GitHub IP Addresses

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

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

Install/Configure GitHub

* Go to <http://github-1.nml.com>
  + Should be pointed at Github-1
  + We have 2 servers: github-1.nml.com and github-2.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
    - If there is an issue we would need to change it to point to github-2.nml.com
* Click "Continue to Setup"
* Upload the license file
* Password: GitHub12345
  + Can reset it from command line on the server if you forget the password
* Click New Install
* Fill in the rest of the Configuration Settings

Management Console - Configuration Settings

* Password: Sanz43!247
* Generate SSH Key
  + 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
  + Regardless of which SSH key is used, you will still get logged in as the admin. So the box won't distinguish who is actually on the server, it will just always be the admin
  + Enter in the public key
* Hostname
  + Hostname: github.nml.com
  + DNS record points to instance 1
  + DNS for GitHub
    - DNS for GitHub points to github-1
  + 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
  + 172.16.31.41 - primary
  + 172.16.29.41 - secondary
* Authentication
  + LDAP
    - This is not LDAP Secure, meaning the backend connection from GitHub to LDAP is in plain text
    - We have now changed this dropdown to LDAPS
  + Host: nm.nmfco.com
  + Port: 389
    - This port is not secure so we have now changed it to be Port 636 to enable LDAPS
  + Domain search user: NM\CDLDAPP
  + Domain search password: D0g0mon7
  + Administrators Group: AA-GITHUB-ADMIN
    - Group to determine if a user should be given admin rights once they log in
  + Domain base: DC=nm,DC=nmfco,DC=com
  + Restricted user groups: AA-GITHUB-ADMIN and AG-GITHUB-USERS
    - Groups to allow a user to sign in
  + User ID: sAMAccountName
  + Profile Name: displayName
  + Emails: mail
  + SSH Keys: sshPublicKey
  + GPG Keys: pgpKey
  + LDAP Sync
    - Check the "Synchronization box"
    - Uses LDAP server to govern teams, email, SSH keys and GPG singing keys from your LDAP server
    - Synchronize all users: every 1 hour
    - Synchronize all teams: every 1 hour
    - Will overwrite any manual users added to GitHub
* Privacy
  + Check SSL Only checkbox
  + 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
    - * Need outbound internet connection from appliance
  + HTTP Proxy Exclusion: leave empty for now
  + Enable GeoJSON rendering: leave unchecked
* Pages
  + Enable Pages: checked
  + You can upload repository that has HTML in it and these pages are rendered under that sub-domain
  + Example: <http://target.github.io/>
  + Could use this for all documentation for CD info
  + They are HTML rich pages
  + Public Pages: checked
    - Anyone in the company can see them without being authenticated with LDAP
* 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
  + Port: 25
  + Domain: leave blank
  + 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
    - This is what GitHub uses to alert you that it is experiencing issues
    - Need to provide a community string
  + Enable log forwarding: leave unchecked
    - Need to provide a server
    - Will send over audit logs and application logs
    - Specify URL and specifically sends out information to that server/URL
  + Enable collectd 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
  + Will keep the connection alive while it 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
  + Logs are 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
* Management Console Password for Github-Test: GitHub@2016

Management Console

* Monitor
  + Shows multiple graphs to show appliance performance
  + Some of this data is accessible via SNMP but not via the API
* Maintenance
  + Enable and Schedule
    - If you check the "Enable maintenance mode" it will turn it on right away if the dropdown says "Now"
      * If 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 used if you are doing an upgrade it will show a maintenance page
    - Stops new jobs from starting
* Updates
  + Can set it up to automatically download the upgrade package but it does not apply the upgrade
  + 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
  + Upgrade happens through the command line, cannot upgrade using the UI
* License
  + Can upload a new license file
  + Can see the version of GitHub you have installed
  + 150 seats = 150 licenses
    - When you log in, you automatically consume a license/seat
    - Can remove dormant users from having a license/seat
  + If a user is removed from an AD Group, because LDAP sync is enabled once they are removed from the AD Group they will be suspended from GitHub and their seat/license will be removed/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
* Support
  + Links to submit issues to GitHub Enterprise Support

Configuring the Replica (second instance)

