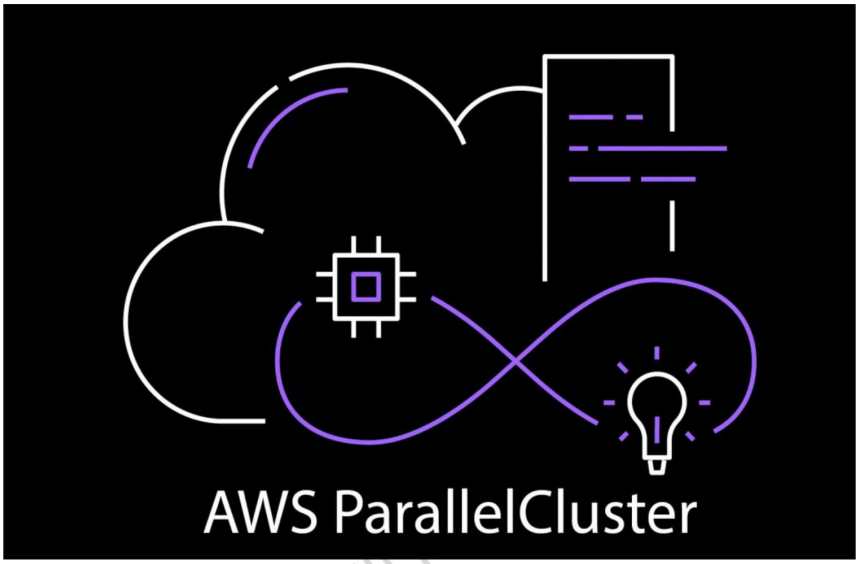


AWS ParallelCluster Lab

Getting started with AWS ParallelCluster
Launching from Cloud9

Please send comments, typos, suggestions to:
hedgesl@amazon.com

1. Table of Contents

2 Overview

High Performance Computing (HPC) harnesses large amounts of compute to answer all sorts of questions. Whether scientific endeavor, financial risk assessment, climate prediction, or rocket science, the goal of HPC is to apply mathematical models that produce the analyses, that solve problems, while spending as little time and money as possible, producing that data. The time to results is accelerated by tools that reduce the time spent standing up an HPC infrastructure and deploying workloads.

AWS ParallelCluster is an AWS supported Open Source cluster management tool that makes it easier to deploy and manage High Performance Computing (HPC) clusters in the AWS cloud. It automatically sets up the required compute resources, a shared filesystem, and offers a variety of batch schedulers such as AWS Batch, SGE, Torque, and Slurm. AWS ParallelCluster facilitates proof of concepts (POCs) and production deployments.

Amazon Web Services (AWS) offers the ability to create scalable compute capacity to power your High Performance Computing (HPC) workloads, on demand. In this Lab, you will deploy an HPC cluster on AWS ParallelCluster. The lab starts by deploying AWS Cloud9 and uses the Integrated Development Environment, IDE, to install and launch a cluster to EC2 compute nodes.

3 Objectives of the Lab

In this lab, you will learn how to:

- ❒ Log onto the AWS Management Console
- ❒ Explore the AWS Management Console
- ❒ Launch AWS Cloud9
- ❒ Install AWS ParallelCluster and Define the Cluster Configuration
- ❒ Create an Elastic Cluster
- ❒ Log onto the Cluster and Launch a Job
- ❒ Tear down the cluster

4 Pre-Requisites

This lab requires:

- ✓ A laptop with Wi-Fi running Microsoft Windows, Mac OS X, or Linux.
- ✓ An Internet browser such as Chrome, Firefox, Safari, or Edge.
- ✓ An AWS account.
- ✓ Familiarity with common Linux commands

Users will spin up AWS cloud9 and EC2 instance resources. Total cost is expected to be only a few dollars for the Lab.

