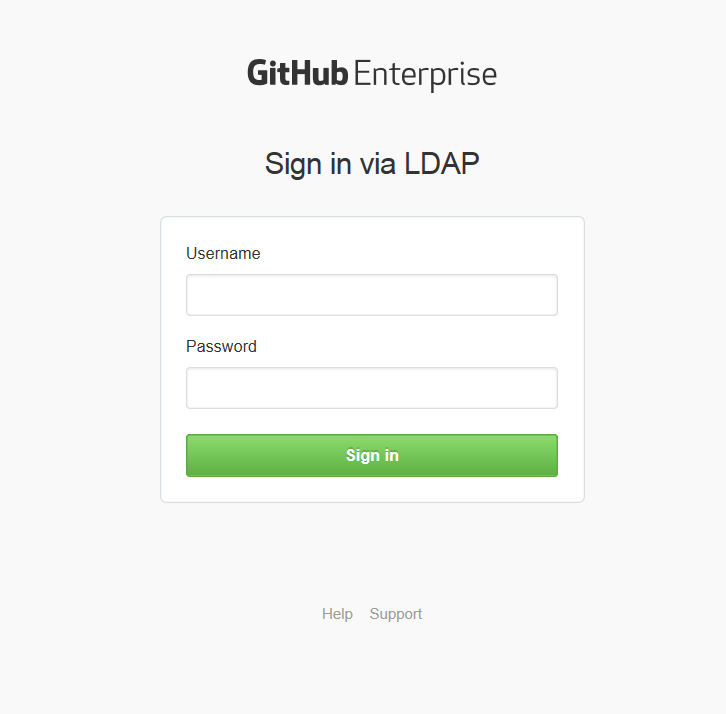
**Github documentation on admin settings, backup settings, SSH key for internally to CD team**

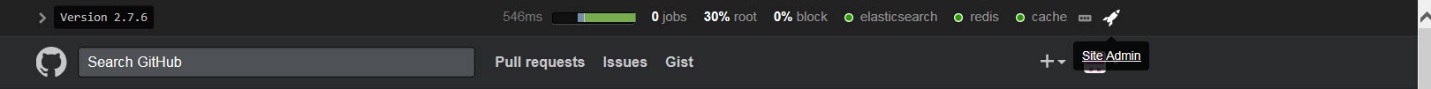
Admin Settings

GitHub Enterprise is the on-premises version of GitHub.com. which is flexible deployment options, centralized permissions, and hundreds of integrations, you and your team can do best parts of working with GitHub without any features business needs.

Login into nm GitHub by https://github.nml.com by using your LANID and PASSWORD.

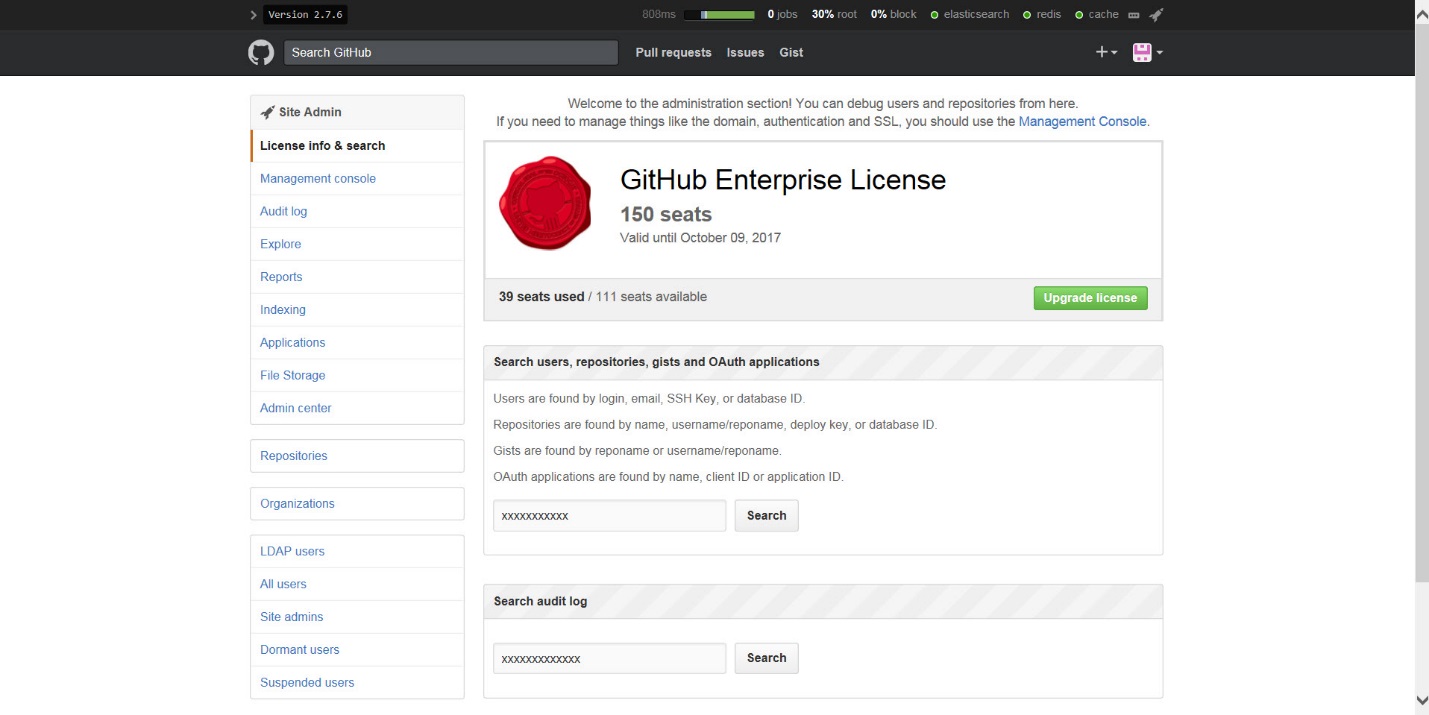


* Once your logged into github go to the site admin page by clicking the Rocket icon in the upper-right corner of the page.



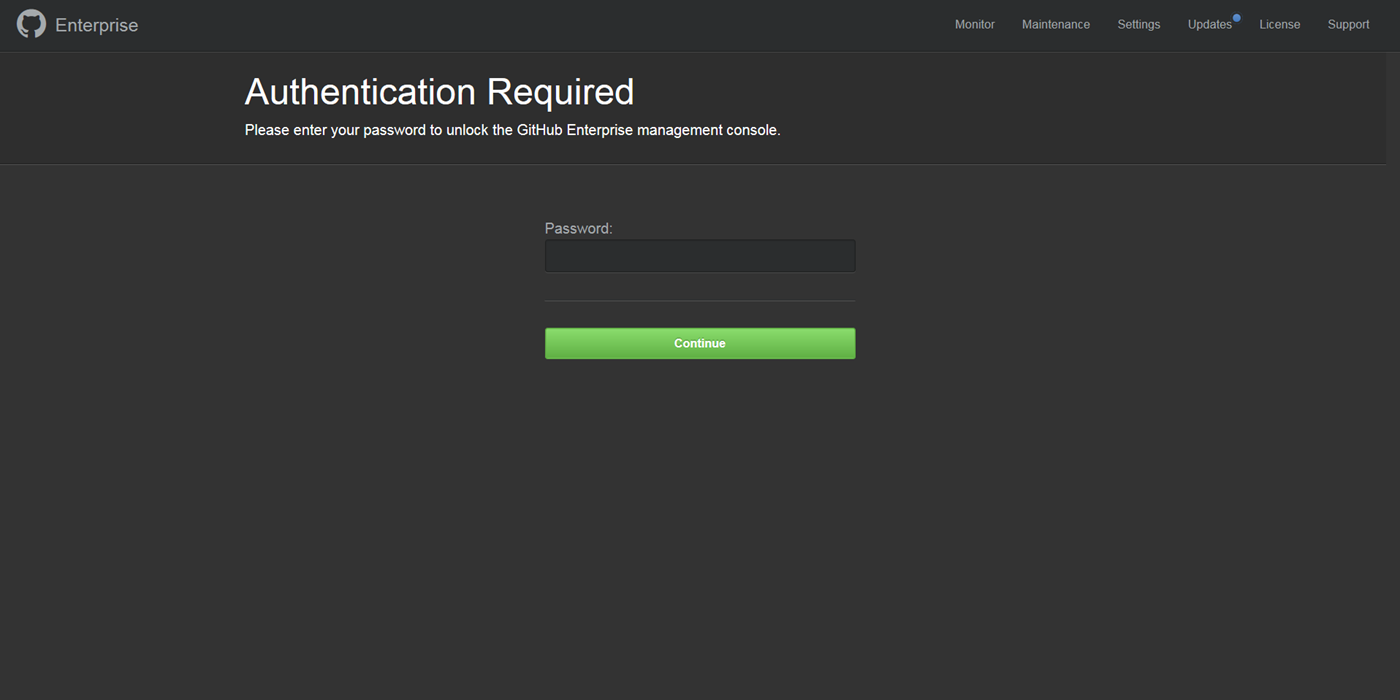
Administration section

* In this section where you can debug add and suspend users and repositories from here you can even upgrade the license can see how many sets are used and still remaining
* By using User search is an exact search where You can only search by user ID not first or last name



Management Console:

The Management Console helps you manage your GitHub where you can install certificates license can monitor, adding keys, complete admin access and handles all of the appliance-level settings to nm github. Where it prompts with authentication required to log into you need to have the right access permission then you can access it



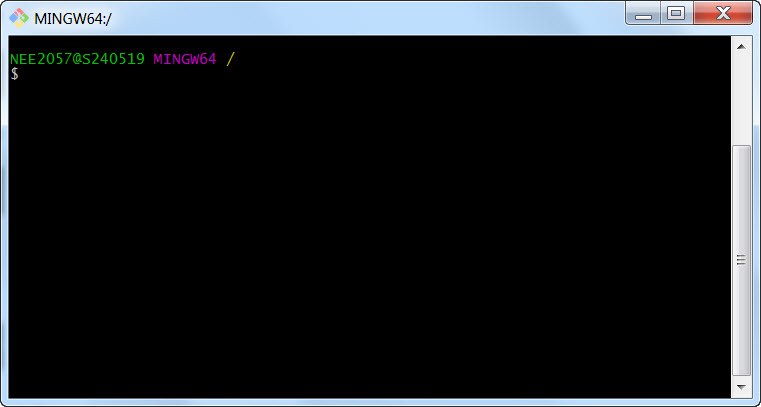
The Management Console will be locked after 10 failed login attempts in the span of 10 minutes. Once locked, the login screen will automatically unlock when fewer than 10 failed login attempts are made during the previous 10 minute period. Once a successful login occurs, the counter is reset.

