Redis Github:

- https://github.com/redis/redis

Initial Steps in Porting Redis:

1. Make a directory that is meant specifically for you.
    a. Ex: mkdir aquiles

2. Go into that directory
    a. cd aquiles

3. Use the command “zopen generate” to specify the tool
    a. How it is formatted:
        i. Project Name
        ii. Description
        iii. License
        iv. Source URLs
            1. Grab the SSH link from the project
        v. Build Dependencies
            1. In the READ.me, the build dependencies should be listed
        vi. Runtime Dependencies
            1. In the READ.me, the runtime dependencies should also be listed

4. CD into your project name
    a. Ex: cd redisport

5. Go into the buildenv file using a text editor
    a. This could be done with an editor outside of the UNIX system such as VS Studio, but it could also be done with Vim or nano.

6. Ensure the information there aligns with the build information from the official redis github.
    a. The buildenv file defines the environment in which the project will be built and tested. Here are some aspects of this file:
        i. Dependencies: Lists the libraries and tools necessary to build the project. May have specified specific versions.
            1. How it may look:
                a. export ZOPEN_STABLE_DEPS="make openssl which pkgconfig check_python"

ii. Environment variables: Sets the variables needed for the build process like paths to tools and libraries
    1. export ZOPEN_STABLE_URL="https://github.com/redis/redis.git"

iii. Build tools: Specifies the tools and compilers used for building the project
    1. export ZOPEN_COMP="CLANG"

iv. Scripts: Automate the process of building
    1. # bump: redis-version /REDIS_VERSION="(.*)"/ https://github.com/redis/redis.git|semver:*
    2. # REDIS_VERSION="V.R.M" # Specify a stable release

v. Configurations Settings: Contains settings that configure the build process such as optimization flags and compiler options
    1. zopen_pre_patch()
    2. {
    3. export CFLAGS="$CFLAGS $CPPFLAGS -mzos-target=zosv2r5 -D__XPLAT -D_OPEN_THREADS=2 -D__thread= -DMAP_ANON=0"
    4. }

7. Save changes then exit out of buildenv
    a. Use ctrl+shift+o to save the changes made to the file
    b. Use ctrl+shift+x for exiting the file

8. Build the project using zopen build -vv
    a. -vv will allow for more detailed information about how the project is building

<u>Commiting changes to Github:</u>

1. Cd into the main repository
    a. Ex: cd aquiles/redisport/redis

2. Use git diff to take the changes made and put in your patch
    a. Ex: git diff > ../patches/pr.patch

3. Create a new patch that has all the new changes:
    a. Ex: git add buildenv patches/pr1.patch

4. (Optional) You could remove a previous patch:
    a. Ex: rm patches/PR1.patch

5. Specify the SSH key you are using to get ready to commit:
    a. Ex: export GIT_SSH_COMMAND="what you have as the ssh"

6. Commit your changes using “git commit”
    a. This will take you to a screen asking you to add a comment for that commit.
    b. Once you are done press escape to get out of insert mode
    c. From there press “:” by doing Shift+; which will put you into command mode
    d. Type wq to save then quit.
        i. This way the commit goes through

7. (Optional) Fix the SSH authorization error since same user is shared:
    a. First use this: export GIT_SSH_COMMAND="/bin what you have as the ssh"
    b. export GIT_TRACE=1

8. Push it into the main repository using “git push”

<u>Process of porting redis to the zopen community:</u>

Week 1:

- Worked on finding dependencies that redis used to work.
    - In order to do this, we needed to go to the redis Github and find the Install file, which normally contains these dependencies. In this case, we were redirected to the README.md file.
    - Once we saw all of the dependencies from Github, we put them into our buildenv file which will allow us to build this tool through z/OS.
        - In the file, it looks like this:
            - export ZOPEN_STABLE_DEPS="make openssl which pkgconfig check_python"
                - Inside the parenthesis, we add the dependencies that are needed to compile this tool

    - From there, we made sure that all of the dependencies were supported by z/OS which some were not so we had to exclude them from the list.
    - Once that was set up, we wanted to build this tool to see if there were any errors that we might encounter that weren't the missing dependencies.
    - Unfortunately, there were some errors in redis's source files that we needed to fix.
        - This would have to do with how these files compile in z/OS and how it would ignore flags placed to streamline the file-building process.

Week 3/11 - 3/14:

- I went to build the redis port and found these errors:
    - clang-14: error: no such file or directory: '../deps/hdr_histogram/libhdrhistogram.a'
    - clang-14: error: no such file or directory: '../deps/fpconv/libfpconv.a'
    - clang-14: error: no such file or directory: '../deps/fast_float/libfast_float.a'

- When looking for these files using the "find" command, we saw that it wasn't created.
    - This was due to the files not compiling in our buildenv.

- To fix this, we changed our file and added a new template that would allow us to change how MAKE works.
    - This is how it looks:
        - export ZOPEN_MAKE="zopen_make"
- From there we defined what the make should take in and reran our program.

    - Here are the specified dependencies:
        - zopen_make() {
        - cd deps; make "$@" fast_float fpconv hdr_histogram hiredis jemalloc linenoise lua; cd -
        - make "$@"
        - }

- This temporarily fixed the problem of the files not being created, but a new error was created in the fast_float file where one of the defined functions has an error.

- To troubleshoot this, we added two checks for the defined variable which would display an error depending on which defined variable failed first.

- Given this, we looked into every instance of the variable in our paths and found that it needed to run C++14. We then went into our buildenv so that when we built the program, the files would compile as C++14.
- This didn't fix our problem since it seemed that it was reverted to C++11 when compiling. We then found that in a file that redis was using to compile, it was hardcoded to be C++11 so we changed it to see if it would fix it and luckily it did.

- From there, we got a new error about the mutex not being included in one of the files. We looked at the file where the error was coming from and found that the Pthread initializer has a condition that checks if it is greater than 2 but since we set -D_OPEN_THREADS to 2, it wouldn't run. In order to fix this, we set it to 3 so that it could run.
    - This is how it looked:
        - export CFLAGS="$CFLAGS $CPPFLAGS -mzos-target=zosv2r5 -D__XPLAT **-D_OPEN_THREADS=3** -D__thread= -DMAP_ANON=0 -D_UNIX03

- We tried to build redis once again and found that the files were still not being created causing an error. This was weird since it was making these files before. These files are made in the buildenv and we found that the " make "@" " caused these files to not be made.
    - Here is the code:
        - zopen_make() {
        - cd deps; make "$@" fast_float fpconv hdr_histogram   hiredis linenoise lua; cd -
        - make "$@"
        - }

- We will continue to work on this next week.