Manifest Rollout Strategy

Manifest rollout is done by environment (or branch in Octopus). When a puppet module is deployed to an environment , all machines in that environment are exposed to the manifest. In some cases we can turn features on and off using facter facts . Nodes are defined to an environment with the following:

* In /etc/puppetlabs/puppet/puppet.conf look for the environment = parameter.
  + Example: environment = labbeta

If you need to change an environment, this must be done as root so a request would need to be submitted to linux engineering. It is recommended that you have a server built dedicated to a specific environment instead of constantly changing the value in puppet.conf.

Lab - AETT

1. labrand - any updates (commits) automatically deploy to labrand
2. labalpha
3. labbeta

Core - AETH

1. test - code migrations from labbeta automatically deploy to test
2. dev
3. qa
4. prod

Cloud lab - heht

1. lclddev - any updates automatically promote to lclddev
2. lcldqa
3. lcldprod

Cloud - heh

1. cldtest - any updates automatically promote to cldtest
2. clddev
3. cldqa
4. cldprod

Lifecycle

All development is pushed to the labrand branch in AETT and lclddev in HealtheHostT. Users can then self-service promote their code through Octopus in AETT/AETH and HEHT/HEH.

Environment Lifecycle

1. Lab Development

1. Initial development environment – known as the build branch
2. Used to code/debug all infrastructure code
3. Confines outages to single lab devices if mistakes are made
4. Local GIT repository on developers workstation/laptop

2. Lab Deploy

1. Move IaC code into the lab deploy GIT repository AETT
2. Used to test/debug puppet master/slave configurations
3. Confines outages to lab devices if there are master/slave issues
4. Code approval process
   * Team review of IaC code
   * Approve/Reject
     + Approve – trigger push to Development
     + Reject – send email notification back to developer

3. Development

1. Move IaC code into DEV branch in AETH
2. Developer reviews results on DEV devices
3. Code approval process
   * Team review of DEV testing result
   * Approve/Reject
     + Approve
       1. Manager approval of IaC code
       2. Trigger push to QA
     + Reject – send email notification back to developer

4. QA

1. Move IaC code into QA branch in AETH
2. Developer reviews results on DEV devices
3. Code approval process

* Approve/Reject
  + Approve
    - Manager approval of IaC code
    - CAM releases code as part of change control process
      * Trigger push to PRO
  + Reject – send email notification back to developer

5. Production

1. Developer reviews code release
2. Follows standard go/no go change control process

# Hiera Manager (web app)

In late 2016, we created a web-based front-end for the data in CouchDB. Below is information on how to use this application, and the technical details of it's setup.

## **Getting Started**

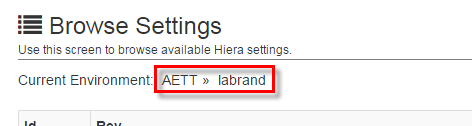
To begin, navigate to the main URL, which is:

* <https://automation.aetna.com/hieramanager/#/>

By default, you will need to be a member of the AETH webeng or zzwebeng group to edit data. Once you have navigated to the web page, click on the Browse Settings navigation to begin:



from this screen, first drill-into the Aetna domain, and then database that you wish to view:

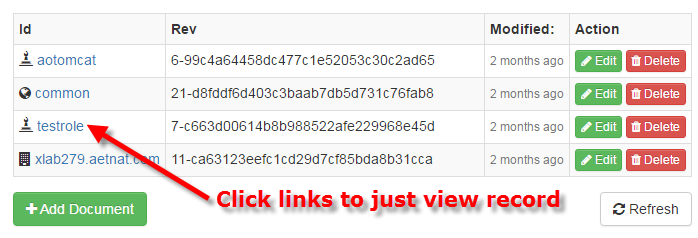


You can click either the "AETT" or the "labrand", as shown above, to show a drop-down of available options. Once both a domain and database have been chosen, a list of "documents", or records in that database will be listed.

On this page of documents, you can click the link to view the document, or click the corresponding edit or delete button for that record.

##### CAUTION

If you do not have permission to edit or delete a document, those buttons will be missing for you. If there is a record that exists in this database, but you do not have permission to view it, it will not be in this list. For more information, see setting permissions, below.



