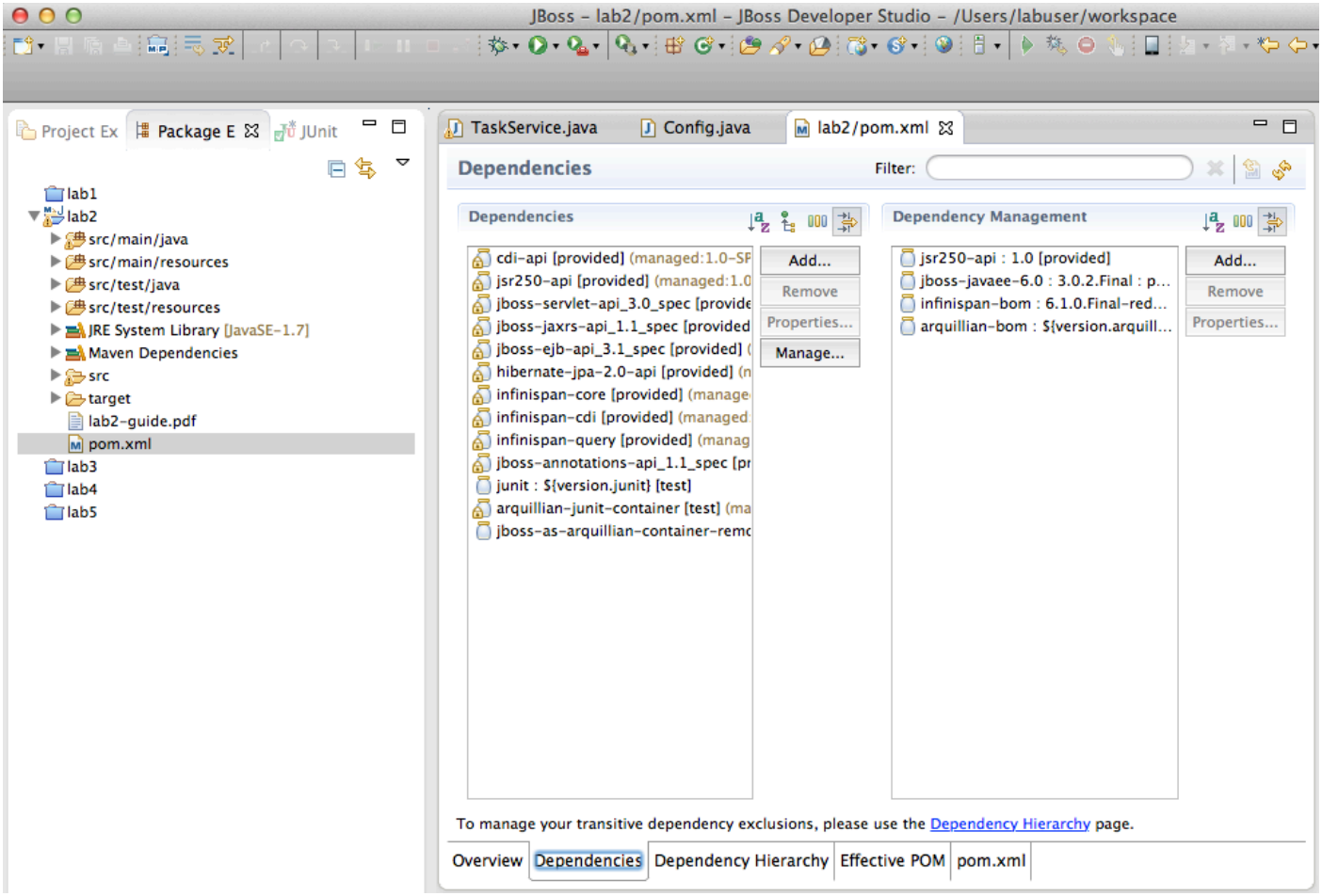
1. Run the shell script by standing in the jdg lab root directory (~/jdg-labs) execute a command like this

```
$ sh init-lab.sh --lab=2
```