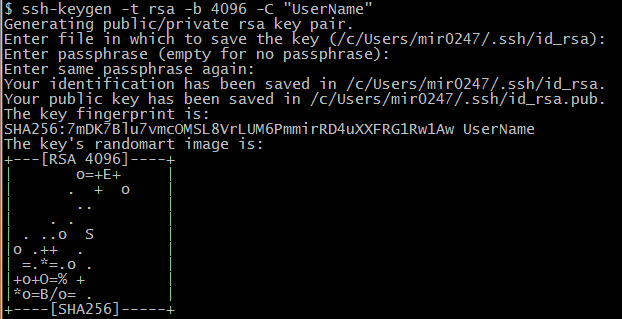
Here you can change the password for the Management Console or else you can even add your SSH administrative access uses authorized SSH keys you've added instead of this password.

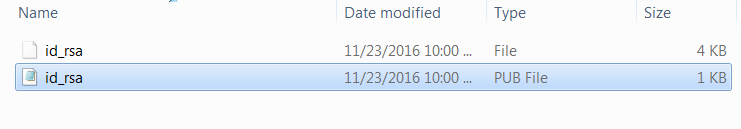
SSH access:-

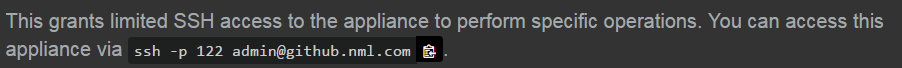
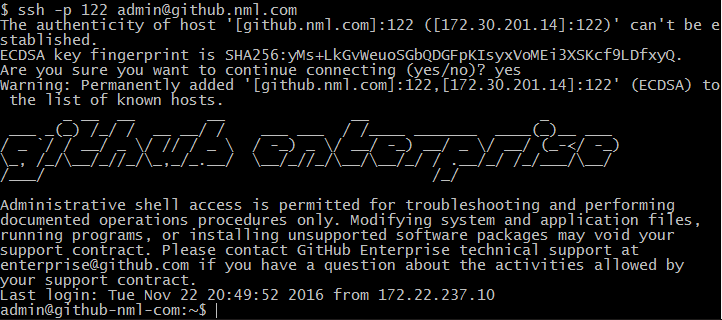
Generating Ssh key to access the ssh access to GitHub. Below are the following steps

1. Open Git Bash terminal.

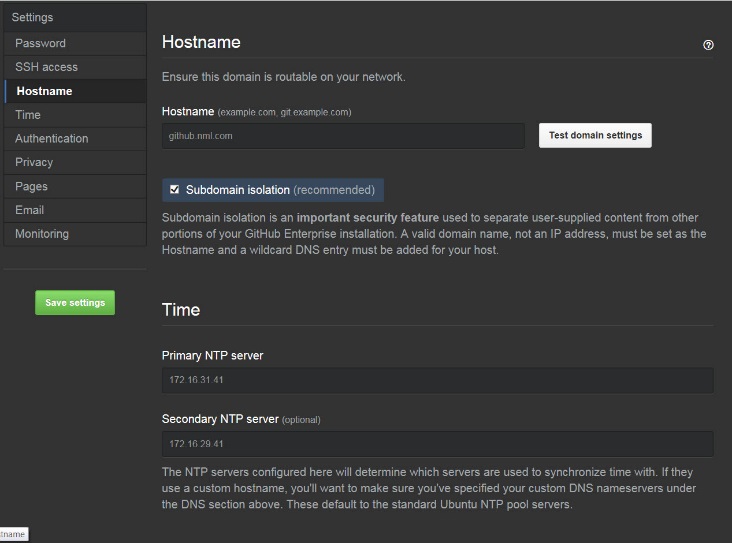


1. Type the below command:
   1. Syntax: ssh-keygen -t rsa –b 40969 –c “User Name”
   2. User Name can be replaced with your name, email, LanID, etc.
   3. Click enter to accept the default location
   4. Click enter twice to ignore adding a pass phrase
   5. The key is now created
2. Git Bash generates a public/private rsa key pair with the following names: id \_rsa and id\_rsa.pub.
   1. id\_rsa file is the private key
   2. id\_rsa PUB file is the public key



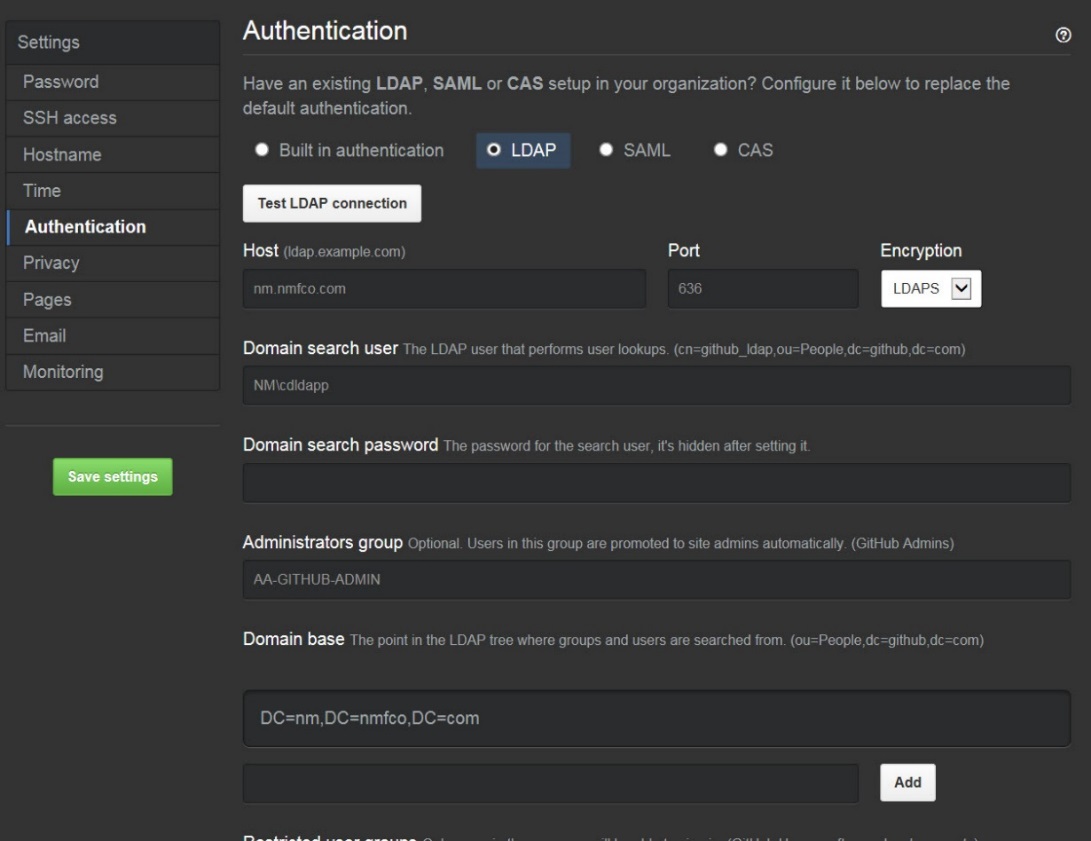
1. View the public key by your favorite editor and copy the key into the management console Ssh key store. 
2. You can now access the GitHub servers by copying the below command from the ssh section on management console
3. Paste the copied command into the Git Bash terminal. You will now be able to do administrative tasks on the servers. 

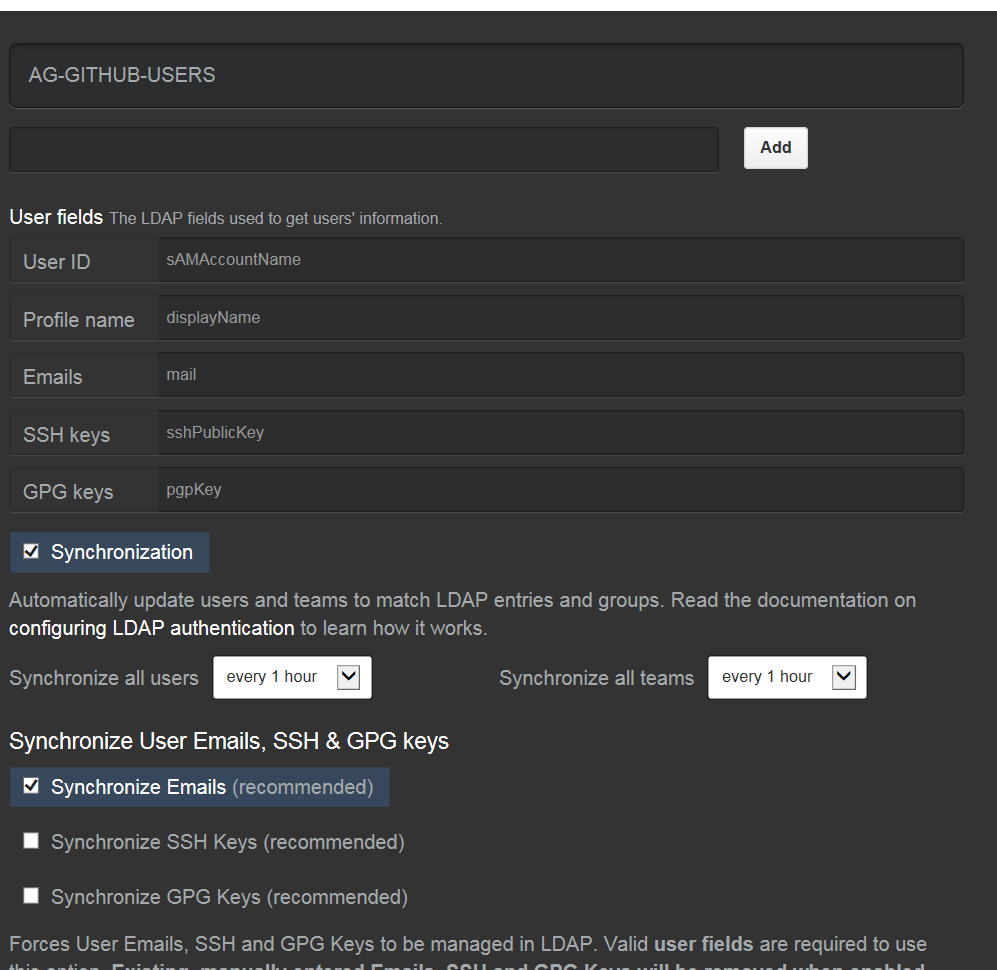
* Hostname
  + Hostname: github.nml.com
  + This DNS record points to instance one for GitHub
  + Subdomain isolation: checked had to get a new SSL cert with wildcards in it)
    - Helps mitigate cross-site scripting attacks
    - When GitHub does releases, if they add a namespace that is the same as a username there would be issues if subdomain isolation is not turned on
    - Pre-requisite to turn it on with SSL is a wildcard SSL cert
    - Having this feature turned on frees up the namespace for GitHub to upload new features
    - The Management Console is not affected by subdomain isolation
* Time Server
  + There are no NMTEST NTP Servers
  + 172.16.29.41 – primary where the github is running on it



