Lab 13 Introduction to Data Bag

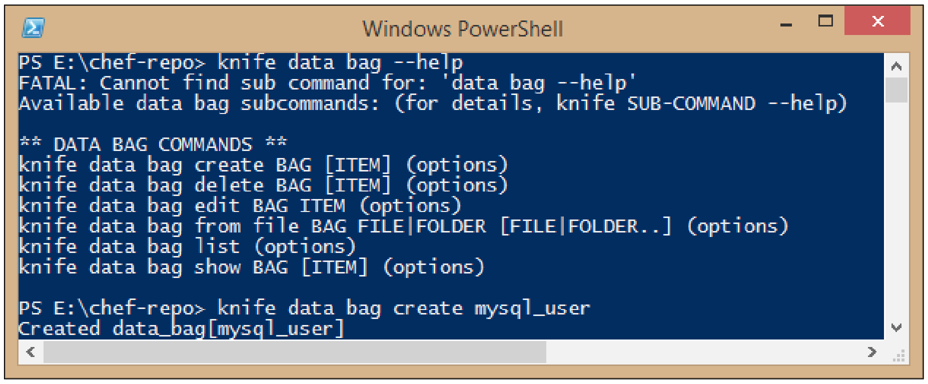
A data bag is JSON's data format global variable that is used as an index for search queries and is accessible from the Chef server. Data bag is loaded through recipes and most of the time, it contains secured information, such as passwords.

Data bag is very good for securing secret information because data bag can be encrypted easily.

The `knife data bag` subcommand is used to create, delete, edit, list, and show data bags.

The `create` argument creates a data bag on the Chef server:

`knife data bag create <<DATA_BAG_NAME>> [DATA_BAG_ITEM] (options)`

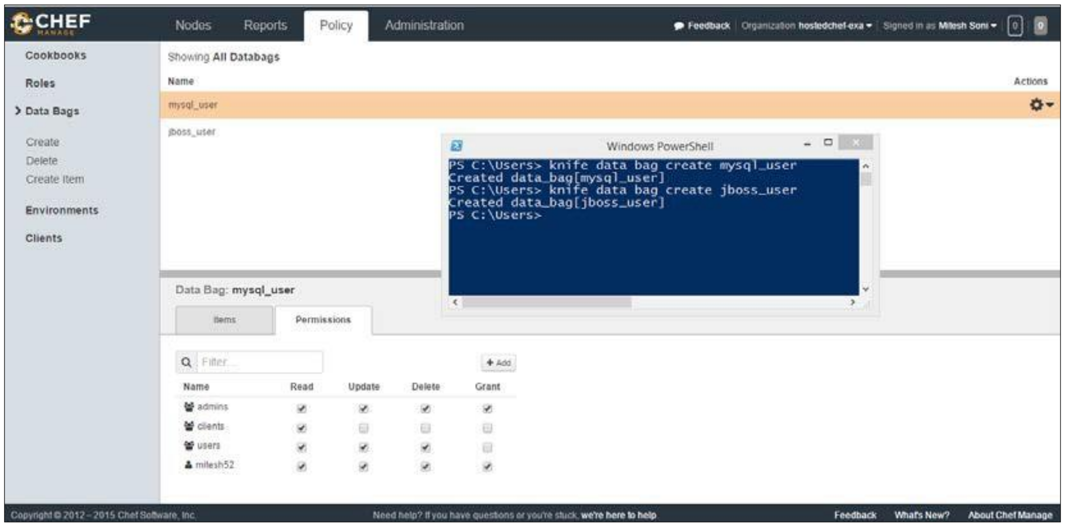


```
PS E:\chef-repo> knife data bag --help
FATAL: Cannot find sub command for: 'data bag --help'
Available data bag subcommands: (for details, knife SUB-COMMAND --help)

** DATA BAG COMMANDS **
knife data bag create BAG [ITEM] (options)
knife data bag delete BAG [ITEM] (options)
knife data bag edit BAG ITEM (options)
knife data bag from file BAG FILE|FOLDER [FILE|FOLDER..] (options)
knife data bag list (options)
knife data bag show BAG [ITEM] (options)

PS E:\chef-repo> knife data bag create mysql_user
Created data_bag[mysql_user]
```

