Certified Jenkins Engineer (CJE) - 2017
==========

Note: It is recommended to follow the chapters chronologically. Some steps depend on the previous steps.

I.
Overview:
-open source
-plugin support
-Groovy language for writing plugins
-Windows/Linux
-lightweight
-Jenkins was build in Java

CInteg.:
-software development practice
-automation & speed is the focus
-assumes high degree of testing - unit/integration/smoke/acceptance
-workflow:
  -checkout from SCM/Git
  -branch and make local changes
  -add or change tests (for new functionality)
  -trigger automated build locally
  -if successful, consider committing
  -update with latest from mainline
  -push change, build and test on integration machine
-best practices:
  -maintain a single source of truth
  -have a common mainline branch (usually master)
  -automate the build every step
  -minimize potential for user error
  -make the build self-testing (ideally TDD)
  -everyone commits frequently (at least daily)
  -communication is key
  -frequent merges will help avoid conflicts
  -avoid large diffs by pulling and updating frequently
  -build every commit
  -prioritize fixing broken builds
  -keep your builds fast!
  -testing environment should be as close to prod as possible
  -make it easy for anyone to get the latest build
  -keep it open, everyone should see what is happening

CDeliv.:
-software is build so that it can be released to prod at any time
-always deployable throughout SDLC
-not breaking the build is priority over adding new features
-feedback is fast and prod readiness is known
-automate the deployment, push button deploy is possible with any version
-once again, communication and opennes is important

Difference CInteg. and CDeliv.:
CDeliv=release at any time
CI=practice of integrating the code continuously

Difference CDeliv. and CDeploy.:

CDeliv=code CAN be released at any time (potentially includes manual step)
CDeploy=code IS released continuously, automatically

II.
Prereqs (on CentOS7):
$ sudo su
$ yum list java*
$ rpm -Uvh jdk-8u121-linux-x64.rpm
$ which java
$ alternatives --install /usr/bin/java java /usr/java/latest/bin/java 200000
$ alternatives --install /usr/bin/javac javac /usr/java/latest/bin/javac 200000
$ alternatives --install /usr/bin/jar jar /usr/java/latest/bin/jar 200000
$ vi /etc/rc.local
  export JAVA_HOME="/usr/java/latest"

Installation v2.19.4 (on CentOS7):
$ sudo su
$ wget -O /etc/yum.repos.d/jenkins.repo ↵
https://pkg.jenkins.io/redhat-stable/jenkins.repo
$ rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
$ yum install -y jenkins-2.19.4-1.1
$ yum-config-manager --disable jenkins
$ netstat -tulpn | grep 8080
$ systemctl start jenkins
$ systemctl enable jenkins

Installation wizard (port 8080):
1) $ cat /var/lib/jenkins/secrets/initialAdminPassword
2) Paste to the password field
3) Install suggested plugins
4) Create first admin user:
  Username/Password/Confirm Password/Full name/E-mail address

III.
The Jenkins Dashboard:
-Log in to the server as admin
  [X] Enable auto refresh
  Edit description - company name
-'New item' is the core of Jenkins functionality:
  Freestyle project
  Pipeline
  External job
  Multi-configuration project
  Folder
  GitHub Organization
  Multibranch pipeline
-'People' is the list of users
-'Build History'
-'Manage Jenkins' has lots of options
-'My Views' for configuring view, customizable views
-'Credentials' is the list of all credentials
-'Build Queue'
-'Build Executor Status' default on master is two executors