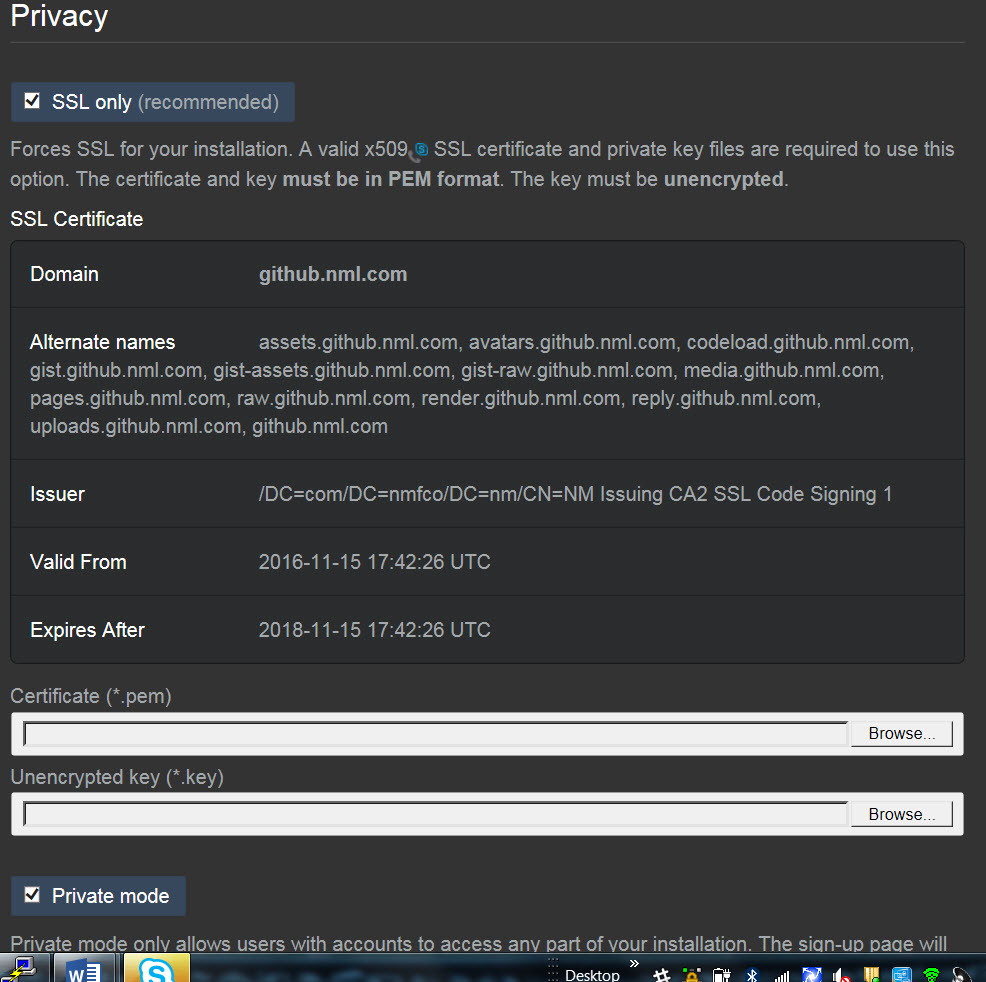
Authentication

* LDAP is running as LDAPS. Where the Host: nm.nmfco.com which runs on port 636 on domain user NM\cdldapp and the password for domain search on which the Administrators Group is AA-GITHUB-ADMIN Group to determine if a user should be given admin rights once they login. The point in which Domain base DC=nm, DC=nmfco, DC=com
  + Restricted user groups only the users in these groups will be able to sign in AA-GITHUB-ADMIN and AG-GITHUB-USERS
  + User Fields to get user information such as name and email
    - User ID: sAMAccountName
    - Profile Name: display Name
    - Emails: mail
    - SSH Keys: sshPublicKey
    - GPG Keys: gpg Key
  + LDAP Sync Interval check the "Synchronization box" .Uses for LDAP server to govern teams, email, SSH keys and GPG singing keys from your LDAP server and allows all users and teams to synchronize for every hour,4 hours or daily
    - Will overwrite any manual users added to GitHub





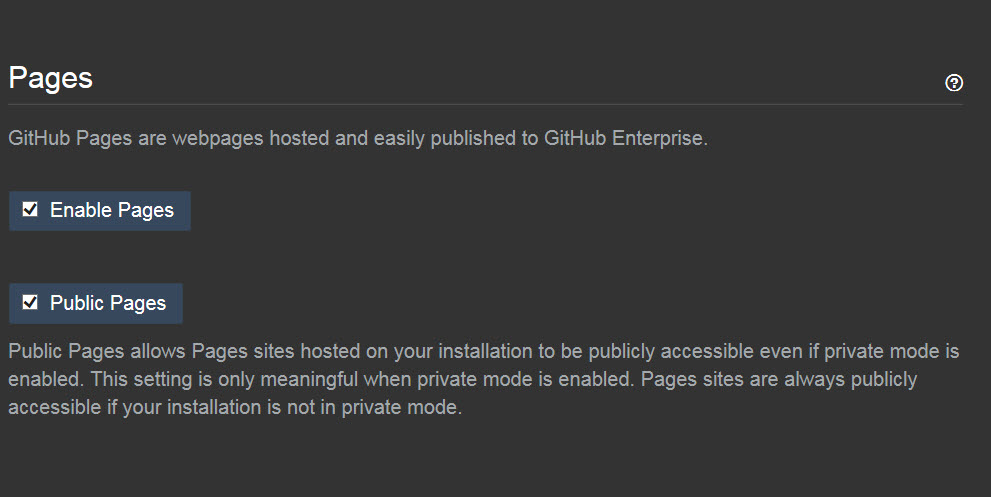
* Privacy
  + Check SSL checkbox only where you can add the certificate and key which must be in .pem or else you can choose file by browsing
  + SSL Cert
    - Certificate: github.nml.com.pem
    - Unencrypted Key: github.nml.com.key
  + Private Mode checkbox is checked
    - Private mode doesn't allow you to see public repos if you're not authenticated via LDAP. If you turn off private mode people would be able to view repos when they are unauthenticated, so anyone in the company can view them. They would be able to clone the repo but they wouldn't be able to push to a repo without authentication
    - If you make a public repo in private mode, anyone who signs in can see that repository and read it (not push to it)
    - Regardless of whether private mode is checked, teams can still create a private repo that only they have access to.
    - Example: if a public repo for Widgets is created with private mode, anyone in NM can see it. If the Widget repo is public but private mode is enabled, they need to log in to GitHub first to see it.
  + HTTP Proxy Server: leave empty for now which is used when we outbound internet connection for the appliance
  + HTTP Proxy Exclusion: leave empty for now
  + Enable GeoJSON rendering: leave unchecked it is used by the third-party services to provide base maps



* Pages
  + Enable Pages: checked where you can upload repository that has HTML in it and these pages are rendered under that sub-domain

