Nautilus Direct Access Tutorial

### Prerequisites

You will need administrative root or sudo access to a Linux, Mac, or Windows system for this procedure. This tutorial assumes you are running Ubuntu but can easily be modified for other Linux distributions by changing the docker install procedure and should work on MacOS or Windows with a bit more modification. You will also need an account on Nautilus with access to a namespace (see “Nautilus Getting Started” document). Finally, you will need to download the “TutorialData” folder to your system.

This tutorial walks through the process of creating and interacting with a Kubernetes Docker Pod on Nautilus. Pods are intended for learning, development, and experimentation and are restricted to 2 CPU cores, 8GB RAM, and 6 hours runtime. When running computational workloads, it is highly recommended to use Kubernetes Docker Jobs which do not have these restrictions. Small modifications (documented at <http://ucsd-prp.gitlab.io/userdocs/running/jobs/>)are required to transition from a Pod to a Job.

Note: items in (*italics*) are commands to be run in a terminal.

* Follow the quick start guide (<http://ucsd-prp.gitlab.io/userdocs/start/quickstart/>) to configure your Linux system to access Nautilus
* Install Docker on your system (*sudo apt install docker.io*)
* Create a docker hub account at <https://hub.docker.com/>
* Login to docker (*docker login*)
* Get a base ubuntu docker image (*docker pull ubuntu*)
  + Note: when creating your own images, you can pull any existing docker image you would like to use as a base in this fashion
* Customize your docker container
  + Start a container from the base image (*docker run -t -d --name base ubuntu*)
    - Note: “base” can be replaced with any name you would like to use for your container
  + Ssh into running container (*docker exec -it base /bin/bash*)
  + Install Spades – this is a genomic assembler we will use for this example
    - (*echo "deb http://debian.bioinf.spbau.ru /" | tee -a /etc/apt/sources.list*)
    - (*apt-get update*) – you may see some warnings when running this command which can be ignored
    - (*apt-get install -y spades*)
  + Leave the container (*exit*)
* Create a new image from your customized container (*docker commit -m "Created example image" -a "Your Name" base YourDockerUsername/example-image:v1*)
  + Note: “Created example image” can be replaced with any desired commit note, “example-image” can be replaced with any desired image name, and “v1” can be replaced with any desired version number
* Upload your customized image to docker hub (*docker push YourDockerUsername/example-image:v1*)
* Cleanup
  + Stop your running container (*docker stop base*)
  + Remove your container (*docker rm base*)
  + Delete the local copy of your image (docker rmi *YourDockerUsername/example-image:v1*)
* Create a .yaml file (used to instantiate jobs on Nautilus) to allocate a persistent volume claim for storing your working data
  + For this example, you can use the “create-volume.yaml” file in in the “Tutorial Data” folder
  + Note: this method only allows one worker pod to access the storage at a time. For other storage options see: <http://ucsd-prp.gitlab.io/userdocs/storage/toc-storage/>
  + Below is an explanation of key entries
    - name: testvol – this is where you set the name of your volume
    - storage: 20Gi – this is where you set the size of your volume
* Create a .yaml file to create a pod to run your docker image on Nautilus
  + For this example, you can use the “example-pod.yaml” file in the “Tutorial Data” folder
  + You will need to add your docker username to the line “image: docker.io/YourDockerUsername/example-image:v1”
  + Below is an explanation of key entries
    - name: example-pod – name of your pod
    - image: docker.io/YourDockerUsername/example-image:v1 – the docker image you want to run
    - args: ["sleep", "36500000"] – the command you want to run in your pod, in this case we are telling the pod to wait for manual commands for 36500000 seconds
    - mountPath: /testvol – the location in the pod where we want to mount our volume
    - name: testvol – the name of the volume we are mounting
  + Note: this pod does not request a GPU and will only have access to CPU resources, for an example .yaml file that creates a GPU pod, see the file “gpu-example-pod.yaml” file in the “Tutorial Data” folder – more information is available at <http://ucsd-prp.gitlab.io/userdocs/running/gpu-pods/>
* Make sure both .yaml files are in your current working directory
* Allocate your volume (*kubectl create -n YourNautilusNamespace -f create-volume.yaml*)
* Run your docker image in a pod (*kubectl create -n YourNautilusNamespace -f example-pod.yaml*)
* Check your running pods (*kubectl get pods -n YourNautilusNamespace*) – do not proceed until the status of your pod is “Running”
* Transfer input data to your allocation (*kubectl cp /[PathToYour]/SampleData/ YourNautilusNamespace/example-pod:/testvol/SampleData/*) – this will take a few minutes
* Ssh into your pod (*kubectl exec -n YourNautilusNamespace -it example-pod -- /bin/bash*)
* Run your analysis (*spades.py -1 /testvol/SampleData/SRR396636.sra\_1.fastq -2 /testvol/SampleData/SRR396636.sra\_2.fastq --careful --cov-cutoff auto -t 20 -o /testvol/SampleOutput*) – this will take several minutes, you can cancel (ctrl + c) the process if you don’t want to wait
* Exit your pod (*exit*)
* Transfer output data to your system (*kubectl cp YourNautilusNamespace/example-pod:/testvol/SampleOutput /[YourDesiredPath]/sampleOutput*) – this will take a few minutes
* Stop your running pod (*kubectl delete -n YourNautilusNamespace -f example-pod.yaml*)
* Your storage allocation will persist on Nautilus until it is deleted, you can see check the status with (*kubectl get -f create-volume.yaml -n YourNautilusNamespace*)
* Delete your allocation (*kubectl delete -n YourNautilusNamespace -f create-volume.yaml*)
* Reminder: When running computational workloads on Nautilus, it is highly recommended to use Jobs rather than pods. This enabled greater resource requests, automates your workflow and only requires adding a few additional commands to your .yaml file. This procedure is documented here: <http://ucsd-prp.gitlab.io/userdocs/running/jobs/>