Dynamic Configs

## Current Issues

* not well documented
  + list of dynamic configs is missing
  + shell functionality not well documented
* Anti-patterns
  + Beer beer = getIPA();

if (beer instanceof SierraNevada) { …} ← *Why? I asked for IPA, I got IPA, I don’t*

*care which color is it. If I wanted SierraNevada, I would have asked for that!*

Similarly, why should a class creating *Foo* object care which configs, dynamic or not, are used by Foo internally, and that it should be registered as observer.

*Solution:* Objects should register themselves with ConfigurationManager to get updates.

* + In current design of ‘creator registering observers’, observers have to inherit from *ConfigurationObserver* to separate them from non-observers. This has also led to one-function-catches-all-updates situation which has it’s own drawbacks explained below.  
    *Solution:* Inheritance doesn’t seem like the right design here and if objects are responsible to register self, we can remove it.
* Catch all function: *onConfigurationChange()*
  + Easy to break functionality of an existing dynamic config by accessing the configuration someplace in code and forgetting to add update handling logic to single function.
  + Tightly couples unrelated configurations

|  |
| --- |
| onConfigurationChange(conf) {  x = conf.get(X);  y = conf.get(Y);  z = conf.get(Z);  ....  } |

In the above example, if updating Y fails and an exception is thrown, Z and other downstream configurations will be skipped.

|  |  |
| --- | --- |
| In class Foo: onConfigurationChange(conf) {  x = conf.get(X);  y = conf.get(Y);  z = conf.get(Z);  } | In class Bar:  onConfigurationChange(conf) {  z = conf.get(Z);  y = conf.get(Y);  x = conf.get(X);  } |

Moreover, if a set of dynamic configurations is being updated in multiple places, it is possible that configurations get updated only partially depending on relative ordering with the failing configuration. For example, in above table if Y fails, X and Z will only be partially updated.

* + Can not report configuration level success and failures if there is only one function
  + Can not support filtering configurations. For e.g. update only *foo.bar, or foo.\**  
      
    *Solution:* Update each configuration in an individual function unless it is tightly coupled with others. And add ‘*getConf(conf name, default value,* ***update\_handler****)’.* More details below.
* Shell user experience (*update\*config()*)
  + no progress updates
    - Updates on which machines were pending/in progress/done/failed?
    - Which configurations’ updates were pending/in progress/done/failed?
  + not even overall success or failure is reported
  + (minor) no command like `*update\_config ‘X = Y*’` to ephemerally update specific configuration.  
    (non-persistent configuration changes are generally bad idea because they can easily introduce bugs and are hard to debug since changes vanish on cluster restart, nonetheless, such a function is really useful to SREs in some cases)

## Proposed solutions

1. Improve documentation
   1. document list of dynamic configuration

(maybe somehow automatically update the list from code)

* 1. Update Apache HBase Reference Guide

1. Code changes
   1. *ConfigurationManager* should know which configurations are dynamic so it can
      1. detect changes in dynamic configurations and notify which configs changed along with with old & new values (helps user avoid unwanted changes)
      2. can detect changes to non-dynamic configuration and notify that they’ll be ignored (can save user lots of trouble from having to figure out why certain changes are not taking effect)

New HConfig enum containing name of config, default value, isDynamic, description, etc will be added. It will be single source of truth for any information related to configurations. Eventually, all get\*(“foo.bar”,…) calls will be replaced by get\*(HConfig.FOO\_BAR).

* 1. Add new function: *get\*(HConfig.FOO\_BAR, update\_handler)*  
     To avoid ‘forgetting’ handling of updates, value of dynamic configurations should be queried via special function which takes in a non-null callback as third argument which will be called if and only if “foo.bar" configuration changes.  
     No changes needed for non-dynamic configurations. However, if get\*() (single argument non-handler version) is called with a dynamic configuration, it can return null and/or throw exception.



* 1. Instead of a single function to update all configuration [*onConfigurationChange()*], design should support updating single configuration/set of configurations in isolation. For example, UpdateFooBar() function in the diagram above. This will:
     1. allow configuration level success/failure reporting. eg.

*foo.bar successfully updated.*

*zing.pooh update failed to update in handler ‘winnieThePooh’*

*the.lion.king failed to update in handler ‘Simba’*

* + 1. prevent tight coupling with other configuration updates
    2. [stretch] allow filtering of which configuration to update.

[stretch] report anomalies in configuration changes detected across servers/RS as warnings. For example. *foo.bar* changed to 10 on one machine and to 100 on all other machines. Probably a misconfiguration?

[stretch, maybe be impossible] Compile time check to verify right getConf() function is  
called for dynamic and non-dynamic configurations.

## Testing

TODO: how can this framework be tested.

some basic ideas:

- add check: handler registered for only dynamic config X should not call getConf(Y) where Y is another dynamic config.

- no duplicate names in HConfig

## Rough plan of attack

1)

* Add HConfig and modify ConfigurationManager interface
* no changes related to dynamic configs will be made
* only few classes will be moved to new interface initially

2) Add components of new dynamic config framework: get\*(conf, handler), Map<conf, handlers>, etc and use this new functionality in classes changed above in step 1.

3) Move all classes which are using dynamic configs to new framework

4) Delete old dynamic configs framework: ConfigurationObserver

5) Improve shell commands reporting. Maybe add new command “*update\_config ‘X = Y*’”

6) Testing framework for dynamic configs

7) Misc: automate HConfigs → Ref Guide Configuration documentation (when all configs are in HConfigs, we should be able to delete hbase-default.xml).

8) Document dynamic configs framework

## Open questions

1. Updates will happen one configuration at a time (iterate over registered handlers) instead of one component at a time (*onConfigurationChange()* updating all configs together). Can this lead to any issues?
2. Is it possible to have a scenario where multiple configurations need to be updated together? Say HFileBlock requires updating x and y, but old x is not compatible with new y, making the intermediate stage risky.  
   In yes, one single function can be added as the handler for all configurations that need to be updated together. *ConfigurationManager* can be smart and call any repeated handlers only once. Might need locking.
3. Configuration changes which are not backward compatible should be avoided. For example, when a region moves from an updated RS to a non-updated RS, it can cause troubles. More thoughts?

**Ignore stuff below**

- partition configs into master/rs/both: for users to better understanding effects of dynamically changing configs, and generally good too

- choose top 20 configs to make dynamic

- working: script (shell?) sends signal to all masters/RS (serial/parallel?) (exits/waits to report?). See how it’s done today.

- inner class ConfigurationUpdater to keep all functions which’ll handle conf updated. will keep pointer to outer class

- outer class: add initialize() method which will first instantiate ConfigurationUpdater and then other components depended on dynamic confgs. Handlers will be CU’s function