For example: <http://target.github.io/>

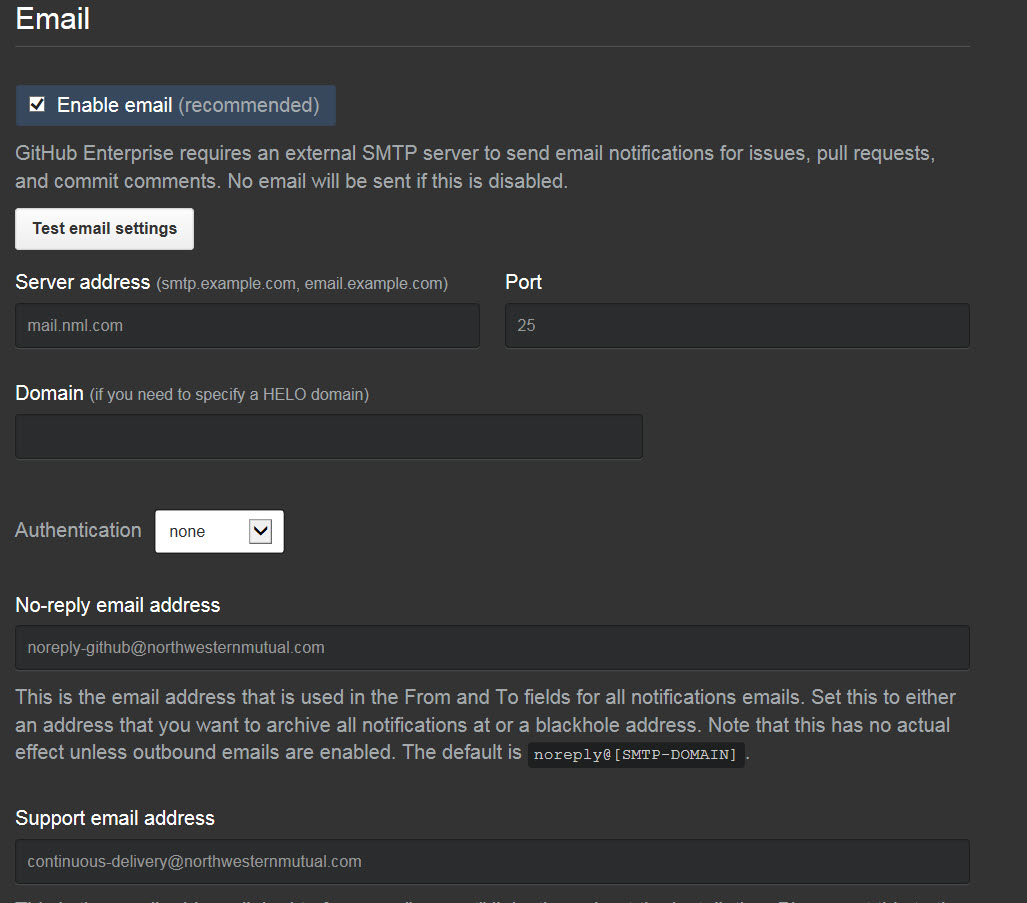
* + Public Pages: checked in which anyone in the company can see them without being authenticated with LDAP group



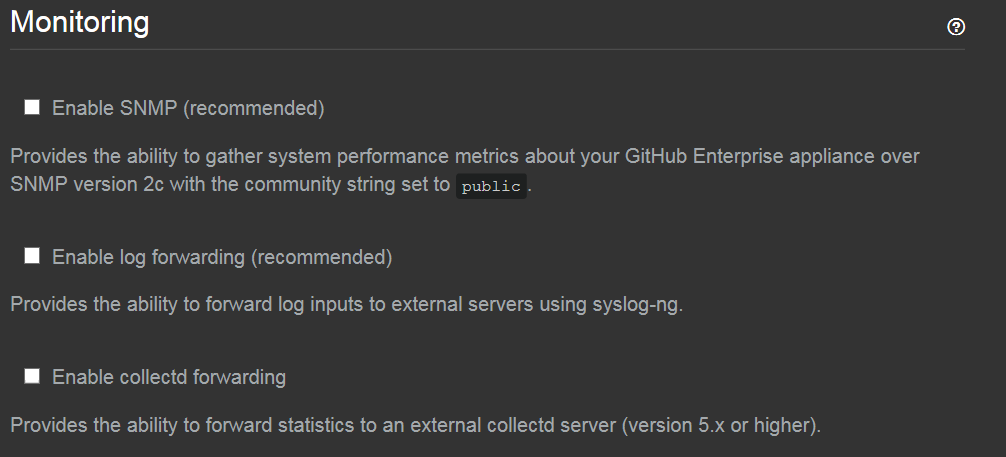
* Email
  + Enable email: checked
    - If GitHub is down, it will automatically email the entered email address .This email address will also show in any maintenance/error messages displayed by GitHub server address: mail.nml.com on port 25.
  + Domain: leave blank
  + This email is used for notification of mails No reply email:

[noreply-github@northwesternmutual.com](mailto:noreply-github@northwesternmutual.com)

* + Support email address: [continuous-delivery@northwesternmutual.com](mailto:continuous-delivery@northwesternmutual.com)



* Monitoring
  + Enable SNMP: leave unchecked
    - We have an internal SNMP client at NM where this is what GitHub uses to alert you that it is experiencing issues and provide a community string
  + Enable log forwarding: leave unchecked
    - Need to provide a server which will send over audit logs and application logs to specify URL and specifically sends out information to that server/URL
  + Enable collected forwarding: leave unchecked



* When you click on "Save Settings" GitHub will go down so we will need to do all configuration updates during off-hours
  + This will keep the connection alive while it is restarts (soft restart), so users will see some latency
  + If one of the steps fails you might get an error
    - Potential troubleshooting steps could be: try restarting the appliance via command line
    - The new processes get started up first and leave the existing processes run so if things fail then the old processes would just continue to run and would not be shut off
  + This is place where you can find Logs or errors on the appliance at /var/log/github
  + After Save Settings is clicked the Certificate and Unencrypted key will be blank because the "SSL Certificate" section will now be filled in

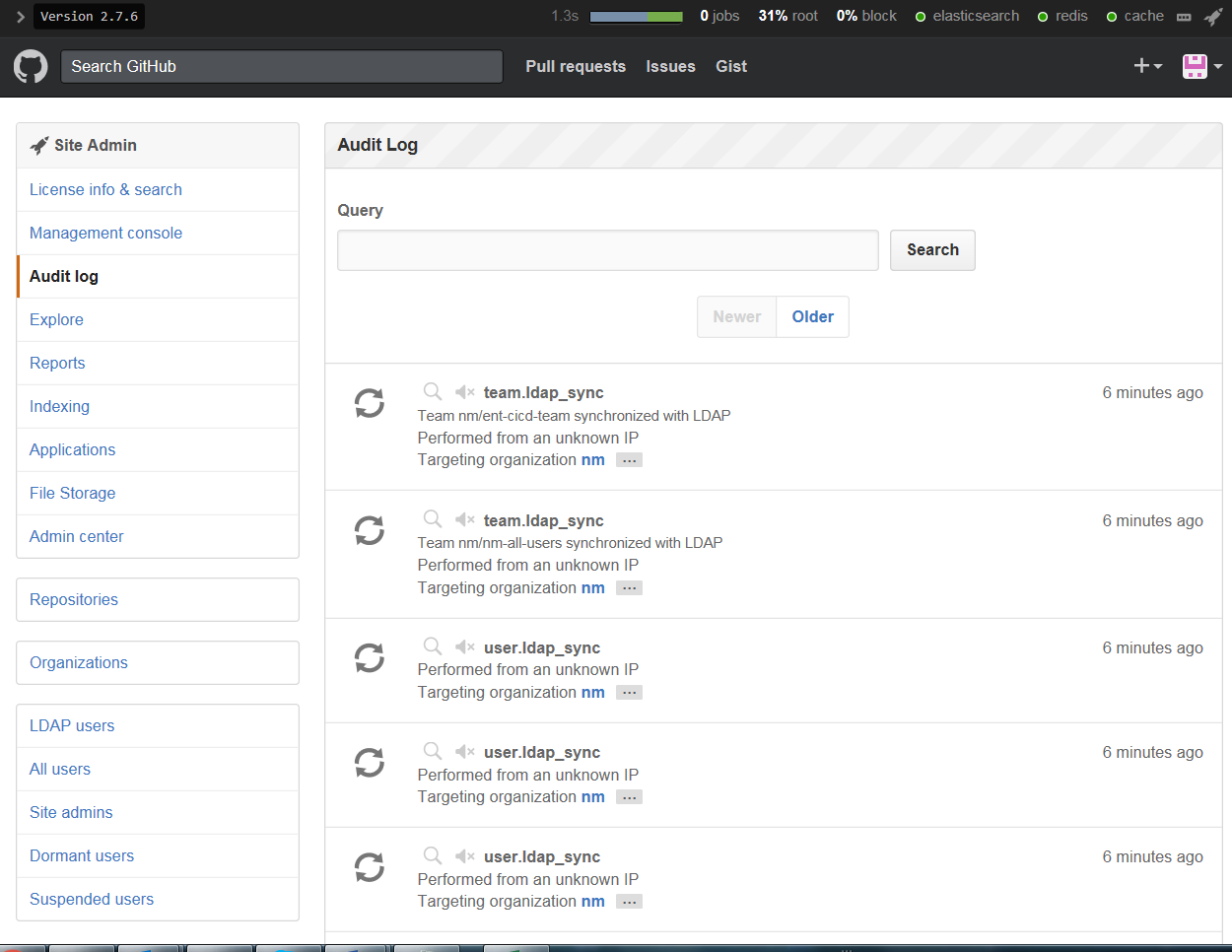
* Monitor
  + Shows multiple graphs to show appliance performance of CPU, memory, load, process, files, forks. Some of this data is accessible via SNMP but not via the API



* Maintenance
  + Enable and Schedule the time to announce when the server is in maintenance.
    - If you check the "Enable maintenance mode" it will turn it on right away if the dropdown says "Now" then after when you set it to a future time/date then it will display a message at the top of the page saying "GitHub will be down for maintenance on [day and time]"
    - This is basically used if you are going to do any upgrade it will show a maintenance page and stops new jobs from starting or updating in it.
* Updates
  + Can set it up to automatically download the upgrade package but it does not apply the upgrade until and unless we do it, You can only upgrade up to 2 minor release versions. When the upgrade is done, root partition is actually split into 2 partitions
  + Anyone with the SSH key can do the upgrade happens through the command line, cannot upgrade using the UI
* License
  + Can upload a new license file and can see the version of GitHub you have installed where it has limited to 150 seats = 150 licenses
    - When you log in, you automatically consume a license/seat can remove dormant users from having a license/seat
  + and their seat/license will be removed and returned to the pool
  + If you max out the license count and another person is added to the AD Group they will not be able to log in to GitHub
  + Can download a new license file from GitHub: Admin -> License file and info