* When we configure Github-2 as the second instance it will pull data from github-1 to sync/replicate itself
* Github-1 operates as if Github-2 is constantly pulling from github-1 and working as the replica of github-1
* It does a data stream from the primary instance to the replica
* There isn't a shared disc between github-1 and github-2
* Everything gets replicated: source code, repos, etc.
* High Availability Documentation: <https://git.io/vP2R7>
* To configure the replica
  + Navigate to github-2.nml.com
  + Upload the license file
  + Click New Install
    - Migrate is used for restoring a backup snapshot from our backup utility
  + Add in the SSH key
    - Open the .pub in a text editor, copy the entire contents and click "Add key"
* To setup the replica
  + Putty into the second instance
    - Host name: [admin@github-2.nml.com](mailto:admin@github-2.nml.com)
    - Under Connection -> SSH -> Auth upload the private key (.git needs to be changed to a ppk file type)
    - Type the following commands:
      * ghe-repl-setup 172.30.201.14
        + Is the githut-1 IP Address
      * Threw an error because it needs an SSH key to access to github-1
        + Enter in the new SSH key that was generated in the output, add the key to github-1 management console. You do not need to click "Save Settings", it will save the new key when you click "Add Key"
      * Type in the first command again: ghe-repl-setup 172.30.201.14
      * Type y
        + After this runs, we have configured github-2 to recognize github-1 as the primary but it has not copied any data over yet
      * Type ghe-repl-start (don't need to give the IP address because its already configured to use github-1)
      * Type ghe-repl-status to get the status of the replica
      * If you type ghe-repl-stop it will stop the replication process
        + ghe-repl-teardown 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

* Documentation: <http://git.io/vP2y0> ; <https://github.com/github/backup-utils#getting-started>
* SSH into lxp-stna-000038
* /opt/github/data/backup-utils/backup.config
  + You can set the GHE\_HOSTNAME, GHE\_DATA\_DIR, GHE\_NUM\_SNAPSHOTS (168) values here
* To generate new SSH key: Ssh-keygen -t rsa -b 4096 -C backup
  + Enter through the commands
* Copy the SSH keys into entdepp home directory: /home/entdepp/.ssh
  + Ended up copying the keys into /opt/github/keys
* Type cd /opt/github/data/backup-utils
* Type ./bin/ghe-host-check
  + Type y
* To run a backup ./bin/ghe-backup
  + Will create a snapshot of our github enterprise instance

GitHub Upgrade

* Stop replication: ghe-repl-stop
* Turn on maintenance mode on the primary
* Grab snapshot/backup of primary to ensure the current state is saved somewhere: ./bin/ghe-backup
* SSH into primary
* Since we have automatically downloading updates enabled you can go to the Management console, wait for all the processes to stop, click Updates, copy the command and run it
* Upgrade the replica independently and then start it again
  + Download the upgrade
  + Run the command to install the upgrade
  + Ghe-repl-start
  + Ghe-repl-status
* Navigate to the Management Console, verify all the settings are correct and click Save Settings
* Take the primary out of maintenance mode
* If you run into an issue with the upgrade (like the upgrade failed) the latest snapshot/backup will just have user data, it will not rollback the version. So to rollback you would need to:
  + Rollback the version: ghe-rollback-rootfs (this will just switch which root partition to boot from)
  + Restore the last snapshot/backup
  + At this point the replica has been turned off and we rolled back the upgrade so we are back to where we started. So SSH into the replica and just do ghe-repl-start
* If the system never reboots then call the VM team at NM

High Availability Failover

* Would need to submit DNS request first to flip it to point to github-2
  + To change the DNS locally you can navigate to C:\windows\system32\drivers\etc\host and edit that file to point to the other instance
* SSH to github-2
* Type ghe-repl-promote
  + This will automatically put the primary in maintenance mode
  + Promote automatically will stop replication
* Go to <http://github-2.nml.com> and verify that it no longer shows the "Server in replication mode" server
* SSH to github-1
* Type ghe-repl-setup 172.30.201.35 (IP Address of the primary you want to copy/target)
  + Type Y
* Type ghe-repl-start
* Logs: ghe-repl-status -vv **or** ghe-repl-status -vv|more
  + cd /data/user/git-repld/transaction.log
* To resolve mysql issue we did the following:
  + Ghe-repl-stop
  + Ghe-repl-start
  + Ghe-repl-status (was still broken)
  + Ghe-repl-teardown (still broken)
    - When doing a teardown you should do a ghe-repl-stop first because ghe-repl-teardown will not automatically stop replication
  + So this means the issue is with our primary instance. So now will try restoring the primary from a backup and then teardown the replica and reset it up to pull from the fixed primary
    - Put the primary into maintenance mode (via UI or via command line: ssh onto primary and type ghe-maintenance -s)
      * Can also type "ghe-maintenance -q" to see if maintenance mode is set
    - Log into backup server
    - Type Sesu entdepp
    - Type cd /opt/github/data/backup-utils
    - Type ./bin/ghe-restore 172.30.201.14 (IP Address of the current primary) (
      * This will automatically apply the last backup
      * To specify a specific backup type: ./bin/ghe-restore 172.30.201.14 -s [snapshot ID]
    - Go to the primary's settings and make sure everything looks good
    - Click the "Save Settings" button to kick off the configuration job
      * This will not automatically turn off maintenance mode
    - Turn off maintenance mode from UI or ssh into primary and type ghe-maitenance -u
    - We now verified that Jared's repo was back on the primary so we think the issue has been resolved because before his repo wasn't showing
    - SSH onto replica
    - Ghe-repl-stop
    - Ghe-repl-teardown
    - Ghe-repl-setup 172.30.201.14
    - Ghe-repl-start
    - Ghe-repl-status
    - All statuses were OK so issue has been resolved

