R & D Workshop
Tomcat in the Cloud
Final Report

Ismaïl Senhaji
ismail.senhaji@unifr.ch

Guillaume Pythoud
guillaume.pythoud@unifr.ch

Supervised by
Jean-Frederic Clere
jclere@redhat.com

June 20, 2017

Contents

1 Project description **3**
1.1 Project and context 3
1.2 Goals and objectives of the project 3

2 Methodology **4**
2.1 State of the art 4

3 Result and realization **5**
3.1 Process 5
3.2 Testing 5
3.3 Result 5

4 Recommendations **6**
4.1 Statements of recommendations 6
4.2 Limitations 6
4.3 Outstanding issues and perspective for future work 6

1 Project description

1.1 Project and context

For this project, we have been assigned to *Red Hat*[1]. Red Hat is an international software company, specializing in open source. Among others, they develop the *OpenShift Container Platform*[2], which is full-stack cloud management solution.

We will work on the *Tomcat*[3] Java web server. Our task is to extend Tomcat in order to make session replication work in a cloud environment.

As it stands, session replication in Tomcat is not cloud ready yet. The main issue is that it uses multicast which only works in a local network. In a cloud deployment, instances can be distributed over the internet. This calls for the development of a new session replication solution.

1.2 Goals and objectives of the project

The major goal of the project to make Tomcat's session work in the cloud. This goal is divided in 3 steps:

1. Study existing methods
2. Implement a new solution based on one of those methods
3. Test the implementation

To test our implementation, we will develop a Java web application. It will simply display these informations:

- The ID of the Tomcat instance that served the request
- The user's session ID
- A counter that is incremented on every request.

A secondary goal is to build and configure a Raspberry Pi cluster, running OpenShift. If successful, it will be presented as a demonstration at the end of the project.

Another important goal is to provide detailed documentation for future users. We will write a "Quick Start Guide" detailing the steps to follow to have a working Tomcat cluster running in an OpenShift cloud.

[1] `http://www.redhat.com/en/about`
[2] `https://www.openshift.com/container-platform/index.html`
[3] `https://www.openshift.com/container-platform/index.html`

2 Methodology

Most development will be done in Java, since Tomcat itself is written in Java. But we suspect that most time will be spent on reading documentation and code than on writing code.

As mentioned above, all our work is available on our GitHub repository, be it code, documentation or simply comments and ideas.

For testing our solutions, we will use MiniShift[4]. MiniShift is a tool that deploys a "local" (i.e consisting of a unique machine) OpenShift cluster on any computer. This will allow us to run tests on our laptops, saving the time necessary to deploy a real OpenShift installation. Only towards the end of the project, when everything else is done, will we finally deploy our solution on a real cluster.

2.1 State of the art

Our research has shown that Tomcat has two built-in session replication solutions that could be built on[5]:

1. *DeltaManager*: this session manager replicates session data to all other instances in a cluster. Peer discovery can be either be done statically or dynamically. In static mode, a list of peers is hardcoded in Tomcat's configuration file. In dynamic mode, peers are discovered using multicast.

2. *PeristanceManager*: session data is stored in the filesystem or in a JDBC-capable database.

Paths to explore are modifying DeltaManager to discover peer using a different method (since multicast over the internet isn't feasible), or configure PersistanceManager to use a distributed datastore. The first solution might be more interesting from a performance point of view.

Finally, if none of the solutions above turn out fruitful, there still are other paths to explore: there are solutions that bypass Tomcat's session manager and use an external one instead. The *Spring Framework*[6] offers such a solution, and is advertised to have built-in support for distributed datastores such as *Redis* or *Hazelcast*[7]. Another solution ports WildFly's session manager to Tomcat[8].

While these solutions using external session managers are known to work, we will mainly concentrate on those based on Tomcat, as they avoid relying

[4]`https://www.openshift.org/minishift/`

[5]`https://tomcat.apache.org/tomcat-8.5-doc/cluster-howto.html`

[6]`http://projects.spring.io/spring-framework/`

[7]`http://docs.spring.io/spring-session/docs/current/reference/html5/`

[8]`https://github.com/wildfly-clustering/wildfly-clustering-tomcat`

on too many external dependencies. But we consider them as a “Plan B” if all else fails.