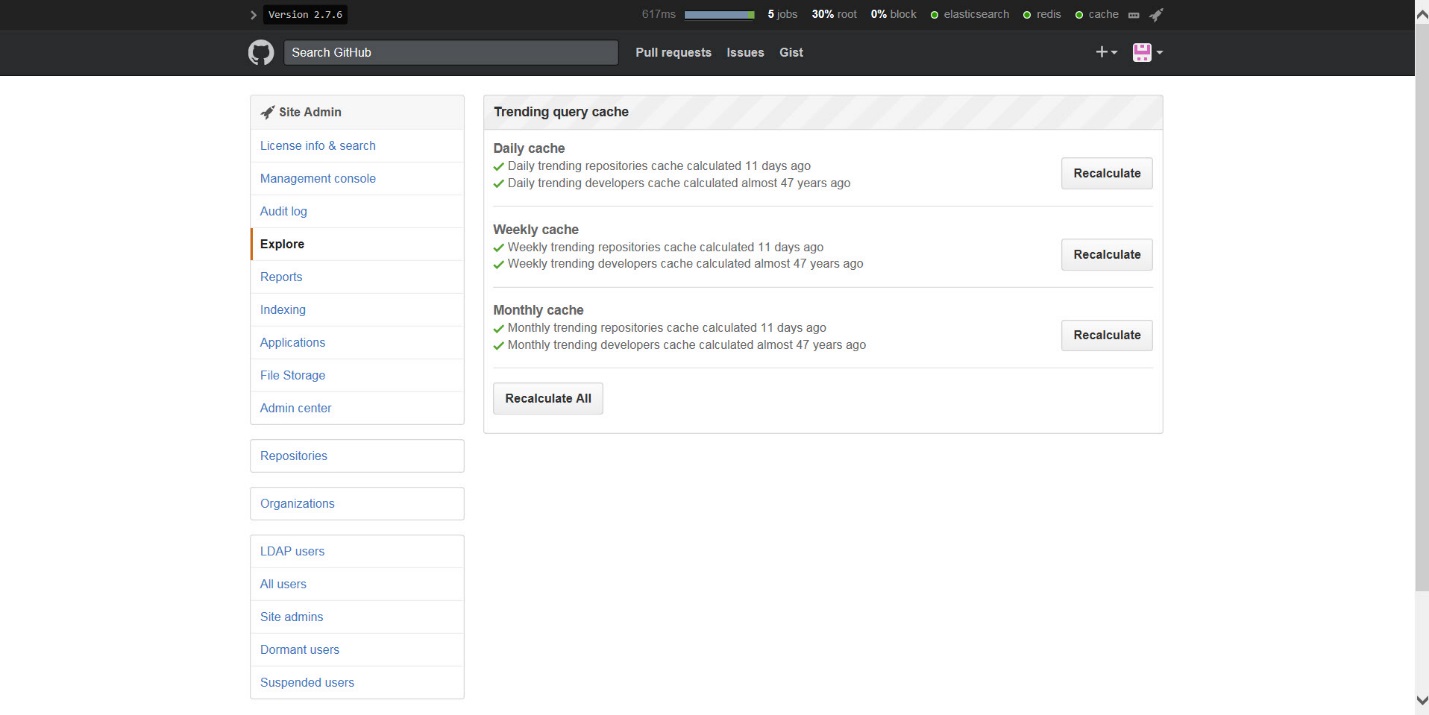
**Audit Log**

* + It show events of what happened on the appliance an can filter events by magnifying glass and mute button
  + By clicking on the ellipses "…" to see more detail about the event



* Explore
  + It contains the trending query caching for the explore page for each user on

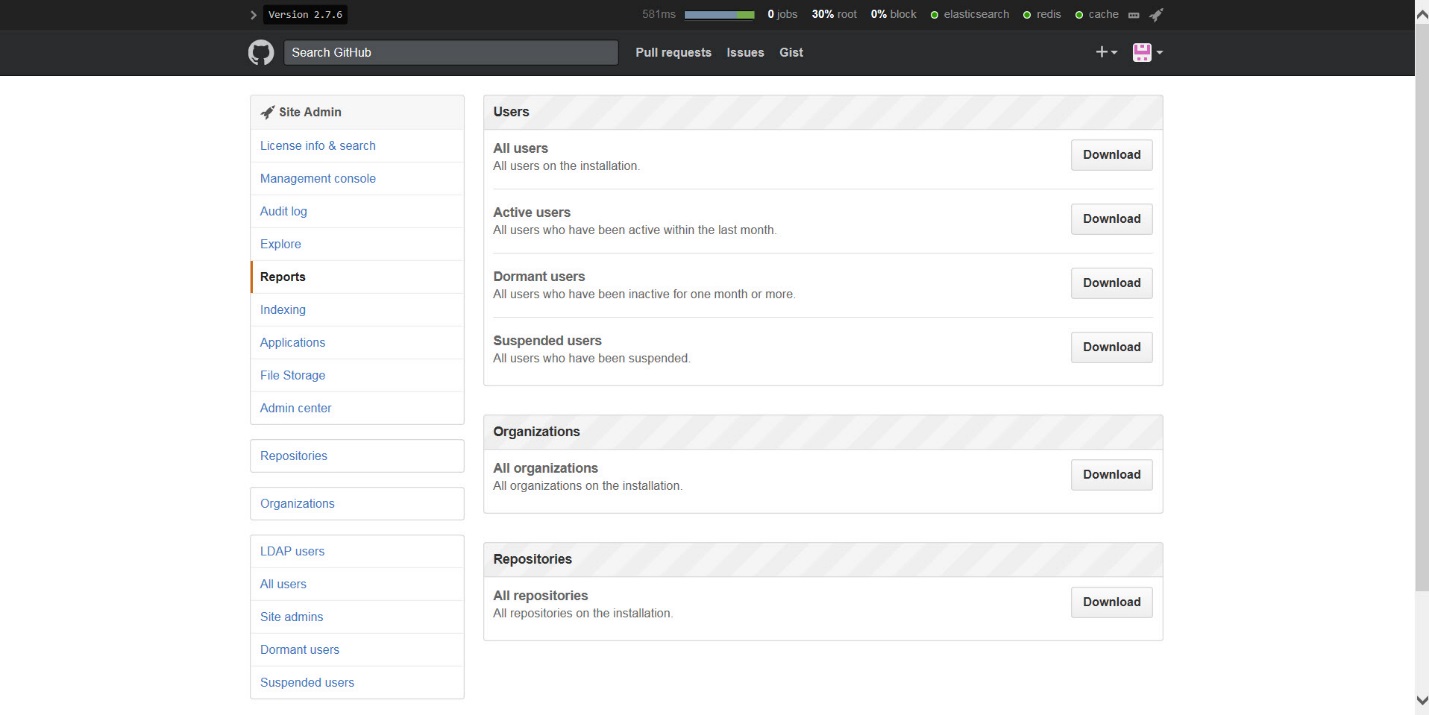
Daily, weekly cache or monthly cache and the Caching jobs are returned run on their own