User mgmt & Security:
-Manage Jenkins - "Configure Global Security" (defaults)
  Security realm/backend = Jenkins own user db

```
  Authorization = Logged-in users can do anything
    (!) Change to Matrix based security, that has the following categories
          Overall - as a whole
            Administer
            ConfigureUpdateCenter
            Read
            RunScripts
            UploadPlugins
          Credentials
            Create
            Delete
            ManageDomains
            Update
            View (on the main dashboard)
          Agent
            Build
            Configure
            Connect
            Create
            Delete
            Disconnect
          Job (also known as project)
            Build
            Cancel
            Configure
            Create
            Delete
            Discover (if a user doesn't have rights, it will bring you to the login page)
            Move (to a different folder)
            Read
            Workspace
          Run
            Delete
            Replay
            Update
          View (ability to have a dashboard with information that you want)
            Configure
            Create
            Delete
            Read
          SCM
            Tag
-Add an existing user and give him all the rights, click "Apply"
  If you try to add non-existing user, the name will be crossed-out
-Also create a new user (Manage Jenkins - Manage Users - Create User)
  Clicking on gear will give you user specific settings

Adding a Jenkins slave/agent/node
-when you scale up, load on the master might be too big and you need a slave
-offload build projects
-doesn't change how you interact with Jenkins
Steps to do:
  On the master $ sudo su
                $ su jenkins -s /bin/bash
                $ ssh-keygen
                $ cat /var/lib/jenkins/.ssh/id_rsa.pub
```

```
  On the slave  $ sudo su
                $ useradd -d /var/lib/jenkins jenkins
                $ mkdir /var/lib/jenkins/.ssh
                $ vi /var/lib/jenkins/.ssh/authorized_keys
                  <Paste the public key from the master here>
                $ INSTALL JAVA, SAME AS ON THE MASTER (II. Prereqs)
  On the master - in the Jenkins console:
    Manage Jenkins - Manage Nodes - New Node
    Node name: Slave1
    [X] Permanent Agent
        Name
        Description
        # of executors
        Remote root directory /var/lib/jenkins
        Labels - you may want to organize slaves by OS, space separated
        Usage
          [X]Use this node as much as possible
          Only build jobs with label expressions matching this node
        Launch method
          Launch agent via execution of command on the master
          [X]Launch slave agents via SSH
            Host: <FQDN>
            Credentials: Jenkins [Add Credentials]
              Username: jenkins
              [X]Private key: From the Jenkins master ~/.ssh
              Host Key verification strategy
                Known hosts file verification strategy
                Manually provided key verification strategy
                [X]Manually trusted key verification strategy
                Non verifying verification strategy
          Let Jenkins control this Windows slave as a Windows service
        Availability
          Keep this agent online as much as possible
          Take this agent online according to a schedule
          Take this agent online when in demand, and offline when idle

IV. Plugins
Three options to install plugins:
    a/ UI
    b/ Advanced method loadng *.HPI file via UI
    c/ Jenkins CLI
-Manage Jenkins - Manage Plugins
-Three tabs - Updates/Available/Installed/Advanced
-Installing the plugin will need a restart of Jenkins
-Uninstall doesn't give you the option to automatically restart, but you can
  "Manage Jenkins" - "Prepare to shutdown" - $ systemctl restart jenkins
-To install specific version of plugin, click on the plugin name - Archives - copy ↵
link of the version
  $ wget <PLUGIN LINK>.hpi
  Upload plugin
  [X]Restart Jenkins when installation is complete and no jobs are running
  You can only downgrade to one version (!) from the console
-Good practice is to use "Manage Jenkins" - "Prepare to Shutdown" and then ↵
http://jenkins_url:8080/restart

V. Projects
-New item - "My Freestyle Project":
```

```
  Discard old builds
    Strategy: Log rotation
    Days to keep builds
    Max # of builds to keep
    + Advanced options:
      Days to keep artifacts
      Max # builds to keep with artifacts
  GitHub project
    Project URL - not for webhooks, just the URL where the project is
  Project is parameterized
  Throttle builds - minimum time between builds, make sure they are spaced out (e.g.
  1 per hour)
    Number of build
    Time period
  Disable project
  Execute concurrent builds if necessary
  Restrict where this project can be run
    Label expression: "master" or "Node1" or "Linux" or "CentOS" - matching the slave
    label or name, you can use boolean operators etc.

  Advanced options:
  Quiet period
  Retry count
    SCM checkout retry count - how many times it will try to reach Git before it fails
  Block build when upstream project is building
  Block build when downstream project is building
  Use custom workspace
    Directory
    Display name
  Keep the build logs of dependencies - everything that is referenced by the build
  will be protected

  Source code mgmt:
    None
    Git
    SVN

  Build triggers:
    Trigger builds remotely (e.g. from scripts)
    Build after other projects are built
    Build periodically - cron format
    GitHub hook trigger for GITScm pollinh
    Poll SCM - cron format

  Build environment
    Delete workspace before build starts
    Abort the build if it's stuck
    Add timestamps to the Console Output
    Use secret text(s) or file(s)

  Build:
    Add build step

  Post-build actions:

Source code mgmt, Git plugin
-Install Git on master & slave:
```

```
  $ sudo su
  $ yum install -y git
-Add the 'jenkins' public key to Github
-Click on the project in Jenkins UI - Source code mgmt:
  Repository URL (clonable link): git@github.com:luckylittle/jenkins-ci.git
  Credentials: 'jenkins'
  Name:
  Refspec: - controls the remote refs
  Branches to build
    Branch specifier (blank for 'any'):
  Repository browser: githubweb
    URL
  Additional behaviors

  Build triggers:
    [X]Poll SCM
      Schedule: H/15 9-17 * * 1-5 (every 15 minutes Mon-Fri only between 9am and 5pm)

  Add build step
    Execute shell - command: git log

Git hooks & build triggers
-[X]Poll SCM, leave "Schedule" empty
-In Github, go to repo Settings - Integrations & services - Add a service - "Jenkins (Git plugin)"
  Jenkins URL: http://(jenkins_master):8080
-When you git push, it will automatically kick off the build
-There is another way of doing it, using the trigger "GitHub hook trigger for GITScm polling" and in Github, add service called "Jenkins (GitHub plugin)"
  Add Jenkins hook url: http://(jenkins_master):8080/github-webhook/
  Test service button in Github services section is useful

Workspace ENV variables
-Generic ENV provided by Jenkins
  $BUILD_NUMBER, $NODE_NAME, $JOB_NAME, $EXECUTOR_NUMBER, $WORKSPACE

Parameterized projects
[X]This project is parameterized - Add Parameter:
  Boolean parameter
  Choice parameter
  Credentials parameter
  File parameter
  List subversion tags (and more)
  Multi-line string parameter
  Password parameter - not protected from showing up in the build log (!)
  Run parameter (jobname, jobname.number, jobname_NAME, jobname_RESULT, jobname_JOBNAME, jobname_NUMBER...)
  String parameter
-Then you will be able to "Build with parameter"

Upstream/downstream
Upstream project=the 'parent' project, the 'predecessor' project, the project that triggers downstream project
Downstream project=the project that has been trigerred by an upstream project
Linked freestyle projects:
[X]Build after other projects are built -> this can be disabled if the "Trigger/call build on other projects" is defined (see below)
```

