CS 549—Fall 2019
Distributed Systems and Cloud Computing
Assignment Three—Server Side Events

In the previous assignment, you developed a simple peer-to-peer distributed hash table for sharing bindings of keys to values. The protocol for retrieving bindings from the database is pull-based: a client (node) searches the network for the bindings for a particular key. In this assignment, you will extend this with a push-based protocol: Clients will be notified when new bindings are added for a key. The implementation of this push protocol will be based on the server-side events (SSE) API in the Jersey framework. See here for more information:
`https://jersey.java.net/documentation/latest/sse.html`

The command line interface (CLI) for a node should support three commands:
1. `listenOn <key>` records an interest in being notified of changes in bindings for the specified key. For this assignment, you can focus just on notifications of the addition of new bindings.
2. `listenOff <key>` removes an interest in being notified of changes in bindings for the specified key.
3. `listeners` lists all of the keys for which this node has registered listeners in the network.

When a listener is registered, the node searches the network for the node where bindings for the corresponding key are stored. A Web service call is made to that node, to open up a communication stream for that node to send notifications to the client node when a binding for that key is stored. At the client node, a listener callback is registered to process event notifications from the server node.

It is possible to remove a notification stream by closing the event notification stream at the client. However before doing this, you must get the server to shut down the event notification stream, otherwise the client-side closing of the stream will just time out. The API therefore includes a Web service call from the client to the server to shut down event notifications (for a particular key) to this client.

Once you have your code running locally, you should test it on some EC2 instances that you define. Copy the server jar file to the EC2 instance using "`scp`," using the "`-i`" option to define the private key file that you use to authenticate yourself to the instances. Then ssh to the instance and run the server script. It should be in a directory of the form

```
/home/ec2-user/tmp/cs549/dht-test
```

that you should have created, again with a subfolder called "`node`" for backing up server state. Now you should be able to run network nodes on several EC2 instances and have them communicate with each other.

The REST API for the original DHT is extended with a new resource, with two operations:

Verb	URI	Specification
GET	/dht/listen?id=ID&key=KEY	Listen for notifications of new bindings.
DELETE	/dht/listen?id=ID&key=KEY	Stop event notifications for a node and key.

The first operation should return an event output stream that notifies the client of changes in the bindings for the specified key. The second operation cancels event notifications for changes to a key KEY (a string value) sent to a client identified by node ID (an integer value).

In addition, you should make your solution robust to reconfigurations of the network. Specifically, when a new node joins the network, some of the existing bindings may be transferred to it. This is a problem for any listeners for changes to those transferred bindings. Fortunately the solution is relatively simple. When the bindings for a key are transferred to a new node, the node where those bindings used to be stored should notify any listener clients (e.g. with a special type of event) and shut down server-side processing. On the client side, the connection to the old server should be broken, and the client should then repeat the logic for finding the node where bindings for a key are stored, and establishing an event channel from that server.

Once you have your code working, please follow these instructions for submitting your assignment:

- Export your Eclipse project to the file system.
- Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory `Humphrey_Bogart`.
- In that directory you should provide the zip archive that contains your sources, and the server jar file, `dht.jar`.
- **Provide short videos demonstrating the successful testing of your instance**, with the scenarios described above (adding listeners for remote bindings, adding bindings at a third site, adding a node that causes a listened-to binding to be moved).
- Also include in the directory a filled-in rubric for your submission (see the submission Web page for the rubric).

It is very important for your grade that you do adequate testing. You should at a minimum demonstrate testing of these scenarios:

1. Joining at least three nodes together into a DHT network.
2. Adding bindings at both local and remote nodes. You can do all your testing using EC2, but use different instances behind their own firewalls. Use the `bindings` and `listeners` commands of the CLI to show the functioning of your network. You should demonstrate adding listeners for remote bindings, adding bindings at a third site, adding a node that causes a listened-to binding to be moved, and show that new bindings are still being reported at listeners after bindings have moved.
3. Record one or more short videos demonstrating your tool working. Upload these videos with your submission.
4. Display the state of your nodes at the beginning and end of the demonstration, including bindings and listeners.
5. Make sure that your name appears at the beginning of the video. For example, display the contents of a file that provides your name.

Finally note any other changes you made to the materials provided, with an explanation.

Remember the format of the submission: A zip archive file, named after you, with a directory named after you. In this directory, provide these files: a zip archive of your source files and resources, a server jar file, a report (rubric) for your submission, and videos demonstrating your app working. Failure to follow these submission instructions will adversely affect your grade, perhaps to a large extent.