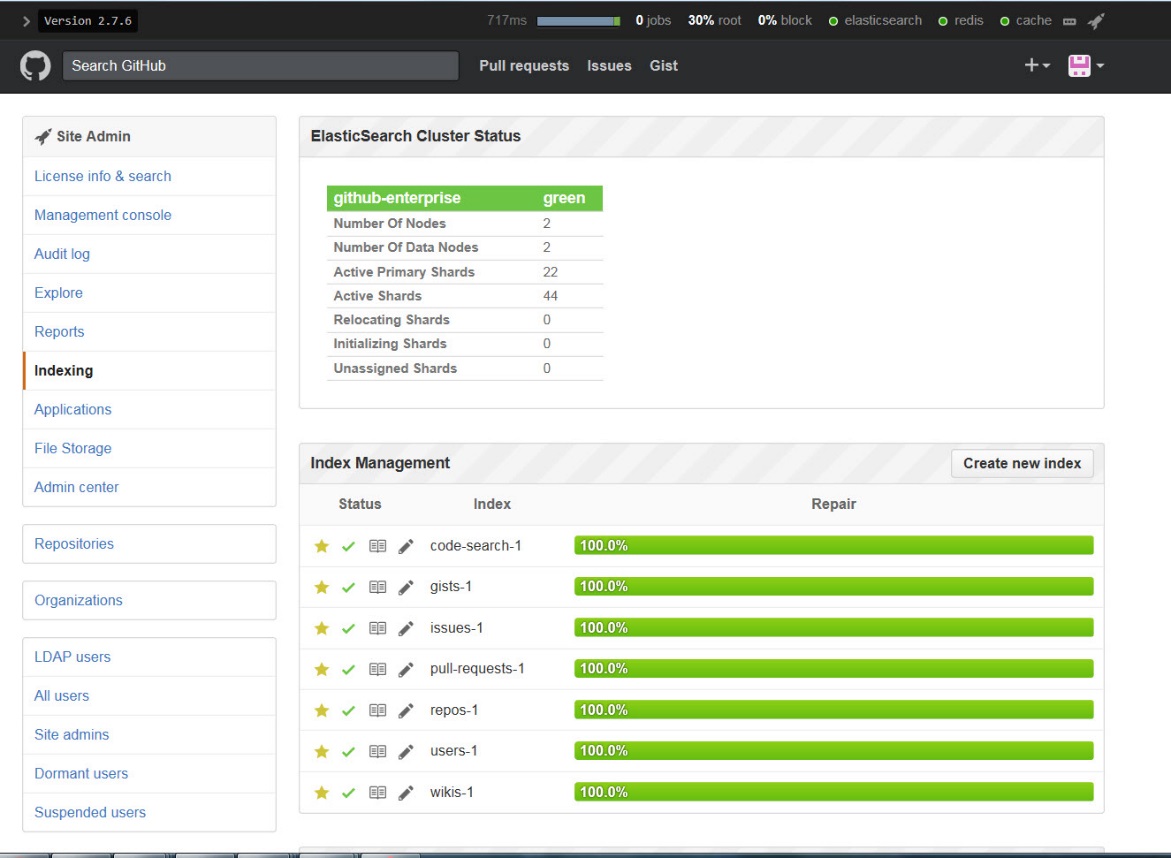
* Reports

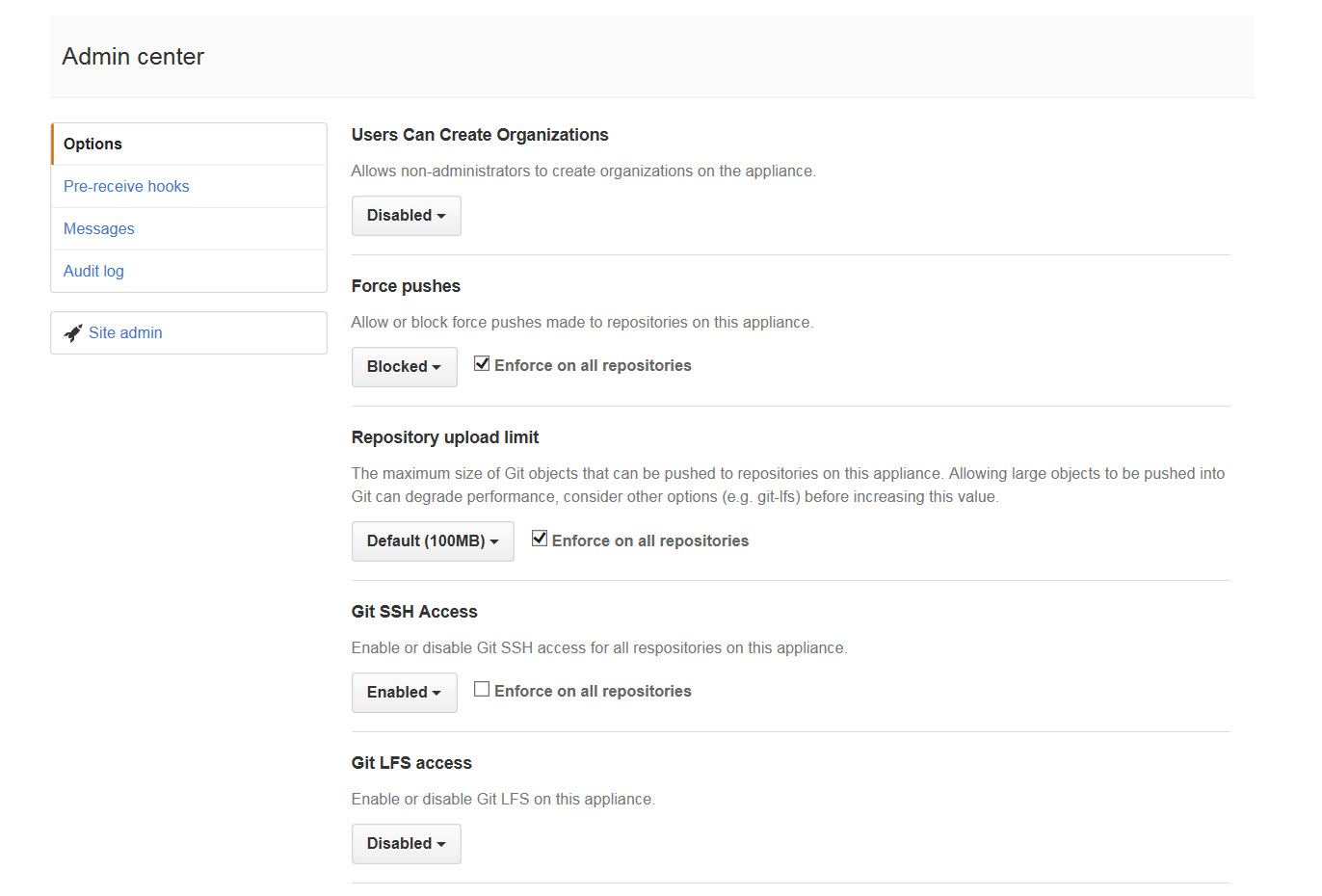
Which shows all types of users like active, dormant users and suspended users. This is where you can find users

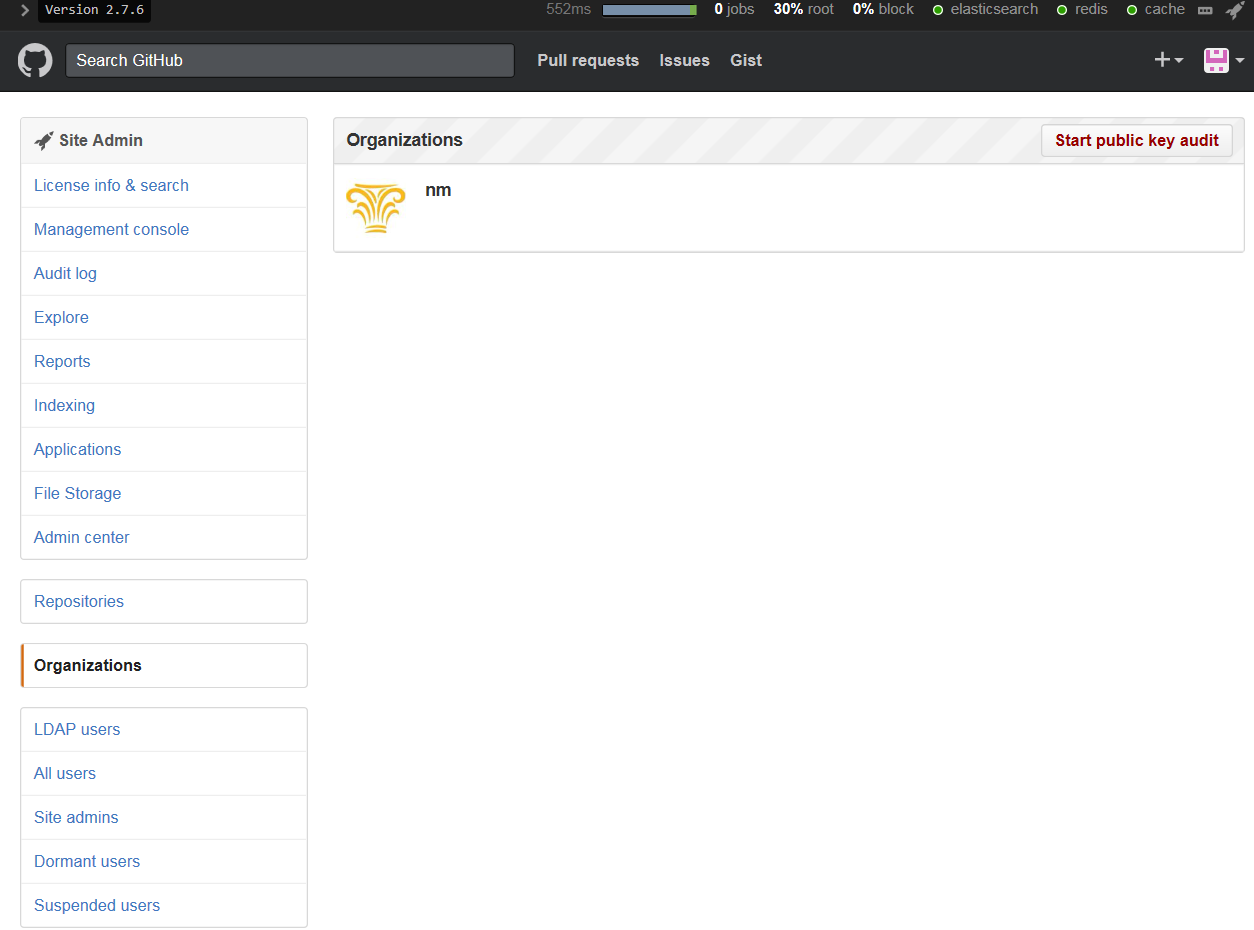
* Active users: All users who have been active within the last month.
* Dormant users is configured to any user who has not logged into GitHub in the last 30 days or more
  + To suspend a user you would remove them from the LDAP AD Groups



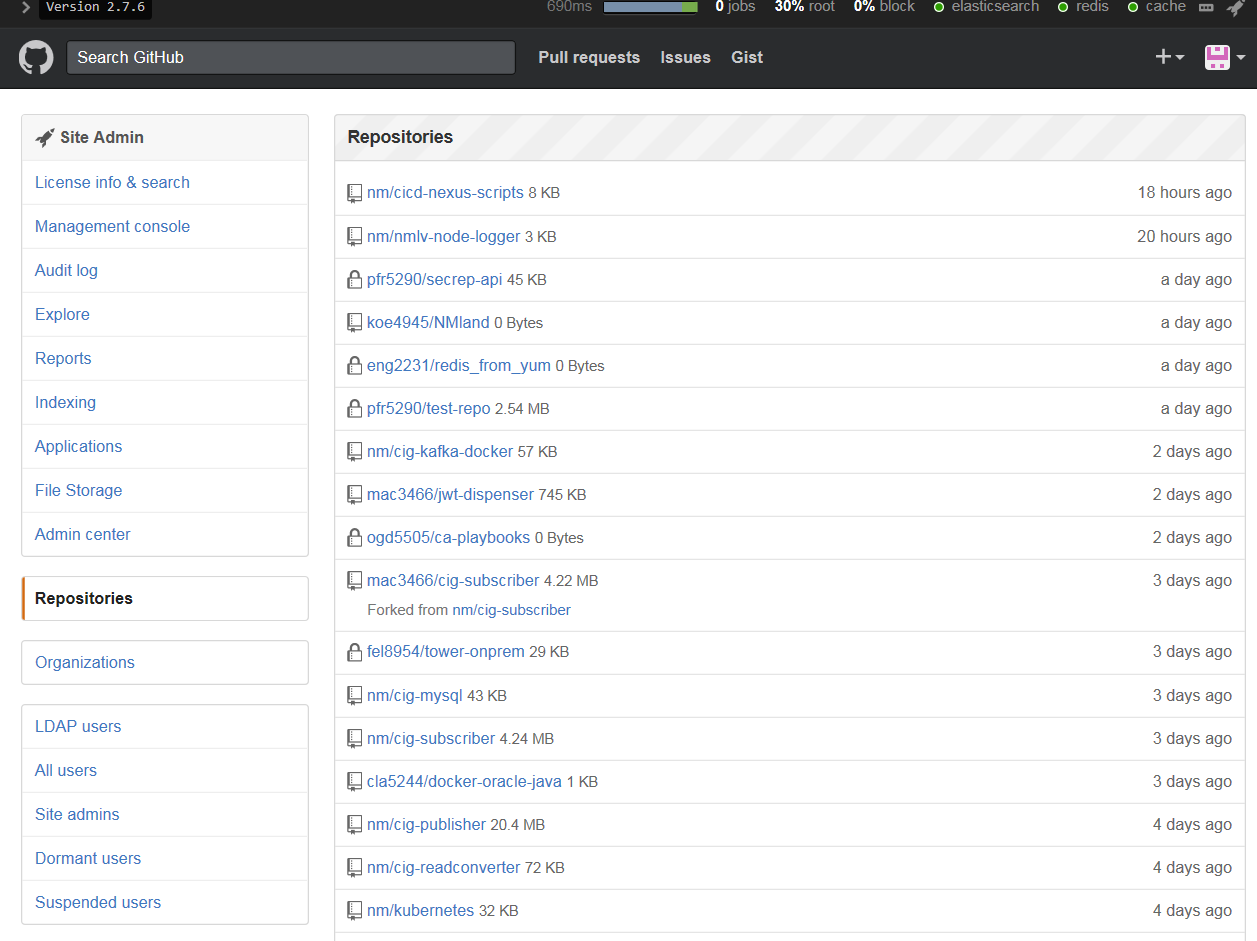
* Indexing
  + Our Elastic search health of cluster and index management where we can create a new index this Could be helpful to diagnose search issues