  Projects to watch
    Trigger only if build is stable
-How to pass parameters between the upstream/downstream:
  Plugins - "Parameterized Trigger plugin"
  Downstream project - must be parameterized - e.g. String parameter $IMPORTANT_PARAM
  Upstream project - Add build step - Trigger/call build on other projects -
  Downstream project - Add predefined parameters IMPORTANT_PARAM=$BUILD_NUMBER

Folders
-One way to manage projects
-New item - Folder
  Name
  Display name
  Description
  Health metrics - Items in nested sub-folders are used to calculate the folder's health
  PRoperties
  Pipeline libraries
  Pipeline model definition
-"Jenkins" path is the root folder of the projects, you can move existing projects to the newly created folder (click project properties - "Move")
-You can have nested folders

Views
-The global view is the same for all of the users
-On the default view, clicking on the "+" tab on the main screen
  View name
  List View - simple list format (choosing this will take you directly to the configuration page)
  My View - automatically diplays all the jobs that the current user has an access to
-After that, you can "Edit View" on the left side
  Name
  Description
  Filter build queue
  Filter build executors
  Job filters
    Status filter - all selected, enabled jobs only, disabled jobs only
    Recurse in subfolders [] - you will see all jobs in all folders below
    Jobs [] TOP LEVEL FOLDER
    Use regular expression to include jobs into the view: .*Downstream.*
    Add Job Filter
  Columns
    Status
    Weather
    Name
    Last success
    Last failure
    Last duration
    Build button
    ...
-Manage Jenkins - Configure system - Default view
-You can specify own set of per-user views in "My Views"
  Include a global view
  List view
  My view

VI. Pipelines
-Testing repository with sample Java project:

https://github.com/linuxacademy/content-jenkins-java-project
-Install Docker on the slave and the master
-Install Ant on the master and the slave
```
  tar xvzf apache-ant-1.10.1-bin.tar.gz -C /opt
  lns -s /opt/apache-ant-1.10.1/ /opt/ant
  sh -c 'echo ANT_HOME=/opt/ant >> /etc/environment'
  ln -s /opt/ant/bin/ant /usr/bin/ant
  ant -version
```
-Have a 'build.xml' file in the root folder (for compiling and building the JAR file) and test manually on the master if it can build it
```
  ant -f build.xml -v
  java -jar rectangle.jar <LENGTH> <WIDTH>
```