You may also click "Add Document" to create a new record in the current database.

## **Types of Documents**

There are 3 kinds of records:

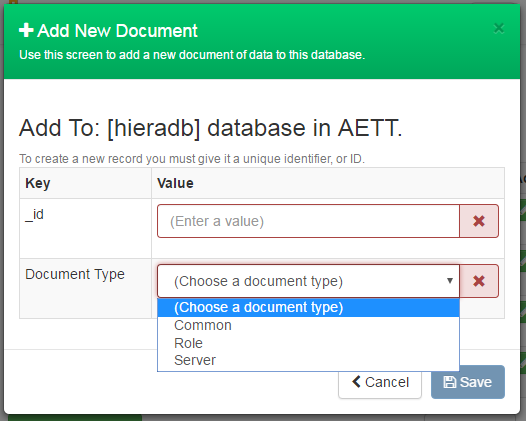
1. **Common** - metadata that applies to targets within this scope.
2. **Role** - metadata that applies to targets who are in the specified role.
3. **Server** - metadata that applies to a specific machine name.

You can determine the type of record, by the icon:



## **Creating a new document**

From the main "Browse Settings" acreen, you click the "Add Document" button, you are confronted with a dialog box:

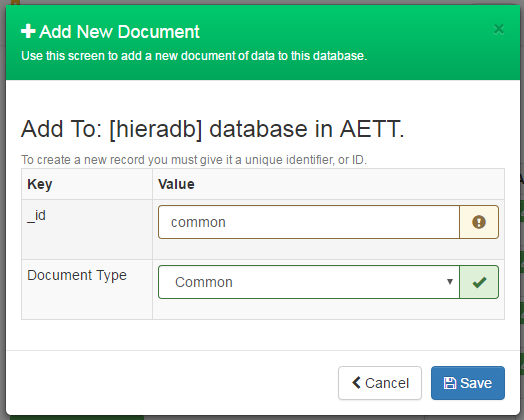


The document type is as-discussed earlier. This defines the scope how this record will apply. The "\_id" field is the main, unique identifier of this document, in CouchDB.

If the "\_id" is not valid, you will see the red coloring and the X to the right of the field. If the "\_id" is valid, you will see green and a checkmark.

##### CAUTION

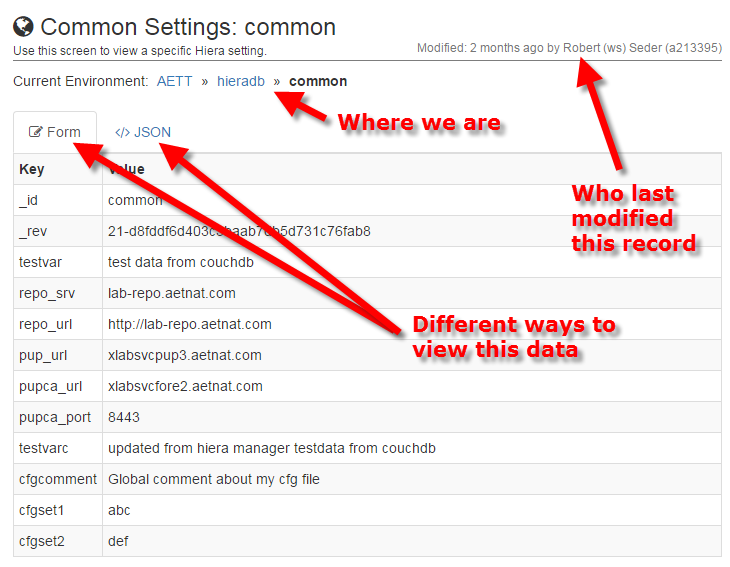
**Overwriting Existing Documents** Since CouchDB is a simple document database, if you name the "\_id" the same ID as an existing record, you will see the warning stating that you will overwrite the existing record. If you hover the exclamation icon, it gives you more detail.