* File Storage
  + Show Information storage about host and its disk used and the available free space LFS
  + Git Large file storage will keep a text pointer file in your repo that points to a remote object in a file server, so when you clone down the repo, Git core automatically gets those pointer files and download the objects into your repo and more over LFS is a git feature, not a GitHub feature
* Admin Center
  + It has Toggle for allowing users to create organizations and force pushes when it is enabled users to allow force a commit and re-write history in GitHub
    - Enforce on all repositories means this is the rule on the instance, no site admins can change it per repo
  + Repository Upload Limit is on individual files. Allowing large objects to push into the git. This does not apply to Git LFS
  + LDAP Debugging
    - Will send out more verbose bugging in the logs if we are diagnosing an LDAP issue on the appliance
  + Pre-receive hook
    - Used to apply trigger that can be used to run scripts on every checking. Create the hooks at the appliance level, and can set them at the repo level which can point to a Docker image to run
  + Messages
    - Suspended user page and login page which prompts message to all the user are added to nm GitHub
  + Audit Log
  + Shows the audit log from where he deleted the keys, verified push or any actives that are done in github as a admin 
* Organizations
  + Type "\" to get a quick search to show up
  + Start public key audit - will send out an email to all the users and make them verify their public keys in SSH, will temporarily suspend all public keys



* Repository Admin Page
  + Public Push is blocked by default because we have Private Mode turned on
    - If you enable this then for this repo anyone can logged into GitHub could push to this repository. You can grant yourself access to a private repository as an admin on the Security tab
  + When you fork a repo, you get a copy that is by default in the same network
    - Forking is taking a copy of a repository that you might not have write/push access, you make your changes to your copy and then submit them back to the original repository. Forking allows a separation of write access



**USE OF SSH KEY**

* SSH keys are a way to identify trusted computers without involving passwords. You can generate an SSH key and add the public key to your GitHub account by following the procedures outlined in this section.
* Need public and private keys on your local machine to get the instance of GitHub on your local machine to admin into the box
* If we all use the same key then we can't tell who is doing what We should have an SSH key for each individual one so that we can come to knowso, where we enter in the public key to get access

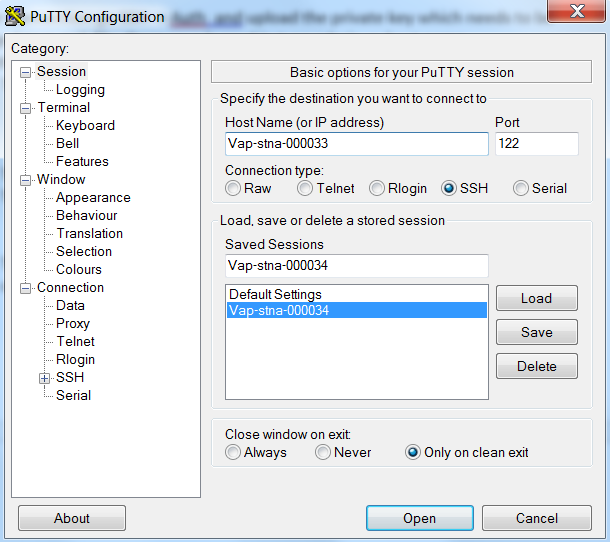
Install/Configure GitHub

We have 2 servers:

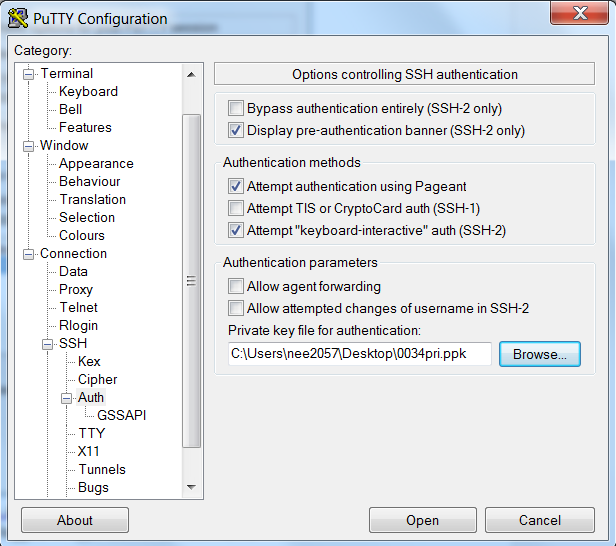
Github-1 IP Address: 172.30.201.14(Vap-stna-000034

Github-2 IP Address: 172.30.201.35(Vap-stna-000033

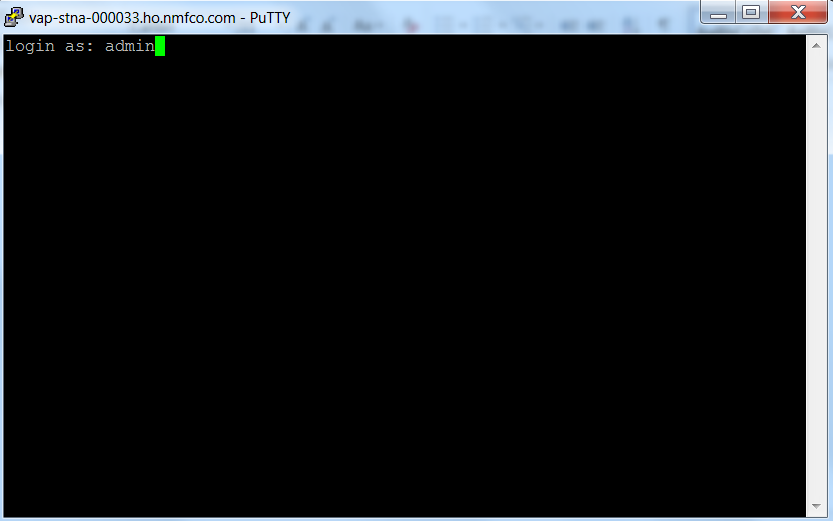
* + Where github-1.nml.com and github-2.nml.com where github-1 is primary and which should be pointed at <http://github.nml.com>
  + github.nml.com is the main link but it is not a true load balancer because it is hardcoded to point to github-1 where changes cannot be made in it
* If there are any issue or dropdown we would need to point it change to point to github-1 to maintain continuity so that there will not be any loss of data. For the connection to the server follow the below steps.
  + Putty into using the IP address 172.30.201.14 or by server name Vap-stna-000034 by using port 122



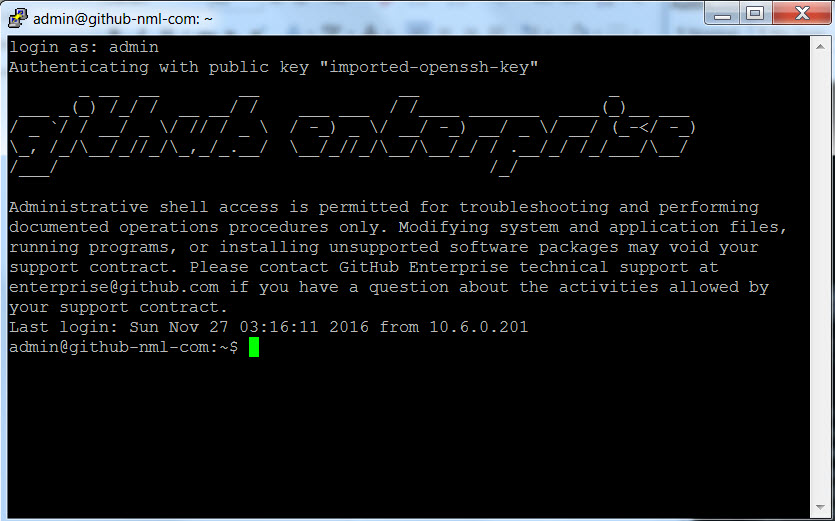
* + - Under Connection go SSH**🡪** Auth­­ and upload the private key which needs to be changed to a .ppk file. We can generate the .ppk by using the **puttygen** by uploading private and save it as private key with extension .ppk ,so that the server can recognize private key to give access to you



Once you’re logged into the server then you can see the command prompt with login where you can type admin and press enter

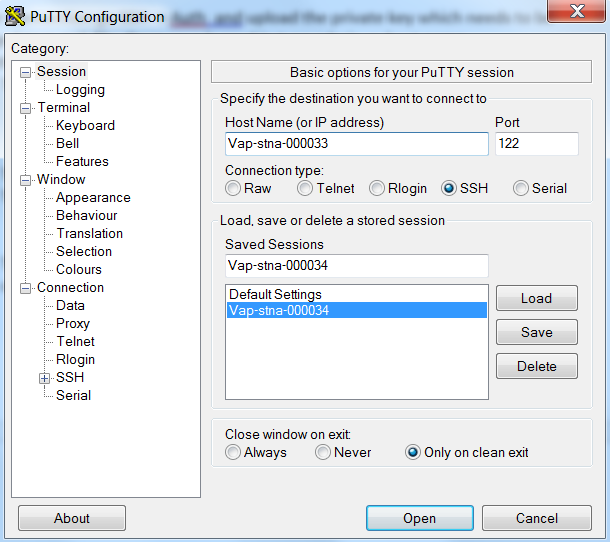


Then you can find the command prompt as shown below.