Jenkinsfile
-lives in the SCM that defines the Jenkins pipeline
-needs pipeline plugin
-2 styles:
  Declarative - defining the state as oppose to scripted
  Scripted - more like a bash script, Groovy programing style
Example - basic layout:
```
  pipeline {
    agent any   <- which agent are we going to use for this pipeline (any/none/label '<MATCH>'/docker '<IMAGE>')

    stages {   <- stages of build
      stage('Build') {
        steps {
          echo 'Buidling..'
        }
      }
      stage('Test') {
        steps {
          echo 'Testing..'
        }
      }
      stage('Deploy') {
        steps {
          echo 'Deploying..'
        }
      }
    }
  }
```
-directives can be either on the top and it applies to all steps, or in each step explicitly (e.g. different docker image for test and for deploy)
-tons of different "steps" associated with plugins
-"sh" for a shell script is the most commonly used
-"echo" prints a string
-environment directive sets EVN vars and they are available from the scope where they are defined:
```
  environment {
    ENV_VAR = "my value"
    }
```
-Configuring the actual pipeline in the Jenkinsfile (use Groovy lint in your IDE):
```
  pipeline {
    agent any

    stages {
```

```
      stage('build') {
        steps {
          sh 'ant -f build.xml -v'
        }
      }
    }
  }
-Add Githook functionality (Github - Project - Settings - Integration & Services -
Jenkins hook url)
-Add new item in Jenkins UI - Pipeline - My Java Projects
  [X]Github project
  [X]GitHub hook trigger for GITScm polling
  Pipeline
    Definition: Pipeline script from SCM
    SCM: Git
      Repository URL
      Credentials
    Branches to build
      Branch Specifier (blank for 'any'): */development
    Repoitory browser: githubweb
      URL: <PROJECT URL>
    Script path: Jenkinsfile

Artifacts & fingerprints
-Add 'post' directive to the Jenkinsfile:
  post {
    always {   <-- this can be also 'success'
      archive '<DIR>/*.jar'
    }
  }
-When you click on the build #, you will see a new column "Build Artifacts"
-If you want better tracking of artifacts (the additional option "See Fingerprints"
on the left in each build)
  post {
    always {
      archiveArtifacts artifacts: '<DIR>/*.jar', fingerprint: true
    }
  }
-Everything lands on the master (!) inside
'/var/lib/jenkins/jobs/<JOB_NAME>/builds/<BUILD_NUMBER>/archive/<DIR>/*.jar'
-Install new plugin 'Copy Artifact Plugin', we can set permissions between projects
to allow artifacts copying
  [X]Permission to Copy Artifacts:
  New build step:
    Copy artifacts from another project:
      Project name:
      Which build: Latest successful build/Latest saved
      build/Upstream/Downstream/Last completed (ignoring status)/Specific etc etc.
      Artifacts to copy: <DIR>/rectangle.jar
      Artifacts not to copy
      Target directory
      [X]Fingerprint artifacts
-When you go to the "Workspace" in the project, you will see the artifact(s)
-Retention policy can be set up in Jenkinsfile:
  options {
    buildDiscarder(logRotator(numToKeepStr: '2', artifactNumToKeepStr: '1'))
    }
```

-In the console this is called "Discard old builds", click Advanced and also post-build action: "Archive the artifacts - File to archive: <DIR>/rectangle.jar"

VII. Testing
Types:
-unit=very specific functionality testing, individual classes, methods etc.
-smoke/functional=sanity testing, general functionality as a whole (came from mechanical engineering, "does the car go?", if not machine start to smoke), after unit testing
-integration=multiple modules/pieces coming together and have expected functionality
-acceptance=business requirement(s) testing, am i meeting all of the requirements?, towards the end
-code coverage=you're testing level of testing, the lower level testing, all of the functionality, is there something untested? "Cobertura" plugin for Java

JUnit & Ant
-Write tests, add junit-4.10.jar into the lib/ directory, add test.xml to include this jar file among other things
-Add new stage 'Unit Tests' to the Jenkinsfile:
```
  sh 'ant -f test.xml -v'
  junit 'reports/result.xml'
```
Note: Watch out for "GitHub hook trigger for GITScm polling option", sometimes this option does not retain when updating Jenkinsfile (bug?)
"Latest test result" will show you:
  Duration
  Fail
  Skip
  Pass
  Total