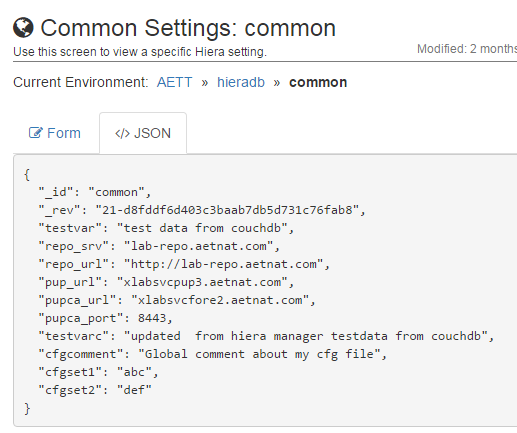
Since the "\_id" is like a primary key, if you reuse an existing one with the same name, you will overwrite the existing record. This is similar to overwriting existing keys, also.

## **Viewing a Document**

From the "Browse Settings" screen, when you click a link on the left to view a document, you are shown a screen with two tabs:

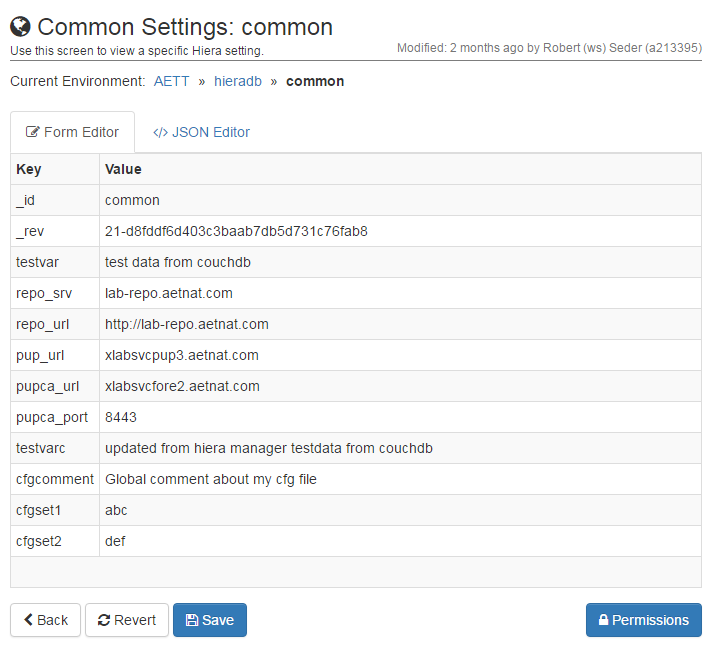


Here you can see where this data resides (domain, database, and document ID), who last modified the record, and you can view the data in a formatted way, or in raw JSON format:



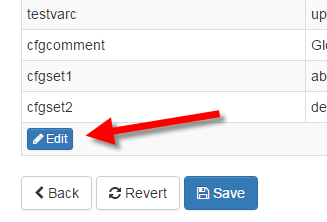
## **Editing a Document**

From the "Browse Settings" screen, when you click an Edit button on the right to edit a document, you are shown a screen with two tabs, similar to the read-only view:



There are couple of differences from the read-only view. The "Revert" button lets you discard any unsaved changes you've made and go back to the version of this document, how it currently exists in the database. Also, there is a permissions button, which is described in more detail below.

In an attempt to make the user interface less-confusing, you don't see edit/clone/delete options until you hover over something that can be modified. The root level, is the document-itself, so there is just an Edit button here:



However, since "hiera" can be a "hierarchy", you may have fields, nested within fields.

##### TIP

**Nesting Levels**

As of this writing, you may easily edit up to 4 levels of nesting - from the root of the document. Meaning you could have: Object3 nested within Object2, nested within Object1 which is nested with Object at the document-level.

So, for any nested objects, you'll see this as you hover over that item:



This allows you to edit, delete, or clone this field.