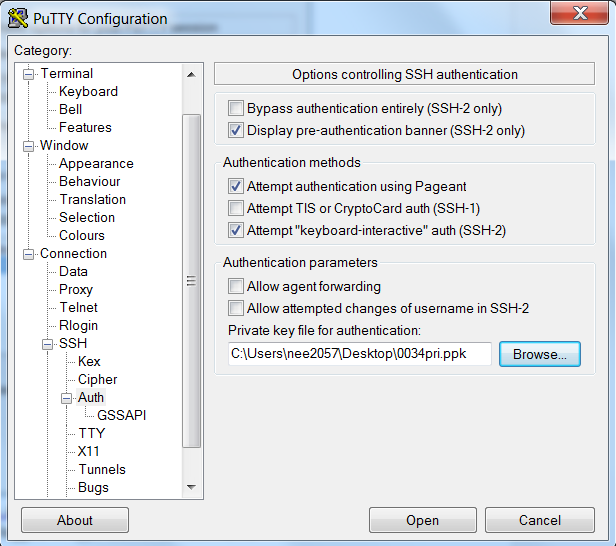


Configuring the Replica (second instance)

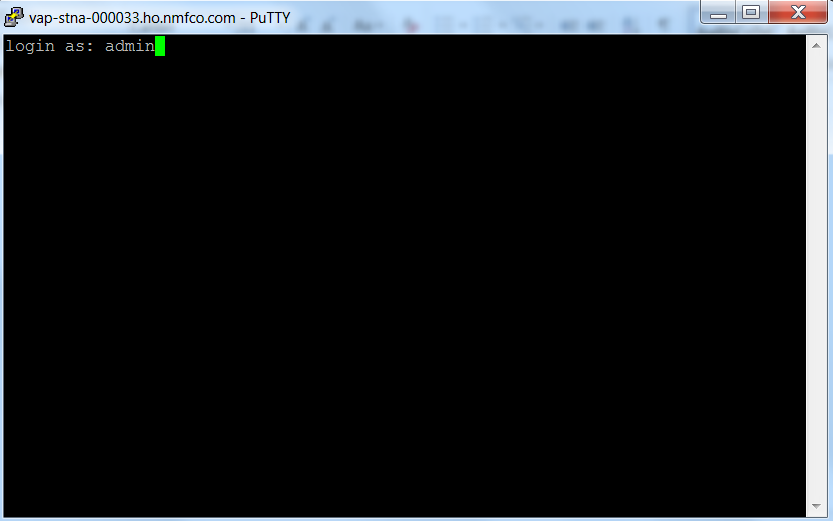
* When we configure Github-2 as the second instance it will pull data from github-1 to keep asynchronous and replicate itself
* When Github-1 operates as if Github-2 is constantly pulling from and working as the replica of github-1. It does a data stream from the primary instance to the replica. There isn't a storage, Ram or CPU between github-1 and github-2. Everything gets replicate from github-1 as the source code, repos etc.
* To setup the replica
  + Putty into using the IP address 172.30.201.35 or by server name Vap-stna-000033 by using port 122



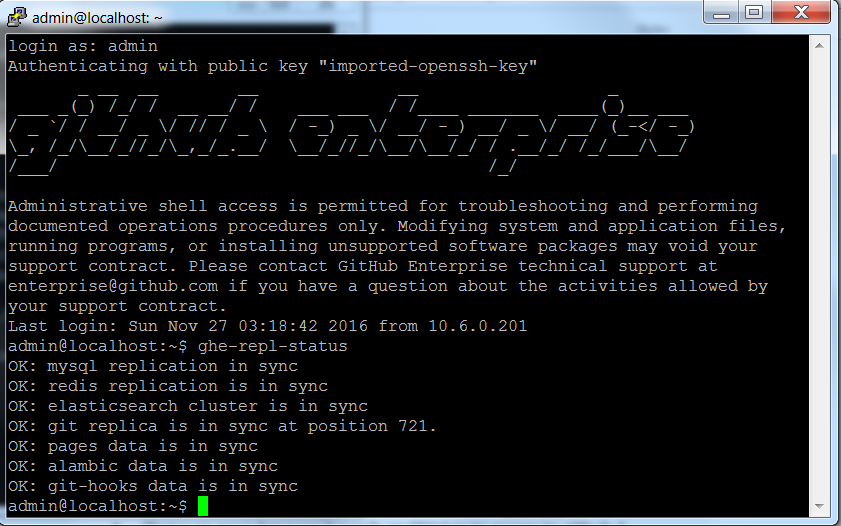
* + - Under Connection go SSH**🡪** Auth­­ and upload the private key which needs to be changed to a .ppk file. We can generate the .ppk by using the puttygen by uploading private and save it as private key with extension .ppk ,so that the server needs the private key to give access to you



Once you’re logged into the server then you can see the command prompt with login where you can type admin and press enter



* + - Type the following command to set up the replicate  **ghe-repl-setup** 172.30.201.14



* + - * + By performing this we can run and configured github-2 to recognize github-1 as the primary

By doing the following commands we can setup the replication to server:-

* + - * Typing **ghe-repl-start** command is used to turn on active replication of all data stores , where don't need to give the IP address because its already configured to use github-1
      * Type this command **ghe-repl-status** to get the status of each data store's replication channel.
      * If you type **ghe-repl-stop** it will disable the replication process for all data stores, either temporarily or permanently. This command only stops the replication services.
        + **Ghe-repl-tear** down will clean everything out and will allow you to start over on the replica. If replications fails we should contact Enterprise Support

**GitHub Backup Utility**

Backup utilities are a disaster recovery. This takes the snapshots of all important data storage. These snapshots are used to restore back when instance is in prior state or by setting up a new instance without having another always-on GitHub Enterprise instance

Use of scheduling corn jobs:

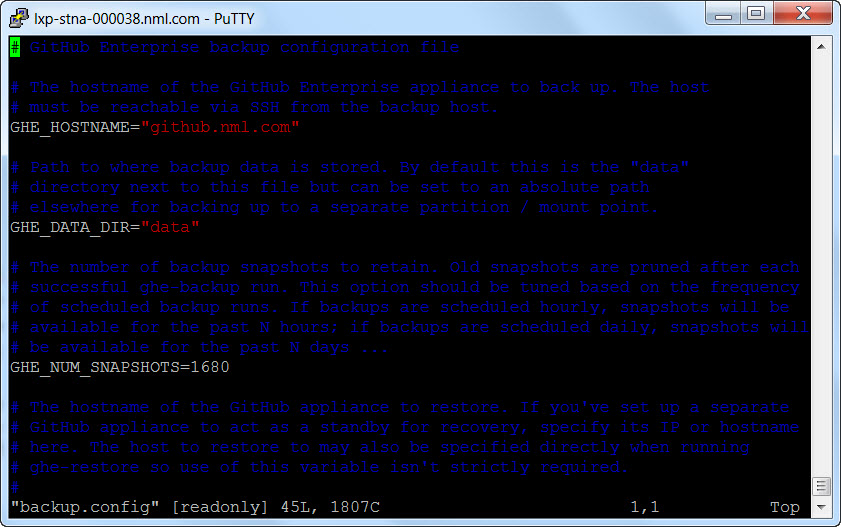
Generally the scheduling is done with the corn job where we set the minute, hour, day, month, day-of-week it might be any depending on time or day the server starts taking the backup and snapshots of the files. We can do it through the command line by running corntab –e within your favorite editor.

* Login with putty to lxp-stna-000038 by using the LANID to the backup server
* This is the location where you can make changes in backup.config which can be used to give configurations to setup **/opt/github/data/backup-utils/backup.config**
  + You can set up the following

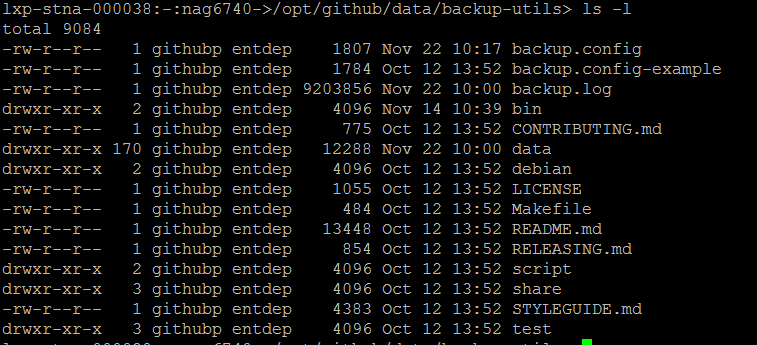
GHE\_HOSTNAME: [“github.nml.com](http://www.github.nml.com)”

GHE\_DATA\_DIR: “data”

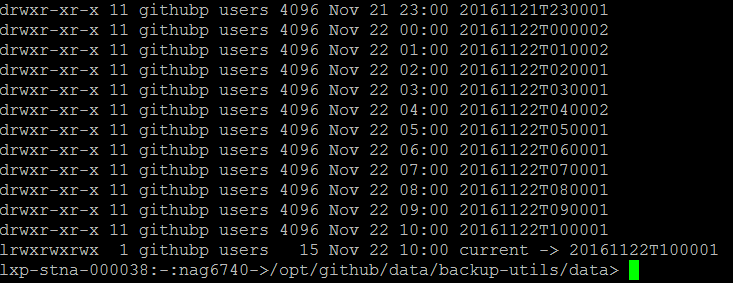
GHE\_NUM\_SNAPSHOTS (1680) values in the backup.config



* But here we have scheduled for evert 1 hour and This is path where you can store the backups $**cd opt/github/data/backup-utils/**



This is how it looks when you take backup snapshots which are stored in rotating increment directories named after the date and time the snapshot was taken. Each snapshot directory contains a full backup of all relevant data stores, repository, Search, and Pages data is stored



Where you can view the files in the backup by going into **opt/github/data/backup-utils/data/Id’s** of the file. By going into the files even you can see the content of that file.