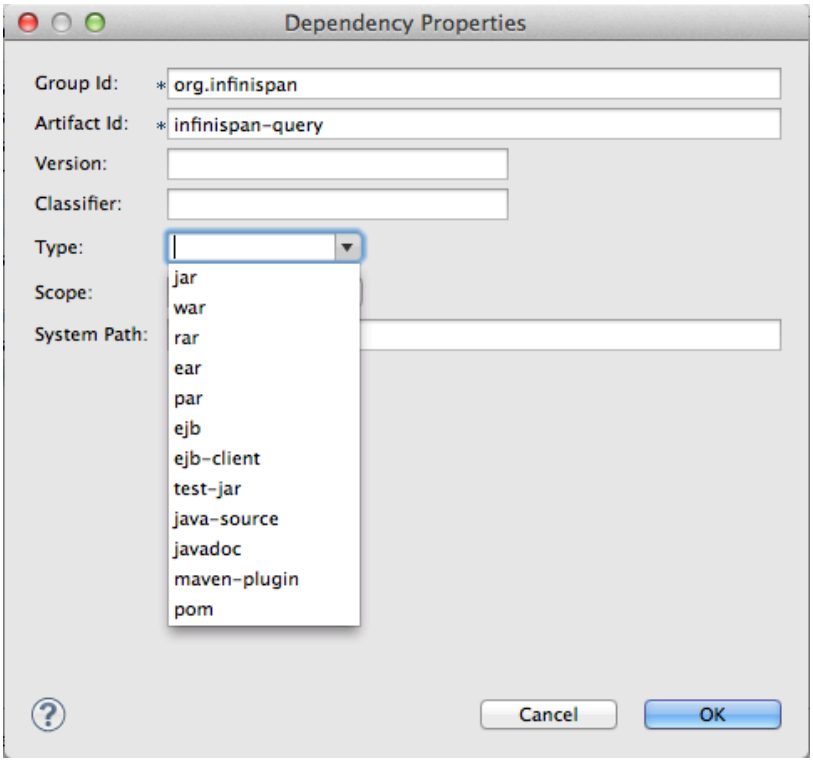
Add developer dependencies

1. Open the lab2 pom.xml (see below)

2. Select the dependencies tab



3. With `infinispan-query [provided]` selected, click the **Properties...** button
4. The Dependency Properties window will appear.



5. Verify the Type value is `jar` and the Scope is `provided`. Click OK once you are done.

Add runtime dependencies

1. Open `src/main/webapp/WEB-INF/jboss-deployment-structure.xml`

2. Add `org.infinispan.query` module. The content of the file should look like this:

```
<jboss-deployment-structure>
    <deployment>
        <dependencies>
            <module name="org.infinispan" slot="jdg-6.3" services="import"/>
            <module name="org.infinispan.cdi" slot="jdg-6.3" meta-inf="import"/>
            <module name="org.infinispan.query" slot="jdg-6.3" services="import"/>
        </dependencies>
    </deployment>
</jboss-deployment-structure>
```

3. After saving It's recommended to run the JUnit test to verify that everything deploys fine.

Update the configuration

1. Open `src/main/java/org/jboss/infinispan/demo/Config.java`

2. After the global configuration we need to create a `SearchMapping` object that tells JDG how to index `Task` objects

```
SearchMapping mapping = new SearchMapping();
mapping.entity(Task.class).indexed().providedId()
    .property("title", ElementType.METHOD).field();
```

3. Create a `Properties` object and store the `SearchMapping` object under the `org.hibernate.search.Environment.MODEL_MAPPING` key.

```
Properties properties = new Properties();
    properties.put(org.hibernate.search.Environment.MODEL_MAPPING, mapping);
```

4. We also need to tell JDG (or Lucene) to store the indexes in ram memory by adding a property with key "default.directory_provider" and value "key".

```
properties.put("default.directory_provider", "ram");
```

5. Now we can enable the index on the configuration object by adding `.indexing().enable()` to the fluid API before `.build()`.