Deployment to Apache
-Install Apache on the master, create subdirectory accessible by 'jenkins' user in /var/www/html
```
  mkdir -p /var/www/html/rectangles/{all,green}   <--green will be used later in the advanced topics
  chown -R jenkins:jenkins /var/www/html/rectangles
  systemctl start httpd; systemctl enable httpd
```
-Make sure this is only executed on master (agent label 'master' or create a new label 'apache')
-Add another deploy stage after build
```
    sh 'cp dist/rectangle_${env.BUILD_NUMBER}.jar /var/www/html/rectangles/all'
```
-If you trigger a pipeline, you will be able to see jar file in http://(jenkins_master)/rectangles/all

Functional testing in different environments
-We want to test our jar file in CentOS and Debian environments
-Change the global 'agent' declaration to 'none'
-Change the Unit test, build & deploy stages declaration of 'agent' to label 'apache'
-Add a new stage "Running on CentOS", agent label 'CentOS', steps:
```
  sh 'wget http://(jenkins_master)/rectangles/all/rectangle_${env.BUILD_NUMBER}.jar'
  sh 'java -jar rectangle_${env.BUILD_NUMBER}.jar 3 4'
```
-To test Debian, the best way is to use Docker
```
    stage("Test on Debian") {
      agent {
        docker 'openjdk:8u121-jre'   <--this is an image from Docker hub
      }
```
-Add another step under "Test on Debian" that is exactly the same as CentOS - wget

```
the JAR file and run Java

VII. Advanced topics
Multibranch Pipelines and Code Promotion
-If the step fails, all the subsequent steps won't run
-We want to add another step "Promote to green":
  steps {
    sh 'cp /var/www/html/rectangles/all/rectangle_${env.BUILD_NUMBER}.jar ↵
    /var/www/html/rectangles/green/rectangle_${env.BUILD_NUMBER}.jar'
  }
-Only the artifacts that passed the previous "Tests on CentOS and Debian" will be ↵
able to make it to the ../green folder
-Until now, we push to the development branch and we want to automatically promote ↵
the code to master
-Note: You can have conditions in the Jenkinsfile:
  when {
    branch 'development'
  }
-When we have trunk branch where everyone pushes the code and then when tests are ↵
fine we want to promote to a different branch - we need to select "Multibranch ↵
pipeline"
  Name
  Display name
  Description
  Branch Sources
    Add source - Git   <--not GitHub, because it uses HTTPS
      Project repository: git@...
      Credentials: jenkins
      Repository browser: githubweb
        URL: https://...
      Additional Behaviours
      Advanced:
        Include branches
        Exclude branches
        Property strategy - All branches get the same properties
  Health Metrics
    [X]Recursive - individual projects will dictate the health
  Pipeline Libraries
  Pipeline Model Definition
-After you save, it is determining the available branches
-Disable the previous pipeline
-Add a new stage "Promote Development to Master"
  agent {
    label 'apache'
  }
  when {
    branch 'development'
  }
  steps {
    sh 'git stash'   <--stashing any local changes
    sh 'git checkout development'
    sh 'git pull origin'
    sh 'git checkout master'
    sh 'git merge development'
    sh 'git push origin master'
  }
-Change "when branch development" to "master"
```

-ENV var called ${env.BRANCH_NAME} is only available in multibranch pipeline project
-Use this ENV var to create subdirectories where artifacts will be stored (../rectangles/all/development etc.)
-Use "Scan Multibranch Pipeline" on the left pane
-Fully automated merge will be triggered after the development build is finished

Tagging (http://semver.org/)
<MAJOR_VERSION>.<MINER_VERSION>
-To do this, you need to introduce ENV vars in the global scope of the Jenkinsfile

```
  environment {
    MAJOR_VERSION=1
  }
```

-And add another step to the "Promote Development branch to Master"

```
  sh "git tag ${env.MAJOR_VERSION}.${env.BUILD_NUMBER}"
  sh 'git push --tag'
```

Notifications
-Manage Jenkins - Configure System - Extended E-mail Notification
  SMTP server
  SMTP port
  Default recipient
-Also adjust "Jenkins Location" - "System Admin e-mail address"
-Add the catchall notification when anything fails in Jenkinsfile (global scope, outside of stages)

```
  post {
    failure {
      emailext(
        subject: "${env.JOB_NAME [${env.BUILD_NUMBER}] Failed!",
        body: "<html> bla bla bla bla</html>",
        to: "admin@domain.com"
      }
    }
  }
```

