Current urls that do things

will flesh this out later. substitute your IP address for the 192.168.1.17 shown (and port address if different)

- http://192.168.1.17:54327/StoreView
- http://192.168.1.17:54327/Remote?Node=/192.168.1.17:54328
- http://192.168.1.17:54327/GetQuery
- http://192.168.1.17:54327/Get?Key=
- http://192.168.1.17:54327/Put
- http://192.168.1.17:54327/PutValue?Value=&Key=&RF=
- http://192.168.1.17:54327/PutFile
- http://192.168.1.17:54327/Timeline
- http://192.168.1.17:54327/ImportQuery
- http://192.168.1.17:k54327/ImportFolder?Folder=&RF=
- http://192.168.1.17:54327/ImportUrl?Url=&Key=&RF=
- http://192.168.1.17:54327/DebugView
- http://192.168.1.17:54327/DebugView?Prefix=
- http://192.168.1.17:54327/ProgressView
- http://192.168.1.17:54327/Network
- http://192.168.1.17:54327/Shutdown
- http://192.168.0.37:54321/Parse?Key=____582ed371-1b3a-4b4c-ab64-46e68a61604a

To get stacktraces of the nodes

- http://localhost:54321/DbgJStack.json
- http://localhost:54321/DbgJStack

Files to/from H2O

A PutFile will create a key for you if you don't specify one. You can put from a normal network path, or `hdfs://192.168.1.151/` to go to the 0xdata hdfs, thru the 192.168.1.151 namenode. `homer.0xdata.loc` can also be used for the namenode.

A GetFile currently sends the file to the browser, which creates a popup window like the normal "Save As" thing in a browser. You can specify where to save the file locally then.

Random Forest

You can specify depth and ntrees for an RF. Currently depth is ignored? Defaults are used if depth or ntrees isn't specified

- http://192.168.0.37:54321/RF?depth=30&ntrees=20&Key=____aad91-3d73-436e-bd44-86e356950633
- http://192.168.0.37:54321/RFView?Key=436f6e667573696f6e4d6174726978206f6620____aad91-3d73-436e-bd44-86e356950633
- http://192.168.1.17:54327/RFTreeView?Key=0e01c0a8011138d4____2662a1ae-6d51-4aaf-90b1-63d434516224&origKey=____a.hex
- http://192.168.1.17:54327/Inspect?Key=____a
- http://192.168.1.17:54327/Parse?Key=____a&Key2=____a.hex

Linear Regression

- http://192.168.0.37:54321/LR?depth=30Key=____aad91-3d73-436e-bd44-86e356950633

&colA= and %colB= can be added to specify two feature columns. The two columns should cover more than one point, otherwise, NaNs will be returned.

Logistic Regression

To run logistic/linear regression via REST API, go to **/GLM** and supply following arguments:

- **Key** - (**required**) - key of the .hex data to run regression on
- **Y** - (**requited**) - column id / name of the response variable, if you run logistic regression it's values currently **MUST** be from {0,1}
- **X** - (**optional**) - list of columns used as an input vector, either names or column indexes (or mixture of both), separated by ','. Ranges can be provided by supplying two values separated by ':', e.g. X=1:3,5:10 is equivalent to X=1,2,3,5,6,7,8,9,10. **Default** is all columns except for Y.
- **-X** - (**optional**) - negative X, these columns will be subtracted from the set of selected columns. Thus, selected columns = X \ -X. **Default** is empty set.
- **family** - (**optional**) - either "binomial" for logistic regression or "gaussian" for linear regression. **Default** is gaussian.
- **xval** - (**optional**) - cross validation factor. suply 10 for 10fold cross validation. **Default** is 0 (no crossvalidation). Currently supported only for logistic regression.
- **threshold** - (**optional**) - decision threshold used for prediction error rate computation. Only used by logistic regression. **Default** is 0.5.
- **norm** - (**optional**) - norm for regularized regression. Possible values "L1", "L2", "NONE". **Default** is "NONE". If regularization is used, data will be **REGULARIZED** so that it has zero mean and unit variance.
- **lambda** - (**optional**) - argument affecting the regularization, **Default** is 0.1.
- **rho** - (**optional**) - used only with L1. Enhanced Lagrangian parameter. **Default** is 1.0.
- **alpha** - (**optional**) - used only with L1. Over-relaxation parameter (typical values for alpha are between 1.0 and 1.8). **Default** is 1.0. **example URLs:**
- http://127.0.0.1:54321/GLM?Key=____prostate.hex&X=6,3&Y=1&family=binomial

runs logistic regression using columns # 6 and #3 as X vector and column # 1 as response variable

- http://127.0.0.1:54321/GLM?Key=____prostate.hex&Y=CAPSULE&family=binomial&xval=10&threshold=0.4

All the columns are used for regression, except for Y (CAPSULE). 10 fold crossvalidation will be produced using decision threhsold 0.4.

Starting a cloud from the command line

The "discovery address" is created with a hash, for a unique address used by UDP traffic. It is unique in case multiple clouds exist on the same network. The ports only tolerate traffic of the specified protocol. Point your browser to the HTTP port.

Every H2O instance uses 3 ports, they are allocated sequentially given the starting port. if another H2O is still alive, you may collide on ports.

```
[h2o] HTTP listening on port: 54327, UDP port: 54328, TCP port: 54329

The Cloud 'kevin' is Up (v0.3) on /192.168.1.17:54328, discovery
address /231.19.132.129:59155
```

It's best to make sure no java processes are running from a prior H2O when you start (killall) Also: the ice keys are sticky between H2O's. Here, the ice dir is in /tmp. There are two kinds of files created by H2O in the ice dir.

Warning: don't use this directly on machines with HDFS processes (192.168.1.150-192.168.1.161) since they are java processes there.

```
killall java

rm -v -f -r /tmp/ice*rm -v -f -r /tmp/File[0-9][0-9]*tmp

java -ea -jar ./build/h2o.jar --port=54327 --nosigar --
ice_root="/tmp/ice."
```

Launch with HDFS

```
java -ea -jar ./build/h2o.jar --port=54327 --nosigar --
ice_root="/tmp/ice." -hdfs hdfs://192.168.1.151 -hdfs_version cdh4 -
hdfs_root /datasets
```