Components:

CycleServer:

- Database used to store internal settings and configuration (templates, current clusters)
- Webserver that generates a webUI that allows for easy configuration
- This is running on one of Cycle Cloud's servers, we have access to their UI through https://{{IP}}/home
- The webUI and the CLI are interchangeable, both present different ways of interacting with the cycleserver.

Pogo.ini:

- This is the file that defines parameters of your cycle cloud setup (endpoints, access keys)
- Must be in .cycle of your home directory before initializing cyclecloud

Cyclecloud.pem:

- The private key associated with all clusters that you start up
- Enable automatic ssh with
    - cd .cycle
    - chmod 400 [my-keypair.pem]
    - eval `ssh-agent -s`
    - ssh-add [my-keypair.pem]

Cyclecloud CLI executable:
Commands:

- Initialize
    - Initialize the account, associate it with a server and user
        - https://docs.cyclecomputing.com/installation-guide-v6.6.0/installation
    - This must be run before anything else and will populate .cycle with other config data for the cycle cloud account
- Create_cluster
    - Create a cluster based off of a given template (see below)
    - *Cyclecloud create_cluster "TEMPLATE1" cluster_name*
- Import_template
    - Make a template accessible to create new clusters
    - Ex: *cyclecloud import_template "TEMPLATE1" -f sge_barebones_template.txt*
- Start_cluster cluster_name
    - Will start the cluster, steps: Provisioning → software install → sshable
- Terminate_cluster cluster_name
    - Will stop a cluster and shut it down (won't delete)
- Project init project_name
    - Creates a new project along with starter files inside of it (more on projects below)
- Delete_cluster cluster_name
    - Deletes cluster (only works if terminated)

- Show_cluster
    - Will list all current clusters you own and give details about their stages
- https://archive.cyclecomputing.com/cyclecloud/2.8.1/guide/commands.html

Templates:

- Whenever you create a template you are defining parameters that the instance will use for configuration
    - Ex. autoscaling? Image (refers to Ubuntu, centos7 etc)? Project name?
    - Templates are defined in files, a few have been given out in emails
        - These files must be imported as templates (which saves them in the cycle server) and then you can create new clusters from them

Projects:

- Spec:
    - A spec defines a specification for a type of startup procedure
    - We will be fine using the default spec directory
- Cluster-init
    - Defines startup procedures for new nodes
    - Contains scripts, files, and tests
        - Scripts are run lexicographically each time a new cluster is spun up
        - Files are data files that can be sent to each node (see unknown below)
- Specs can be referenced inside of templates
- Lockers define “S3-ish” storage buckets where the projects are uploaded
- If you run “*cyclecloud project upload azure-storage”* it will upload the project to a locker, so that it can be run when it is pulled down by a node when it is reading its template
- On clusters created, these files will be stored in
    - /mnt/cluster-init/sge_data/default/files
- Script output will be in
    - /opt/cycle/jetpack/logs/cluster-init/
- https://docs.cyclecomputing.com/administrator-guide-v6.6.0/projects

Example:

- The following is the beginning of a slightly altered barebones template emailed out
- We are creating a template SGE_BAREBONES_UBUNTU
- Referencing a project sge_data that we have created
- If you are creating new templates: beware of the following params. “Cluster _____” must match the name of the template you are trying to import. project = ____ must match a real project stored in azure storage. Locker = _____ must be “azure-storage”

```
##########################
## SGE Configuration File ##
##########################

[cluster SGE_BAREBONES_UBUNTU]

# Enable/disable autoscaling
# The scheduler load will determine the number of execute machines that are started, machines will terminate themselves if they are idle
# and approaching a billing cycle.
Autoscale = True

    # defines default values used for all nodes. This is inherited automatically by all nodes.
    # You can create other nodes like this by adding Abstract = true to them and adding
    # Extends = foo to the nodes that should inherit the definitions from foo.
    [[node defaults]]
    Credentials = azure
        Region = westus
    KeyPairLocation = ~/.ssh/cyclecloud.pem

    ## NOTE: You must change this subnet to match what you have access to in the UI
    SubnetId = scc17-sandbox15/DefaultNetwork_westus/default
    ImageName = cycle.image.centos7


        [[[cluster-init default]]]
        Project = sge_data
        Spec = default
        Version = 1.0.0
        Locker = azure-storage
```

Procedure:

- Pogo.ini and cyclecloud.pem must be moved to the .cycle directory of your home directory. You then must initialize and cycle cloud to connect to the cycle server instance
- Create a cluster using a template. Check out the functional sge_barebones_template in my directory
- Start the cluster, either from the cli or the UI
- Ssh into the cluster
- Run jobs using qsub

Qsub:

- Once you are on the provisioned node, this is considered the master node
    - No work is done from this node
- If you want to do any work you use qsub, which will add work to the queue, spawn up necessary execute hosts, and run the script
- For some reason when I ran qsub with a simple shell script echoing hello, it did not create separate output files. I'm not sure if it didn't worked, or just directed output to somewhere else. However running qstat (displays the queue) it did spin up an execute cluster, and then spin down that cluster after the job was taken off of the queue.
- http://gridscheduler.sourceforge.net/htmlman/htmlman1/qsub.html
- http://bioinformatics.mdc-berlin.de/intro2UnixandSGE/sun_grid_engine_for_beginners/how_to_submit_a_job_using_qsub.html

Gotchyas:

- If you define a new project, upload it, and try to create a new cluster using that template, you must make sure the scripts and tests inside of the spec referenced in the template are runnable and correct. I think the cause of my blocker was a readme file that was

auto-generated in the scripts directory that was attempting to run but failing over and over. If a script fails it will be run again and again to infinity.
    - If you have an error here it will not fail fast
- After running ./cyclecloud initialize, ~/.cycle will be populated with default templates
    - If you import a template as Grid Engine, it will overwrite the default binding of Grid Engine to sge_template.txt in ~/.cycle
    - To get back the GUI to create Grid Engine clusters you need to import Grid Engine from the sge_template.txt adding --force at the end
- Spinning up an execute cluster can take some time
    - It spends a while provisioning hosts, not sure what our priority is for grabbing resources but maybe others are higher than us.

Email notes:

- Must use a cluster instance that you set up before the competition
- Only place we can upload to cloud is from within our own cluster
- Can connect to local subnet (your cluster)

Side notes:

Cluster Type Options:
When you go to add a new cluster from the GUI, you will be faced with several options that Cycle cloud supports.

- Kafka is for distributed streaming, customer based applications
- Redis is for caching and data store
- Zookeeper is for distributed synchronization
- HTC is for high throughput workloads but is more daunting to start with
- Our main candidates are: PBSpro or Grid engine, both of which are job schedulers. Most emails have been going off of grid engine templates so I have been using those