##### NOTE

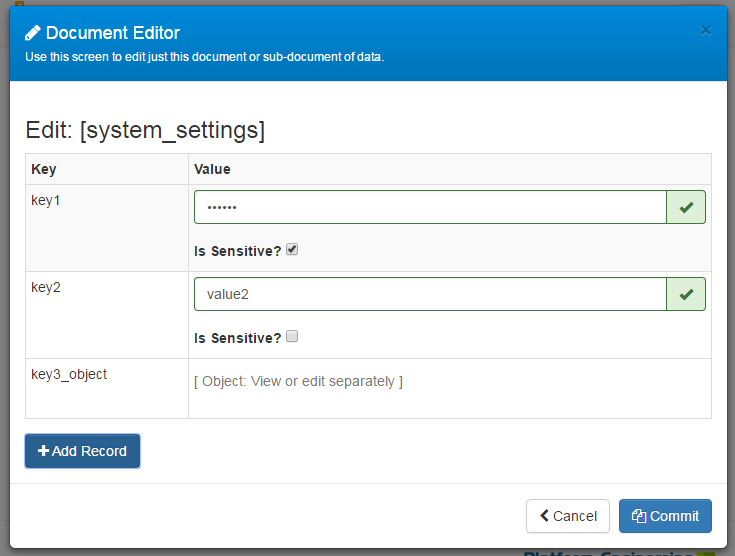
**Cloning a Section**

Cloning does a deep-copy of an object and prompts you for a new name. This is useful for creating several similar sets of data, or for making a backup copy of data while you work on a new idea.

Also see the "Cloning Data" section, below.

## **Editing Data**

When you click on the Edit button to edit that specific data set, you have several options:



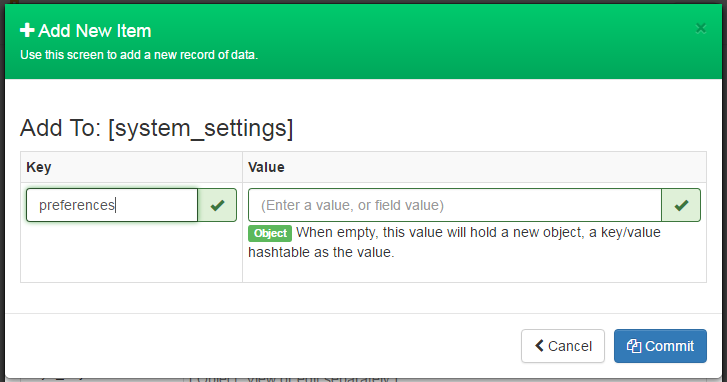
First, if the "Value" is a text or number, you can type in the new value. If the value is an "object" (a nested set of values), then you can't edit it from here. You need to go back out to the edit screen and click "Edit" on that item.

##### CAUTION

**The "Is Sensitive" Indicator**

The "Is Sensitive" indicator is really just for the user interface. It does NOT encrypt or obfuscate the value in the JSON view, nor in CouchDB. However, it can be useful if you are sharing your screen or doing a presentation and don't want to needlessly show passwords or other valuable information.

You may also add a record to this particular data set by clicking the "Add Record" button.

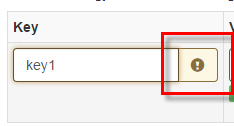


For the key, this is similar to creating a document. This key MUST be unique. If you put the name of an existing key, the field will turn amber and you will get a warning that if saved, this will overwrite the key with the same name. This is the nature of a key/value pair document structure like this.

##### CAUTION

**Overwriting Existing Keys**

If the "Key" value is the same name as an existing key, you will see an amber warning and exclamation in the field, like this:



That means that if you commit the changes, it will OVERWRITE the data of the existing key. This is similar to overwriting existing documents, also.

For the value, either enter a discrete value: a string, a number, a date, etc. Or, if you leave "Value" empty, it will be saved as a nested object, a key/value pair dictionary to which you can nest more values.

##### TIP

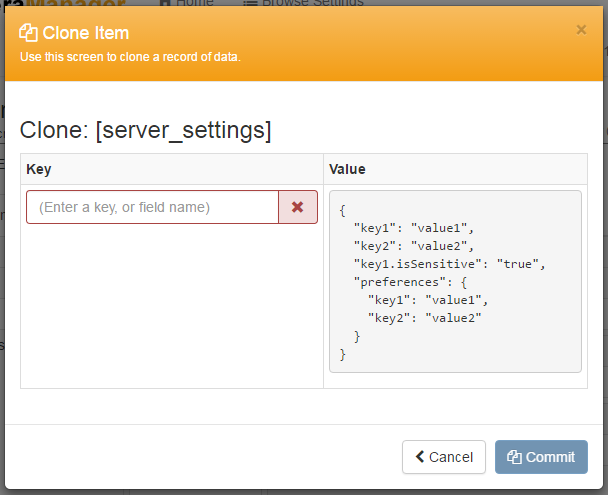
**Object and Discrete Values**

If the value field is empty, the "value" will store an object containing a key/value pair dictionary - a sub-object. if the value field has a value, like "test", or 123 - then it can only store that single, discrete value.\*\*

