Goal:
Provide a simple way of externalizing bootstrap nodes. System admins and devops should have an easy way to change bootstrap nodes without having to touch the actual application configurations.

Summary:
Utilizing the DNS Service Resource Records (http://en.wikipedia.org/wiki/SRV_record), it is possible to fetch a list of bootstrap nodes through a single "service record" identifier.

Process:
1) Before bootstrapping the application, the network administrator configures SRV record entries like this:

_cbmcd._tcp.example.com. 0 IN SRV 20 0 11210 node2.example.com.
_cbmcd._tcp.example.com. 0 IN SRV 10 0 11210 node1.example.com.
_cbmcd._tcp.example.com. 0 IN SRV 30 0 11210 node3.example.com.

_cbhttp._tcp.example.com. 0 IN SRV 20 0 8091 node2.example.com.
_cbhttp._tcp.example.com. 0 IN SRV 10 0 8091 node1.example.com.
_cbhttp._tcp.example.com. 0 IN SRV 30 0 8091 node3.example.com.

2) A utility class (or directly baked into the bootstrap process) takes the service ID "example.com" as an argument and looks for both _cbmcd.tcp and/or _cbhttp._tcp. Depending on what we find, we can bootstrap through
CCCP (11210) or fallback to HTTP (8091). Of course if the user specifies different ports, use those.
From the given list, the regular bootstrap process can be run.

- Note that as part of DNS SRV, both weighting and priorities are supported. While weighting is somewhat more complicated (because the value would tell you the percent chance that it gets used), implementing priority is straightforward and makes sense. What should happen is that lower priority gets put first in the list, so its sorted ascending by the priority value. In our example case above, we would get it sorted 10, 20, 30… so node1, node2, node3.

- Also, since IO errors could happen all the time (dns not reachable,…) docs and the process should always think about the case what happens then. Most of the time there would need to be a fallback to a static list, or fail appropriately.

Sample Implementation:
There is a changeset up for review for the Java client, which can be found here: http://review.couchbase.org/#/c/32994/ It is implemented as a utility class, mapping from String -> List<URI>, which can be passed to the CouchbaseClient constructor. Priority sorting is also implemented. It is implemented on top of JNDI which does the heavy magic and comes shipped with the JDK, so no external dependencies are needed.
Note that it is not 100% uptodate with the latest proposal right now – I'll get that fixed.