k8s - Task #3 - Final

Goals

- Advanced processing flow with k8s jobs

- Prepare several input text files to JARVIS
- Create a new image for *jarvis-service* that incorporates these files
- Push image with a new tag to private repository
- Write python code that creates jobs to process the input files and save results
- Monitor job status and read back results after finish

Estimated Time: 60-90 minutes

Prerequisites

All steps below should be performed in your personal Cloud9 environment.

Steps

New JARVIS Service Image

1. Create 3-5 different input text files for JARVIS
   Can be of arbitrary name, but it is encouraged to number them as:
   input_0.txt, …, input_5.txt and so on

2. Update the previous *Dockerfile* for *jarvis-service* to copy these input files into a known location inside the image (under */root* is OK)
3. (Optional) Make sure the image is still configured with **ENTRYPOINT** directive so that it is possible to pass arguments over to the jarvis python script

4. Build the image
5. Tag it as v2
6. Push the image to your private repository

Send Jobs Using Python

1. Create a new python script file *send_jarvis_to_k8s.py*

2. Based on this example, copy the code with the following modifications
3. In function *create_job_object*
    a. Modify the job definition such that it refers to the *jarvis-service* image in the private repository (lines 31-34 of interest)
    b. Update the code to create as many jobs as input files prepared (each job should refer to a different input)
    c. Return a list of jobs created

4. In function *create_job*
    a. Modify to accept a list of jobs and send them in parallel
    b. It should print initially the job status and additional info to all jobs deployed
    c. Loop over all jobs *api_response* and call *get_job_status* to wait until each job finishes

5. In function *main*
    a. Comment-out calls to *update_job* and *delete_job*

6. Run the script to send jobs for execution

7. (Optional) Copy the final result from each finished job back to your Cloud9 environment
    a. May be accomplished by direct invocation of kubectl with cp subcommand from python script
    b. More advanced - use kubernetes *stream* API