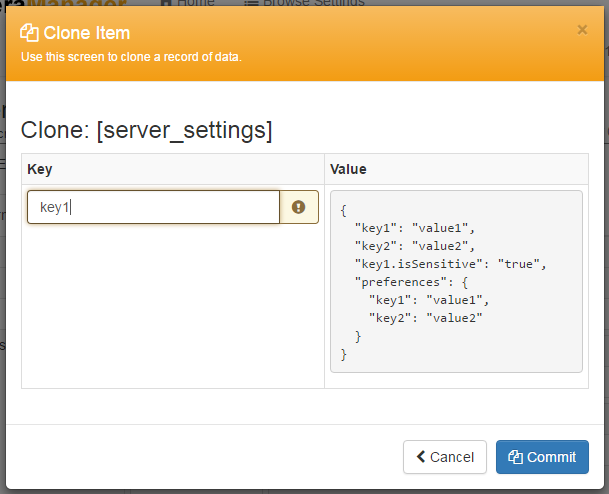
When you "Commit" this data, you are merely committing the data to the current, working model that is open. It is NOT saved to the database until you click "Save" at the document level.

## **Cloning Data**

When you click on the Edit button to edit that specific data set, you are confronted with this dialog:

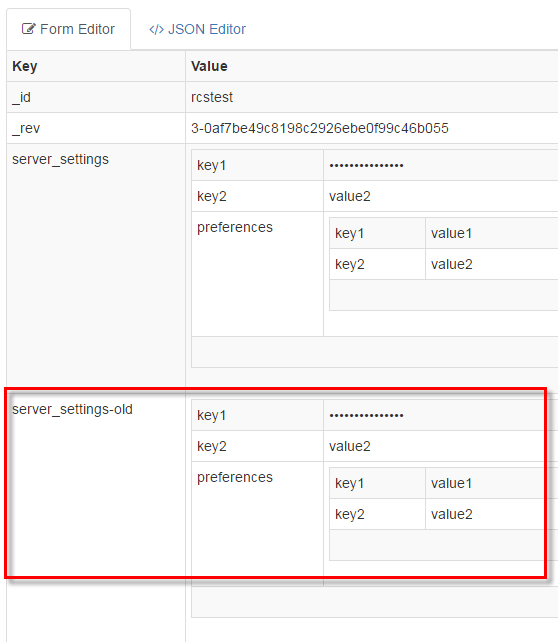


and notice that if you were to type in the name of existing key, you see the warning:



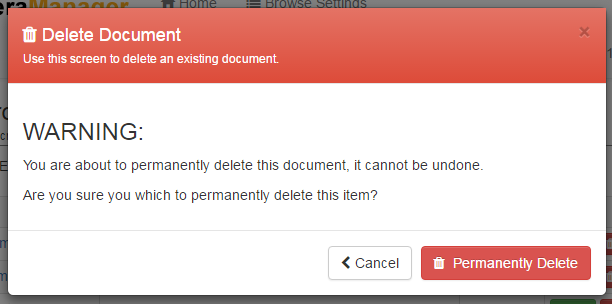
Note the amber color and exclamation mark for that field. That is indicating that it will overwrite that key, with the data that is in the Value area. For more information, see overwriting existing keys.

To effectively clone this item, change the "Key" to something unique. For example "server\_settings-old". You now have a clone of that data set:



## **Deleting a Document**

While on the main Browse Settings screen, if you have permission to delete a document, you will see a Delete button. When you click it, you'll see a confirmation like this:



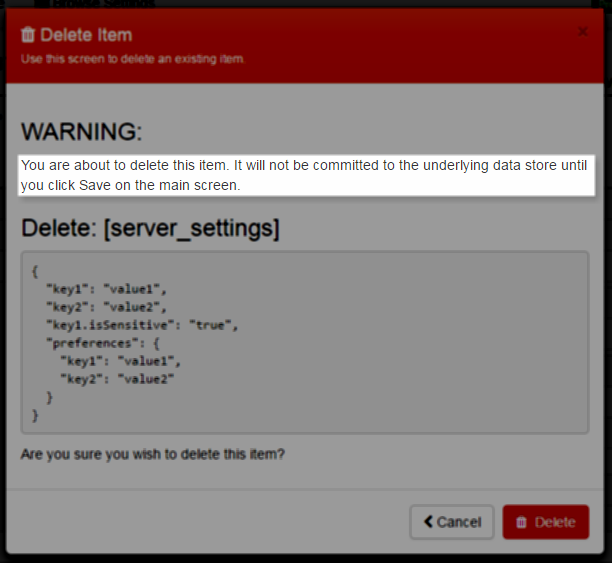
and as the dialog suggests, this does permanently delete the document from the database, immediately.

## **Deleting Data**

When you are editing a document, when you hover over a section, you will see the option to delete a section:



The idea is that you are currently working on an in-memory, in-browser version of a document. It is not committed to the database until you click "Save". So, when you click the Delete button here, you'll see a confirmation like this:

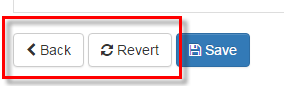


where it confirms that you are not deleting this data from the database just yet. This will commit the changes to this in-browser, in-memory version of the document you are working on. You must click "Save" to commit it to the database.

##### NOTE

**Reverting Your Changes**

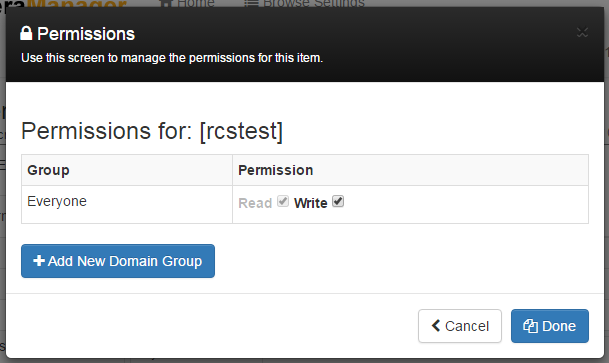
Since you are editing an in-memory, in-browser version of the document, nothing has been written to the database yet. If you've made in error in deleting a value or whole section, you can always click "Revert" at the bottom of the screen to re-load the changes from the databases, discarding your changes - or you can also click the Back button to go back to Browse Settings - again, discarding your current changes.



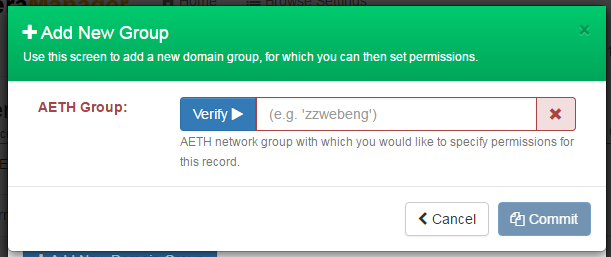
## **Setting Permissions**

Permissions are set at the document-level. You cannot set permissions on a per-field level. By default, new documents are created with Everyone being able to read and write.

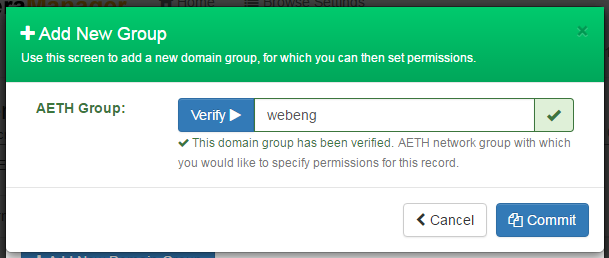
To modify the permissions for a document - from the main Browse Settings screen, click the "Edit" button to edit the document. In the bottom-right, click on the "Permissions" button. You will be confronted with a dialog like this:



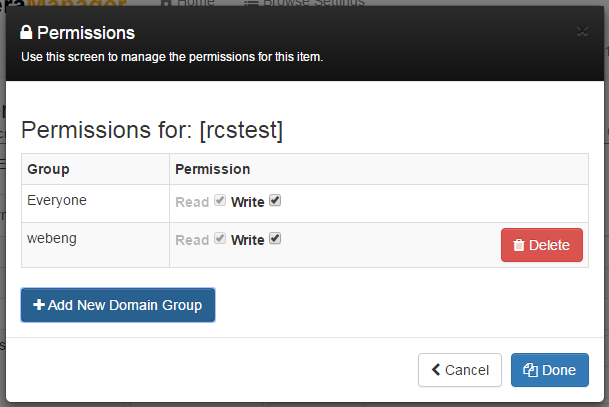
To add group authorization, click on the "Add New Domain Group" button to see:



You then type in a name of an AETH group and click Verify:



you can then commit those changes:



From here, you might remove read and write access for everyone, for example - or set up permissions however you wish.

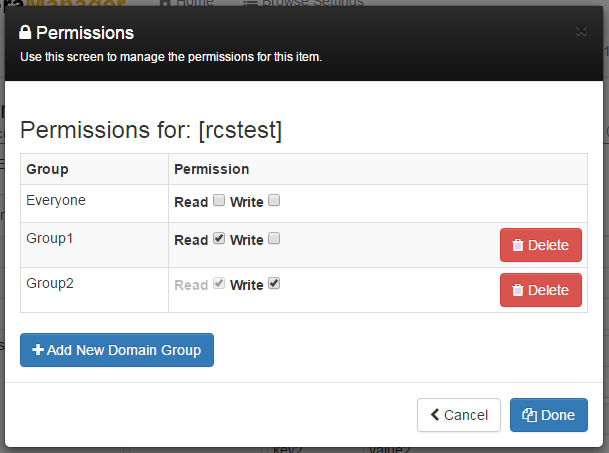
##### TIP

**Deleting Group Permissions**

You might note that there is a Delete button next to group records, but not for "Everyone". You may remove write and then remove read access, but you may not remove the default "Everyone" group.

Since write implies read, when you uncheck "Write", the "Read" checkbox will become available to un-check as well.

Permissions work in an additive way. This will take the maximum privilege of all available permissions that are defined. For example, if you are in Group1 and Group2 - permissions are defined as:



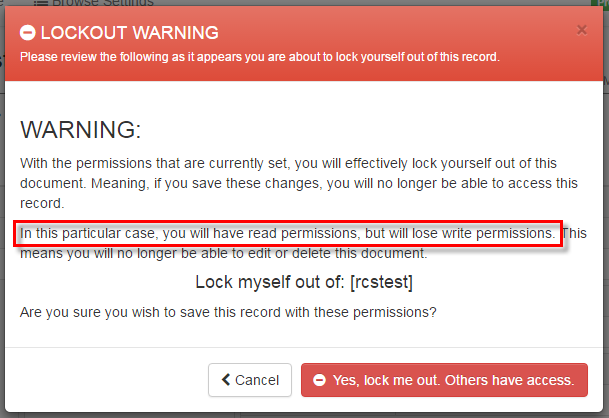
The system takes the most amount of privilege you have - in this case, that is for Group 2, which has read and write.

##### CAUTION

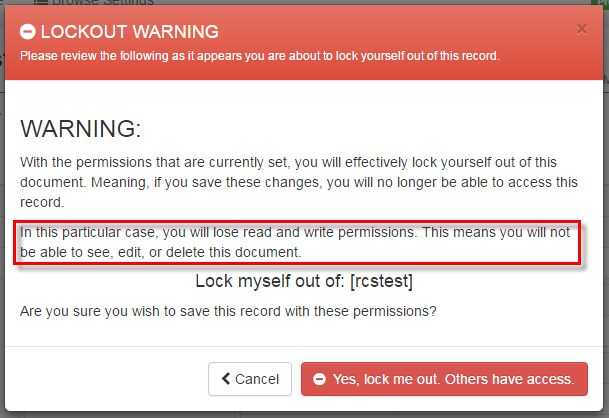
**Dealing With Account Lock-Out**

One key issue you can easily run into, is locking yourself out of your document. For example, you could take away Everyone:W permission, and then make it so a domain group only has write - and you aren't in that group. To prevent these possibilities, you will get warning messages if your changes will result in your being locked-out of your own record.