-Add similar alert when you build a master (post on success) - not global scope
-On the freestyle type project, you can add "Editable Email Notification" as a post-build action
  Project Recipient List
  Project reply-to list
  Content type
  Default subject
  Default content
  Attachments
  Attach build log
  Advanced
    Triggers
      Aborted
      Always
      Failure
      Fixed
      Not built
      Success
      etc.
        Send to: Recipient list

Shared libraries
-library that has methods accessible amongst different projects
-extend the functionality of your pipeline

-Groovy/pipeline syntax
-Put a file "sayHello.groovy" inside jenkins-global-library/vars
```
  def call(String name = 'you') {
    echo "Hello, ${name}"
  }
```
-To add it on the global level - Manage Jenkins - Configure System - Global Pipeline Libraries - Add
  Name
  Default version: master
  Load implicitly [X]
  Allow default version to be overriden [X]

  Retrieval method
    Modern SCM
    [X]GitHub
      Owner
      Scan credentials
      Repository: jenkins-global-library
-Modify Jenkinsfile to invoke it
```
  stage('Say Hello') {
    agent any
    steps {
      sayHello 'Lucian Maly'
    }
  }
```
-We can add it per specific project as well (!)
-Scripted pipeline is more flexible when you call library
```
  script {
    def myLib = new linuxacademy.git.gitStuff();
    echo "My Commit: ${myLib.GitCommit("${env.WORKSPACE}/.git")}"
  }
```

VIII. CLI & API

CLI
-Associate the SSH public key with the user: Manage Jenkins - Manage Users - gear - SSH Public Keys
-Download the Jenkins client:
```
  wget http://(jenkins_master):8080/jnlpJars/jenkins-cli.jar
  echo "JENKINS_URL='http://localhost:8080'" >> /etc/environment
  echo "alias jenkins-cli='java -jar jenkins-cli.jar'" >> ~/.bashrc
  <LOGOUT><LOGIN> the bash session
  jenkins-cli
    Usage: java -jar jenkins-cli.jar [-s URL] command [opts...] args...
    Options:
      -s URL
      -i KEY
      -p HOST:PORT
      -noCertificateCheck
      -noKeyAuth
  jenkins-cli help
```
-Few different commands that are useful:
```
  jenkins-cli who-am-i
  jenkins-cli build "Freestyles/My Freestyle Project"
  jenkins-cli version
  jenkins-cli shutdown
  jenkins-cli safe-shutdown
```

```
  jenkins-cli restart
  jenkins-cli install-plugin thinBackup -restart
  jenkins-cli console "Freestyle/My Freestyle Project" 51

API
-XML & JSON
-There is an issue with this particular version of Jenkins, so you need to uncheck/disable:
  Manage Jenkins - Configure Global Security - []Prevent Cross Site Request Forgery exploits
-Go to http://(jenkins_master):8080/me/configure (same as Manage Users - your username)
  API Token
    Show API Token
-You can see the API help, when you add /api at the end of the URL and it will show you the help page
-Examples using cURL:
  curl -X POST http://(jenkins_master):8080/job/Freestyles/job/My%20Freestyle%20Project/buildWithParameters --user luckylittle:<API_TOKEN>
  curl -X POST curl -X POST http://(jenkins_master):8080/job/Freestyles/job/My%20Freestyle%20Project/buildWithParameters?BRANCH=foo --user luckylittle:<API_TOKEN>
  curl http://(jenkins_master):8080/job/Freestyles/job/My%20Freestyle%20Project/config.xml --user luckylittle:<API_TOKEN>
  curl http://(jenkins_master):8080/job/Freestyles/job/My%20Freestyle%20Project/api/json --user luckylittle:<API_TOKEN>
  curl -X post http://(jenkins_master):8080/job/Freestyles/job/My%20Freestyle%20Project/disable --user luckylittle:<API_TOKEN>
  curl http://(jenkins_master):8080/job/Freestyles/job/My%20Freestyle%20Project/43/consoleText --user luckylittle:<API_TOKEN>
  curl -X POST http://localhost:8080/quietDown --user luckylittle:<API_TOKEN>
  curl -X POST http://localhost:8080/cancelQuietDown --user luckylittle:<API_TOKEN>
  curl -X POST http://localhost:8080/safeRestart --user luckylittle:<API_TOKEN>
```