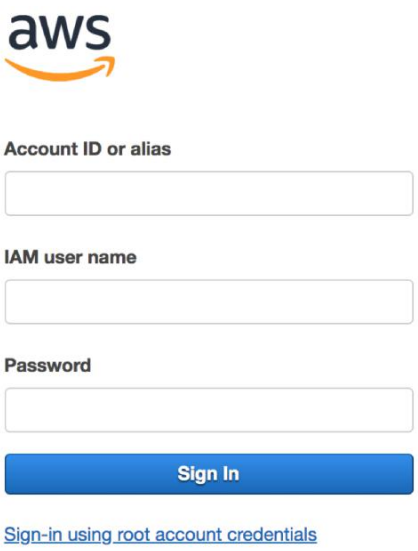
5 Checking out the AWS Management Console

The AWS Management Console allows users to manage Amazon Web Services through a simple and intuitive web-based user interface.

5.1 Log in to the AWS Management Console

Navigate to the IAM user login for the AWS Management Console:
https://signin.aws.amazon.com/console

Enter your AWS Account ID or alias, IAM user name, and Password.



5.2 AWS Management Console

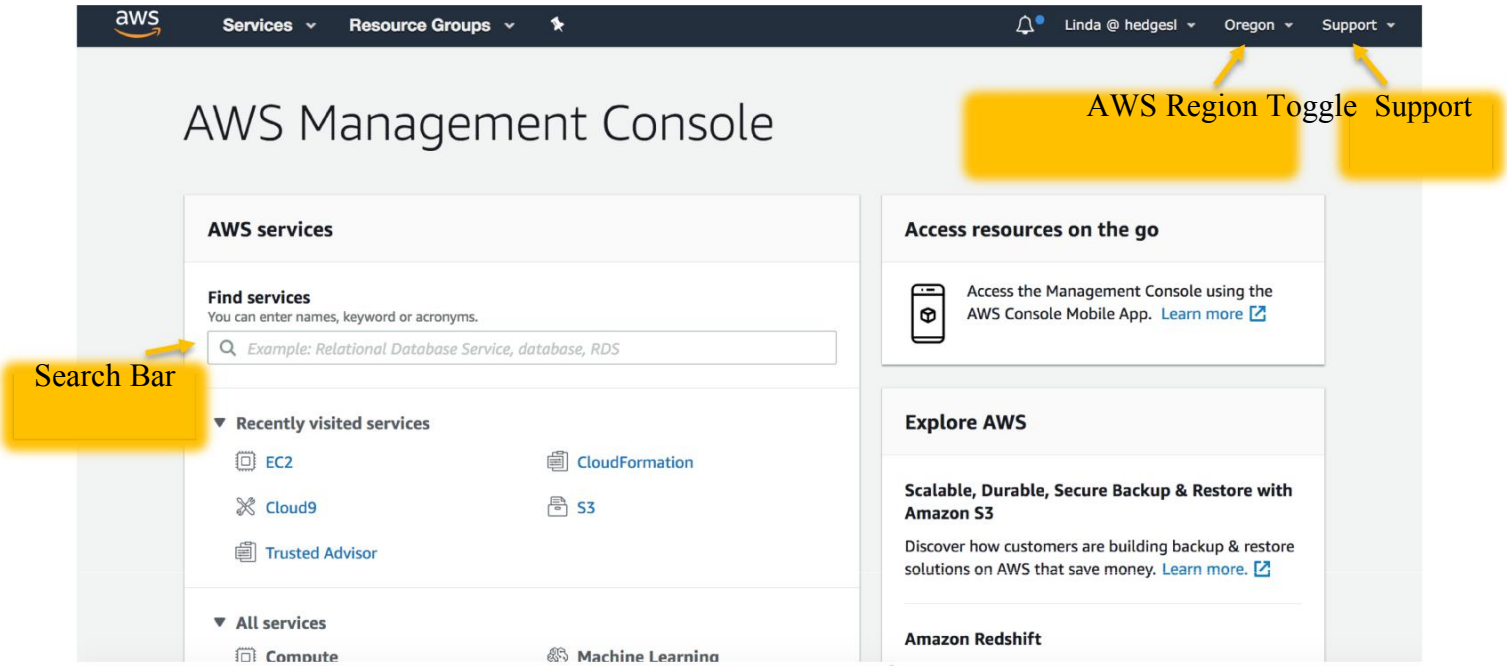
After you log in, you will see the main AWS Management Console. Take a few minutes to check it out. A search bar is located near the upper left. Recently used services are located below that. Your console may look differently depending on the recently used services. A list of all services can be found below the list of recently used services.