If the net result is that you no longer have write permissions, you will see a dialog like this - not when you exit the Permissions dialog - but when you click the "Save" button for the document:



If the net result is that you will no longer have read NOR write permissions, you will see a dialog like this:



Having a strategy of who should have access to read and write the record, helps - and just be mindful to not lock yourself out of your own record. Please note that if you do lock yourself out, someone who does have permission must use this app to correct the permissions - or someone would need to modify the document by hand, in CouchDB.

## **Technical - Metadata Fields**

Since this application is meant to be a thin layer on top of the CouchDB data, we implemented a way to include "hidden" metadata within the document. Hidden, meaning that the UI is specifically looking for these fields, and will hide them from view - in this Hiera Manager application ONLY. They exist and are viewable if you are viewing or modifying the data directly from CouchDB.

Following the concept of a "." prefix making an item hidden in Linux, several keys with a "." prefix are used within this particular application. Here is an example document, with the placement of these meta fields (with the dot-prefix):

**Example CouchDB Document**

{

"\_id": "rcstest",

"\_rev": "3-0af7be49c8198c2926ebe0f99c46b055",

"server\_settings": {

"key1": "value1",

"key2": "value2",

"key1.isSensitive": "true",

"preferences": {

"key1": "value1",

"key2": "value2"

}

},

".permissions": {

"Everyone": [

"r",

"w"

],

"webeng": [

"r",

"w"

]

},

".documentType": "server",

".lastModifiedByAetnaId": "A213395",

".lastModifiedByName": "Robert Seder",

".lastModifiedDate": "2017-02-07T11:48:51.4544663-05:00"

}

Here is a closer look at each:

* "**.documentType**" can be: common, role, or server. This is the type of document this is, as described above.
* "**.lastModifiedByName**" and "**.lastModifiedByAetnaId**" are the name and credentials of the last person to modify this document, as reported by the server. This information is stripped out from the browser client.
* "**.lastModifiedDate**" is the [ISO 8601](https://www.iso.org/iso-8601-date-and-time-format.html) formatted date, including the Eastern Timezone. It was decided all data in this database will use this timezone for all metadata timestamps.
* "**.permissions**" stores permission objects which have arrays of "r" read or "w" write permission. By default, new records are created with Everyone having read and write access. See "Setting Permissions" above for more information on this, and about locking yourself out of a record.
* "**.isSensitive**" can be paired with any "key" entry at any level in the hierarchy. This is nothing more than an indicator for the UI to hide the value for that key. In a textbox, it will show as a password field, and in read-only form, it will show as asterisks. In the JSON view however, the value WILL still be viewable. See the "Is Sensitive" warning, above.

If you are using this Hiera Manager application, you won't see any of these hidden fields.

##### CAUTION

If you are editing this document directly from CouchDB, please be careful in modifying this data because if invalid, it could potentially affect the Hiera Manager who depends on it.

## **Technical - Architecture**

This application uses Bootstrap and AngularJS on the front-end. All data that is needed is made via AJAX calls. The backend of those REST-based AJAX calls is ASP.NET Web API. That Web API makes basic REST calls to the CouchDB database.

Authentication is done SiteMinder, and authorization is done at the Web API REST endpoint - and reflected in the AngularJS front-end.

## **Hiera Search Hierarchy**

* http://couchdbserver:5984/hieradb/%{::clientcert}
* http://couchdbserver:5984/%{::environment}/%{::aetna\_role}
* http://couchdbserver:5984/%{::environment}/common
* http://couchdbserver:5984/hieradb/%{::aetna\_role}
* http://couchdbserver:5984/hieradb/common

couchdbserver list:

* AETT
* xlabsvccdb2.aetnat.com
* AETH
* xsvccdbw2p.aetnat.com
* HealtheHostT
* xlabosvccdb2.healthehostt.com
* HealtheHost
* xosvccdbw2p.healthehost.com

Parm definitions:

* ::clientcert = server name (or node name) entered as a document in couchdb
* ::environment = environment parm from /etc/puppetlabs/puppet/puppet.conf entered as a database name in couchdb
* ::aetna\_role = role parm from /etc/aetna.info entered as a document in couchdb
* common = static document name enetred in couchdb