Admin Site

* It is context-specific, so if you click the Rocket from a repo, you will get the admin page for that repo
* You can get to the home admin page by clicking the Rocket twice
* User search is an exact search - can search by userID, not first name or last name
* Audit Log
  + Show events of what happened on the appliance
  + Can filter events by magnifying glass and mute button
  + Click the ellipses "…" to see more detail about the event
* Explore
  + Contains the caching for the explore page for each user
  + Caching jobs run on their own
* Reports
  + This is where you can find dormant and suspended users
    - Dormant users is configured to any user who has not logged into GitHub in the last 30 days
  + To suspend a user you would remove them from the LDAP AD Groups
* Indexing
  + Our Elasticsearch health
  + Could be helpful to diagnose search issues
* Applications
  + All the 3rd party applications that have access to data on our appliance
* File Storage
  + Information about Git LFS
  + Git LFS 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
  + LFS is a Git feature, not a GitHub feature
* Admin Center
  + Toggle for allowing users to create organizations
  + Force pushes, if enabled, allows users to 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
    - 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
  + Pre-receive hook
    - Can be used to run scripts on every checkin
    - Create the hooks at the appliance level, and can set them at the repo level
    - Can point to a Docker image to run
  + Messages
    - Suspended user page and login page
  + Audit Log
    - Shows the audit log from a map view if GeoJSON is turned on
* 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 logged into GitHub could push to this repo
  + You can grant yourself access to a private repo 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 repo that you might not have write/push access to, you make your changes to your copy and then submit them back to the original repo
      * Forking allows a separation of write access

Organizations & Teams

* Team-level admins are called Team Maintainers
* No way to communicate from one organization to another within GitHub
* GitHub recommends using as few organizations as possible
* Having only one organization allows full communication between all of its members
* Admins will have access to all organizations
* Organization-level admins are called Organization Owners
* <https://github.com/blog/2020-improved-organization-permissions>
* Teams are like security groups
* People can belong to many teams
* Teams can be tied to LDAP groups
* A team can have a repo which they are admins of
* If you create a repo, by default it's not associated to any team but anyone in the NM organization logged into GitHub can read that repo, they just can't write to it. You would then have to go in and associate a team to that repo to give them write access to that repo.
* Organizations grant ownership of a repo
* Teams grant access to a repo
* When a user is added to the AG-GITHUB-USERS group, they are added to GitHub but they are not added to the organization. They would then need to be invited to the NM Organization from an Organization Owner.
* You can have Organization repos or user repos. Repos can be public or private
* You can have a repo be private to a single Team. So only that Team would have access to read and write to that repo
  + Teams would then grant access to read or write to that repo
* When you create a new repo you can create it either under NM Organization or under your own User
* Organization Owners had admin rights over the entire organization
  + You can have multiple owners of the organization
  + You can determine who the Owner of a repo is by looking to see if the repo starts with "NM" or your user ID
  + If you create a repo under an organization, you will be automatically added as an admin "collaborator" on that repo
* Collaborators can be set to read, write, or admin permission
* So to give someone admin access to a repo:
  + You can add them to a Team that has admin access to that repo
  + You can make them a collaborator with admin access
  + You can make them an Organization Owner
* When you are on a team, you can only grant access to repo's that you are an admin to

The GitHub Workflow

* Deploy early, deploy often
* Master = Trunk
* Master is what we consider to be our good source code, the build should always be passing in Master
* When you start a new feature, you branch off of Master
  + Give branches a brief but descriptive name, could relate to the feature you are developing (JIRA #)
  + Branch names should be readable so people know what you are working on
  + When you start working in this branch, none of your changes affect Master
  + You make changes and commit them to your branch with descriptive comments
  + Commit often
  + Commits to that branch are only on your own local computer
* After you finish coding against your branch you merge it back up to Master
  + First you open a Pull Request about the changes you want to submit to Master
  + You can "@[team name]" to alert a team to view your pull request to get peer review feedback
  + Based on the feedback you would make changes and add more commits to your branch
  + Your CI server would run the build off the branch
  + You would deploy the branch to Test and Stage
  + When you are ready to deploy to Production you would deploy to Prod from your branch
* After your prod deployment is successful you would merge back to Master
* You can "protect" a branch by Settings -> Branches
* You can delete your branch after you merge to Master, the commit history of the branch will be moved to Master
* You can use the "Releases" feature to point to certain commits
* You can turn "Issues" off at a repo level in the Settings -> Options tab in GitHub

Web Hooks

* Settings -> Webhooks
  + Will show a history of when the webhook was fired off
  + Can use the json output/payload for Jenkins to read and start builds/events off of
  + Json output/payload will differ based on the event type
* Webhooks can be added at the repo-level or at the organization-level

Created with Microsoft OneNote 2013.