6. Also we want to configure the index to support clustering adding `.indexLocalOnly(false)` to the fluid API before `.build()`.

7. And finally we want to pass in the properties configuration by adding `.withProperties(properties)` to the fluid API before `.build()`. The config class should now look like this:

```
package org.jboss.infinispan.demo;

import java.lang.annotation.ElementType;
import java.util.Properties;

import javax.annotation.PreDestroy;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Default;
import javax.enterprise.inject.Produces;

import org.hibernate.search.cfg.SearchMapping;
import org.infinispan.configuration.cache.Configuration;
import org.infinispan.configuration.cache.ConfigurationBuilder;
import org.infinispan.configuration.global.GlobalConfiguration;
import org.infinispan.configuration.global.GlobalConfigurationBuilder;
import org.infinispan.eviction.EvictionStrategy;
import org.infinispan.manager.DefaultCacheManager;
import org.infinispan.manager.EmbeddedCacheManager;
import org.jboss.infinispan.demo.model.Task;

/**
 * This is Class will be used to configure JDG Cache
 * @author tqvarnst
 *
 * DONE: Add configuration to enable indexing of title field of the Task class
 *
 */
public class Config {

    private EmbeddedCacheManager manager;

    @Produces
    @ApplicationScoped
    @Default
    public EmbeddedCacheManager defaultEmbeddedCacheConfiguration() {
        if (manager == null) {
            GlobalConfiguration glob = new GlobalConfigurationBuilder()
```

```
                    .globalJmxStatistics().allowDuplicateDomains(true).enable() // This
                    // method enables the jmx statistics of the global
                    // configuration and allows for duplicate JMX domains
                    .build();

            SearchMapping mapping = new SearchMapping();
            mapping.entity(Task.class).indexed().providedId()
                  .property("title", ElementType.METHOD).field();

            Properties properties = new Properties();
            properties.put(org.hibernate.search.Environment.MODEL_MAPPING, mapping);
            properties.put("default.directory_provider", "ram");


            Configuration loc = new ConfigurationBuilder().jmxStatistics()
                    .enable() // Enable JMX statistics
                    .eviction().strategy(EvictionStrategy.NONE) // Do not evic objects
                    .indexing()
                        .enable()
                        .indexLocalOnly(false)
                        .withProperties(properties)
                    .build();
            manager = new DefaultCacheManager(glob, loc, true);
        }
        return manager;
    }

    @PreDestroy
    public void cleanUp() {
        manager.stop();
        manager = null;
    }
}
```

Write the implementation to Query JDG

1. Open `src/main/java/org/jboss/infinispan/demo/TaskService.java`
2. Navigate to the `public Collection<Task> filter(String input)` method and delete the code implementation

3. In order create QueryBuilder and run that query we need a `SearchManager` object. We can get that by calling `Search.getSearchManager(cache)`

   ```
   SearchManager sm = Search.getSearchManager(cache);
   ```

4. To create a `QueryBuilder` object we can then get a `SearchFactory` from the `SearchManager` and call `buildQueryBuilder().forEntity(Task.class).get()` on it.

   ```
   QueryBuilder qb = sm.getSearchFactory().buildQueryBuilder().forEntity(Task.class).get();
   ```

5. Now we can create a `Query` object from the `QueryBuilder` using the fluid api to specify which Field to match etc.

   ```
   Query q = qb.keyword().onField("title").matching(input).createQuery();
   ```

6. We can now get a `CacheQuery` object by using the `SearchManager.getQuery(...)` method.

   ```
   CacheQuery cq = sm.getQuery(q, Task.class);
   ```

7. The `CacheQuery` extends `Iterable<Object>` directly, but since we are expecting a `Collection<Task>` to return we will have to call `CacheQuery.list()` to get a `List<Object>` back. This will now have to be cast to typed Collection using double Casting.