A more thorough description of the console is found here:
https://aws.amazon.com/console/

In general, AWS documentation is found here:
https://docs.aws.amazon.com/

Support and documentation are also found in the support pull down menu at the upper right of the management console.

Start by selecting a region from the pull down in the upper right hand corner. For today, select the Oregon region. The Oregon region is also known as US-West-2.



Spend a few minutes checking out AWS Management Console.

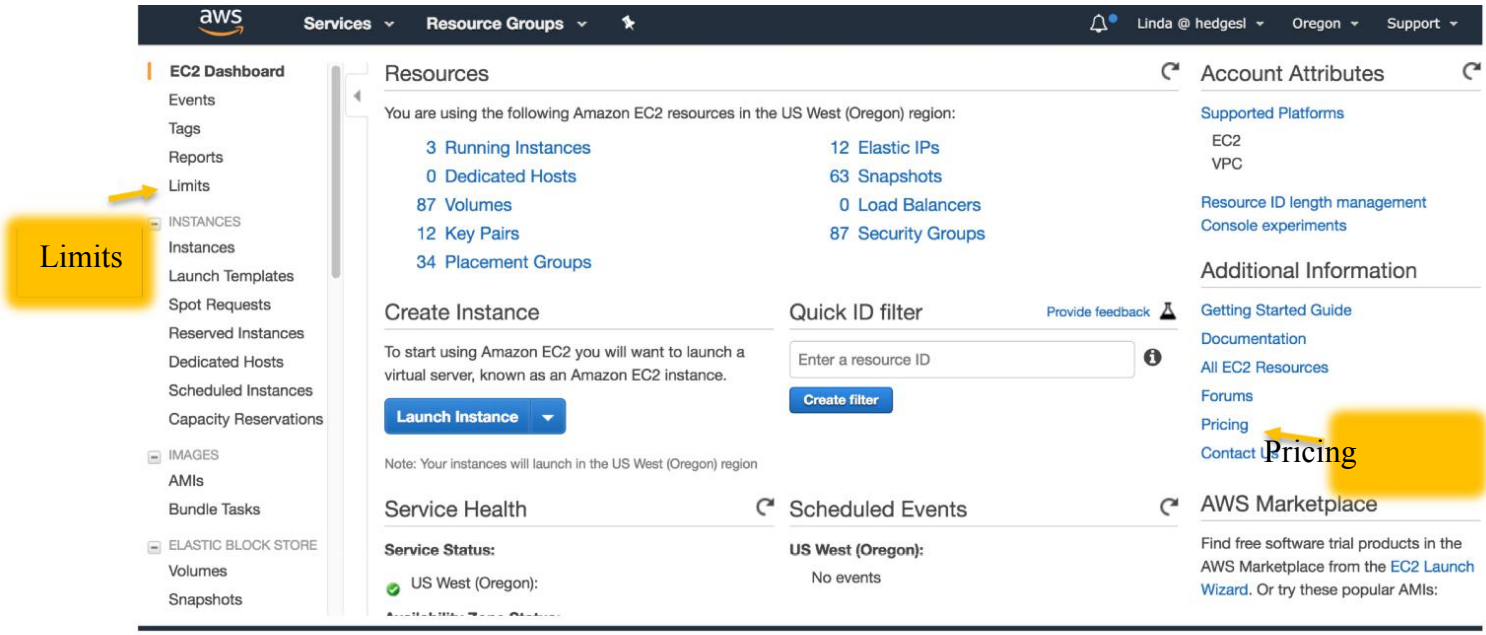
5.3 The EC2 Dashboard

Next, type EC2 into the search bar, click on EC2, and move to the EC2 Dashboard. While instances are generally launched through scripts such as AWS ParallelCluster, they can also be launched directly from the EC2 dashboard. Spend a few minutes looking at the information on this page.