3 Result and realization

3.1 Process

As our knowledge of Tomcat was low we started by implementing a basic web application that used the built-in Tomcat solution for session replication within a cluster. We continued with this idea by developing new iteration with always new solutions for the session replication. Hence we were able to get at each iteration a better idea on how Tomcat was working and we were able to provide every time a new solution which was closest to the goal. During this process we provided different testing application that included different technology. At the beginning of the project we wanted to use Spring Boot but as it add a lot of complexity in the tomcat configuration we finally decided to not include it. All those testing applications are stored in the *experiment*[9] branch in the github repository. We also discovered technologies that we eventually used in the final solution, for example *fabric8 maven plugin*[10] that helped us build an deploy our application on Openshift.

3.2 Testing

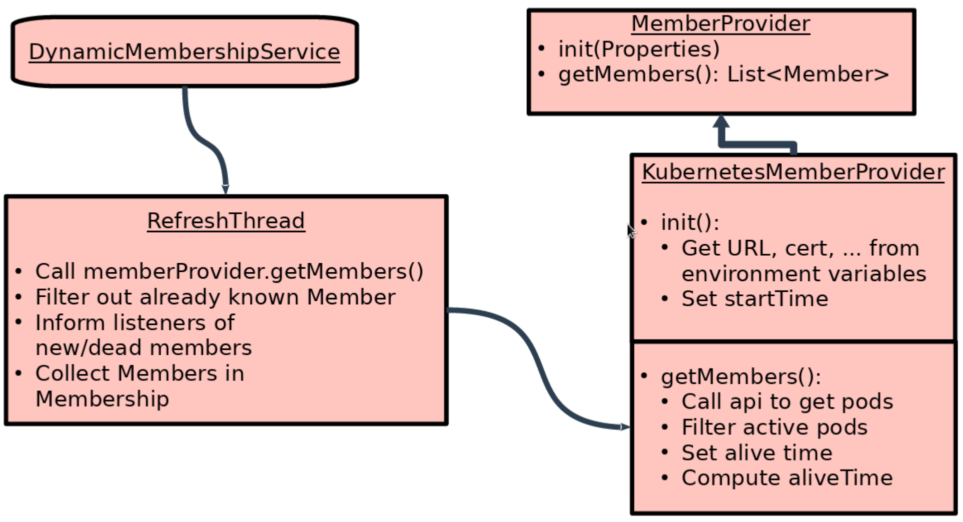
To test our applications we did not implement any testing suite. As the application was simple it was quick to manually check if the requests were correctly handled. Basically we checked if the data of a selected session was correctly stored and shared on all the pods of the cluster, even after some scale up and scale down.

3.3 Result

The final solution we provided is inspired from *Kube PING*[11], a Kubernetes discovery protocol for JGroup. We also analyzed the implementation of the multicast membership service of Tomcat and made our own membership service that used the Openshift API [12] to get the running pods of the project. We also compute the alive time of each pods with the help of the *creationTime* of each pods and finally add it to the membership. The implementation is described in the figure1. A small guide on how to use this application is specified in a *Readme* file in the github repository.

[9]`https://github.com/iSma/tomcat-in-the-cloud/tree/experiments`
[10]`https://github.com/fabric8io/fabric8-maven-plugin`
[11]`https://github.com/jgroups-extras/jgroups-kubernetes`
[12]`https://docs.openshift.com/enterprise/3.0/rest_api/index.html`

Figure 1: DynamicMembershipService


DynamicMembershipService
MemberProvider
• init(Properties)
• getMembers(): List<Member>
RefreshThread
• Call memberProvider.getMembers()
• Filter out already known Member
• Inform listeners of new/dead members
• Collect Members in Membership
KubernetesMemberProvider
• init():
• Get URL, cert, ... from environment variables
• Set startTime
• getMembers():
• Call api to get pods
• Filter active pods
• Set alive time
• Compute aliveTime

Implementation of the DynamicMembershipService

4 Recommendations

4.1 Statements of recommendations

The solution we finally provide at the end of the project is meant to be seen as a proof of concept to help others that looks for a way of enabling session replication in an Openshift environment within a tomcat application. Hence we tried to provide a solution with the lowest possible number of dependencies and that could be easily replicated by others.

4.2 Limitations

As we focused on a having a working solution, our error handling is currently at bare minimum, this is something that could be improved. Even if the Class that call the API is called *KubernetesMemberProvider* we don't know if the implementation is working on a pure Kubernetes environment as the call could differ than the one of the Openshift API. At the moment the session data of all the pods of a namespaces/project are shared even if they are not running the same application, this could become a problem with a large scale project with multiple applications running.

4.3 Outstanding issues and perspective for future work

Our tests was quite simple, so developing a testing tools for session data replication checking within a cluster could be a great future step. It would provide us a best way to test our implementation and maybe find new issues that our previous testing phases did not show up.