   ```
   return (Collection<Task>)(List)cq.list();
   ```

   Note that since we are using a QueryBuilder specifically for Task.class we can safely do this cast.

8. You also need to add the following import statements

   ```
   import java.util.List;
   import org.apache.lucene.search.Query;
   import org.hibernate.search.query.dsl.QueryBuilder;
   import org.infinispan.Cache;
   import org.infinispan.query.CacheQuery;
   import org.infinispan.query.Search;
   import org.infinispan.query.SearchManager;
   ```

9. `TaskService.java` should now look like this

   ```
   package org.jboss.infinispan.demo;
   ```

```
import java.util.Collection;
import java.util.Date;
import java.util.logging.Logger;

import javax.annotation.PostConstruct;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.inject.Inject;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

import java.util.List;
import org.apache.lucene.search.Query;
import org.hibernate.search.query.dsl.QueryBuilder;
import org.infinispan.Cache;
import org.infinispan.query.CacheQuery;
import org.infinispan.query.Search;
import org.infinispan.query.SearchManager;
import org.jboss.infinispan.demo.model.Task;

/**
 * This class is used to query, insert or update Task object.
 * @author tqvarnst
 *
 */
@Stateless
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public class TaskService {

        @PersistenceContext
    EntityManager em;

        @Inject
        Cache cache;

        Logger log = Logger.getLogger(this.getClass().getName());

        /**
         * This methods should return all cache entries, currently contains mockup code.
         * @return
         */
        public Collection findAll() {
                return cache.values();
        }

        /**
         * This method filters task based on the input
         * @param input - string to filter on
         * @return
         *
         * FIXME: The current implementation is database query, replace it with a JDG query instead
         */
        public Collection filter(String input) {
                SearchManager sm = Search.getSearchManager(cache);
                QueryBuilder qb = sm.getSearchFactory().buildQueryBuilder().forEntity(Task.class).get();
                Query q = qb.keyword().onField("title").matching(input).createQuery();
                CacheQuery cq = sm.getQuery(q, Task.class);
                return (Collection)(List)cq.list(); //Since we only Query Task.class we can safely cast this
        }

        /**
         * This method persists a new Task instance
         * @param task
         *
         */
        public void insert(Task task) {
                if(task.getCreatedOn()==null) {
                        task.setCreatedOn(new Date());
                }
                em.persist(task);
                cache.put(task.getId(),task);
        }


        /**
         * This method persists an existing Task instance
         * @param task
         *
         */
```

```
        public void update(Task task) {
                Task newTask = em.merge(task);
                em.detach(newTask);
                cache.replace(task.getId(),newTask);
        }

        /**
         * This method deletes an Task from the persistence store
         * @param task
         *
         */
        public void delete(Task task) {
                //Note object may be detached so we need to tell it to remove based on reference
                em.remove(em.getReference(task.getClass(),task.getId()));
                cache.remove(task.getId());
        }


        /**
         * This method is called after construction of this SLSB.
         *
         */
        @PostConstruct
        public void startup() {

                log.info("### Querying the database for tasks!!!!");
                final CriteriaBuilder criteriaBuilder = em.getCriteriaBuilder();
                final CriteriaQuery criteriaQuery = criteriaBuilder.createQuery(Task.class);

                Root root = criteriaQuery.from(Task.class);
                criteriaQuery.select(root);
                Collection resultList = em.createQuery(criteriaQuery).getResultList();

                for (Task task : resultList) {
                        this.insert(task);
                }

        }

}
```

Test and deploy

Now you are almost finished with Lab 2, you should now run the Arquillian tests and then deploy the application.