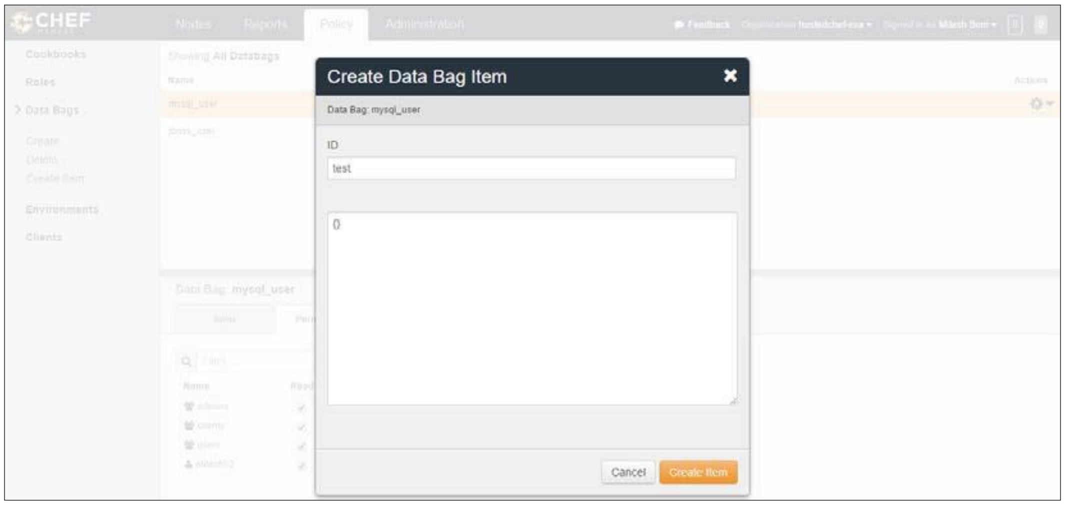
The following is a screenshot of the verification of a data bag creation on the Hosted Chef server:



```
PS C:\Users> knife data bag create mysql_user
Created data_bag[mysql_user]
PS C:\Users> knife data bag create jboss_user
Created data_bag[jboss_user]
PS C:\Users>
```

Name	Read	Update	Delete	Grant
admins	✓	✓	✓	✓
clients	✓			
users	✓	✓	✓	
mitesh52	✓	✓	✓	✓

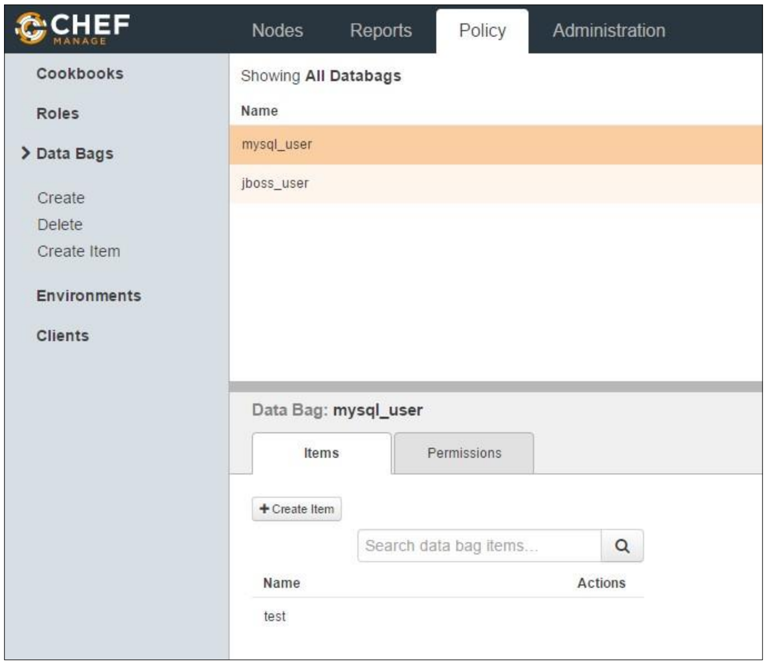
We can also create a data bag from the dashboard of the hosted Chef server. Go to **Policy**, click on **Data Bags**, and select **Create Item**.



ID: test

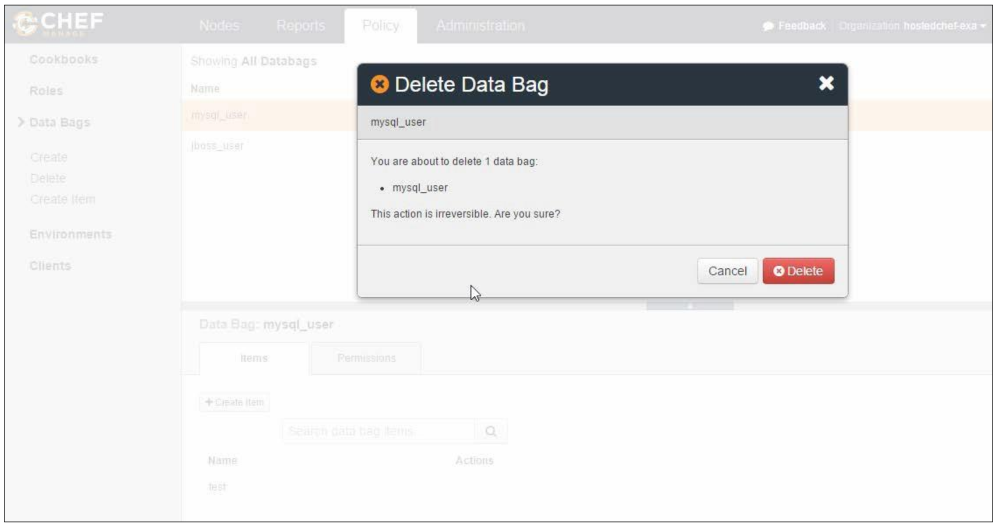
```
{}
```

We can also create an item associated with a data bag from the dashboard of the hosted Chef server.



The `create` argument deletes a data bag from the Chef server:

```
knife data bag delete <<DATA_BAG_NAME>> [DATA_BAG_ITEM] (options)
```



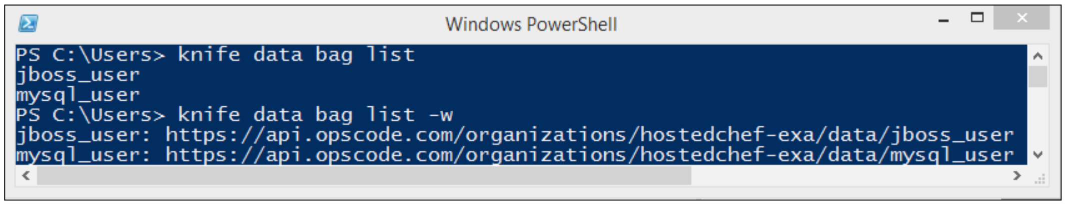
You are about to delete 1 data bag:

- mysql_user

This action is irreversible. Are you sure?

[142]

The `list` argument lists data bags that are currently available on the Chef server.



```
PS C:\Users> knife data bag list
jboss_user
mysql_user
PS C:\Users> knife data bag list -w
jboss_user: https://api.opscode.com/organizations/hostedchef-exa/data/jboss_user
mysql_user: https://api.opscode.com/organizations/hostedchef-exa/data/mysql_user
```

To find a data bag or item of a data bag, it is necessary to have a specified data bag name. Also, while the search is going on, a query named as the `search` query string should be provided.

If we want to search with the `knife` command within data bag items that are named as `admin_data`, but exclude items that are named as `admin_users`, then the `search` command would be as follows:

```
$ knife search admin_data "(NOT id:admin_users)"
```

We can also include this search query in a recipe, and the code block will be similar to this:

```
search(:admin_data, "NOT id:admin_users")
```

There can be confusion between data bag items about what is needed and what is not. A search query is the best way to avoid this confusion, although, all essential data is returned by using this `search` query. However, it can be mandatory to save everything in a data bag, but whole data is not known for information. Some examples are here to show that a recipe uses a list of search queries in a specified way inside a data bag, such as `admins`. For example, in order to search for administrator, use the following command:

```
search (:admins, "*:*")
```

Search for an administrator named `mark`:

```
search (:admins, "id:mark")
```

Search for an administrator that has a group identifier named `dev`:

```
search(:admins, "gid:dev")
```

In the same way, we can also search for an administrator whose name starts with the letter `b`:

```
search(:admins, "id:b*")
```

After successful execution of a search query, we get the items of a data bag as a result, and these items can be used as hash values.

We got `admins` and `id` as a `search` result for the preceding query. These could be used for the following query:

```
mark = search(:admins, "id:mark").first
# => variable 'mark' is set to the mark data bag item
mark["gid"]
# => "ops"
mark["shell"]
# => "/bin/zsh"
```

The following example demonstrates how the same thing can be done using explicit access of data bag fields as can be done by using the search mechanism.

A specific recipe is used to align a user with each administrator by saving all the items with the name of the data bag as `admins`. By looping all `admin` in the data bag, and after that, by creating a user resource for each `admin`.

The recipe to do this is as follows:

```
admins = data_bag('admins')
admins.each do |login|
admin = data_bag_item('admins', login)
home = "/home/#{login}"
user(login) do
uid       admin['uid']
gid       admin['gid']
shell     admin['shell']
comment   admin['comment']
homehome
supports  :manage_home => true
end
end
```

We can use the preceding recipe with some changes, such as we can make an array to store search results. The following query will `search` for all `admin` users and the result will be stored in the array:

```
admins = []

search(:admins, "*:*").each do |admin|
login = admin["id"]
admins<< login
```

[144]

```
home = "/home/#{login}"
user(login) do
uid       admin['uid']
gid       admin['gid']
shell     admin['shell']
comment   admin['comment'] home
          home
supports  :manage_home => true end
end
```