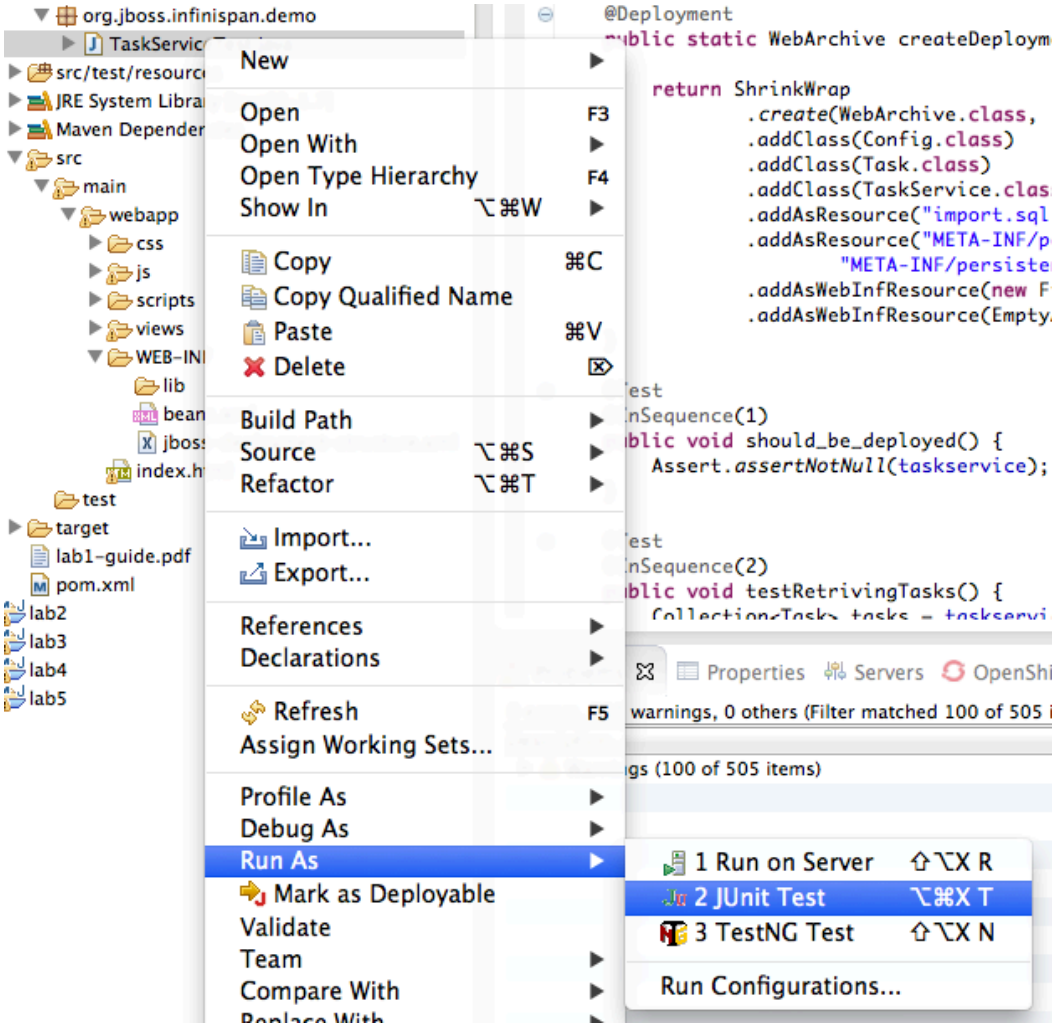
- In another terminal (on the developer PC) change directory to the project lab2

```
$ cd projects/lab2
```

Run the JUnit test either in JBDS (see an example in the next step) or by using command line (below). To run the test the `arquillian-jbossas-remote-7` profile will have to be activated.

```
$ mvn -P arquillian-jbossas-remote-7 test
```

- Run the JUnit test by right clicking TaskServiceTest.java and select Run As ... -> JUnit Test



- Build and deploy the project

```
$ mvn package jboss-as:deploy
```

- Verify in a browser that application deployed nice successfully by opening http://localhost:8080/mytodo in a browser.
- Click around and verify that you can now filter tasks, in addition to other functions