Resource listings and the ability to launch an instance are located in the center of the dashboard while tools and features, such as the saved Amazon Machine Images (AMI's), Storage Volumes, and ssh keys, are accessed from the left-hand side. General information is on the right-hand side, such as documentation and pricing.

Find the EC2 limits link on the left-hand side of the page. EC2 soft limits are in place to protect your account. These limits define the maximum number of resources that can be used, such as the total number of c5.xlarges. Limits can be increased as needed when more are desired.

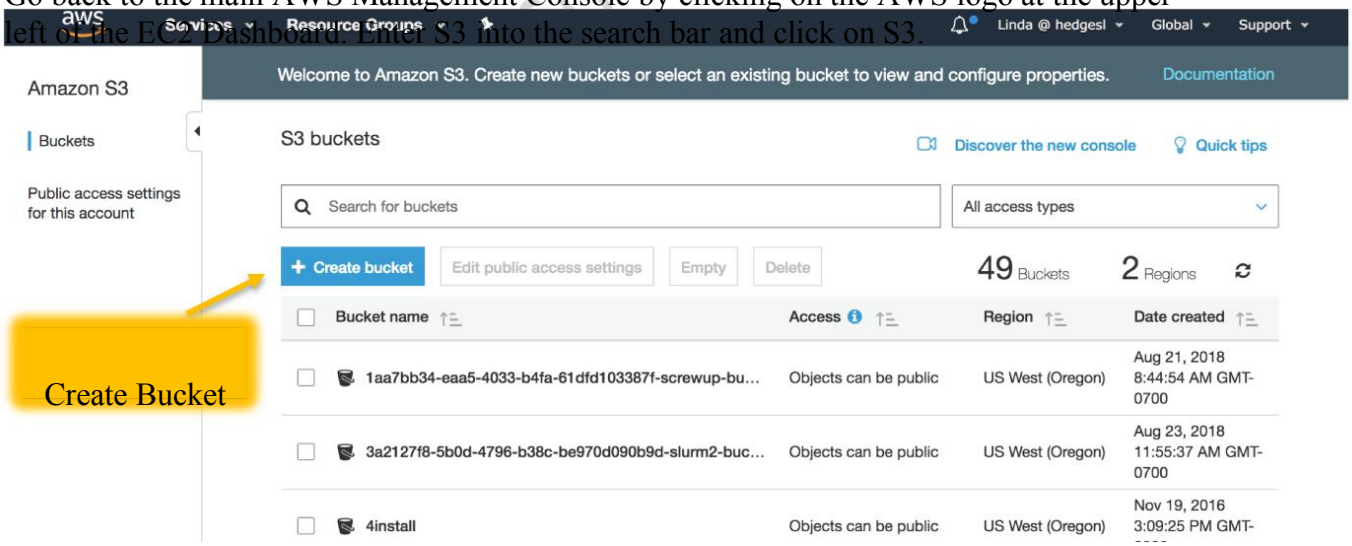
Check out the limits page on the upper left-hand side of the EC2dashboard.



AWS Instance pricing is found on the right-hand side of the page. Instance price changes depending on region. Check out the Pricing page from the link on the right-hand side of the dashboard. Select the "on-demand" link, and then use the pull-down menu for pricing in the desired region.

5.4 S3

Go back to the main AWS Management Console by clicking on the AWS logo at the upper left of the EC2 Dashboard. Enter S3 into the search bar and click on S3.



Click on "Create Bucket" and enter a unique name. Note that this name must be unique in a namespace that is shared by all AWS accounts. Incorporating your initials into the name can help to make it unique.

Once created, select the bucket name and upload a file from your laptop. Spend a few minutes checking out the S3 dashboard. Note the name of the bucket and leave the file and bucket for section 7 of the lab.

5.5 Cloud9

Go back to the main AWS Management Console by clicking on the AWS logo at the upper left of the S3 Dashboard. Enter cloud9 into the search bar and click on cloud9.

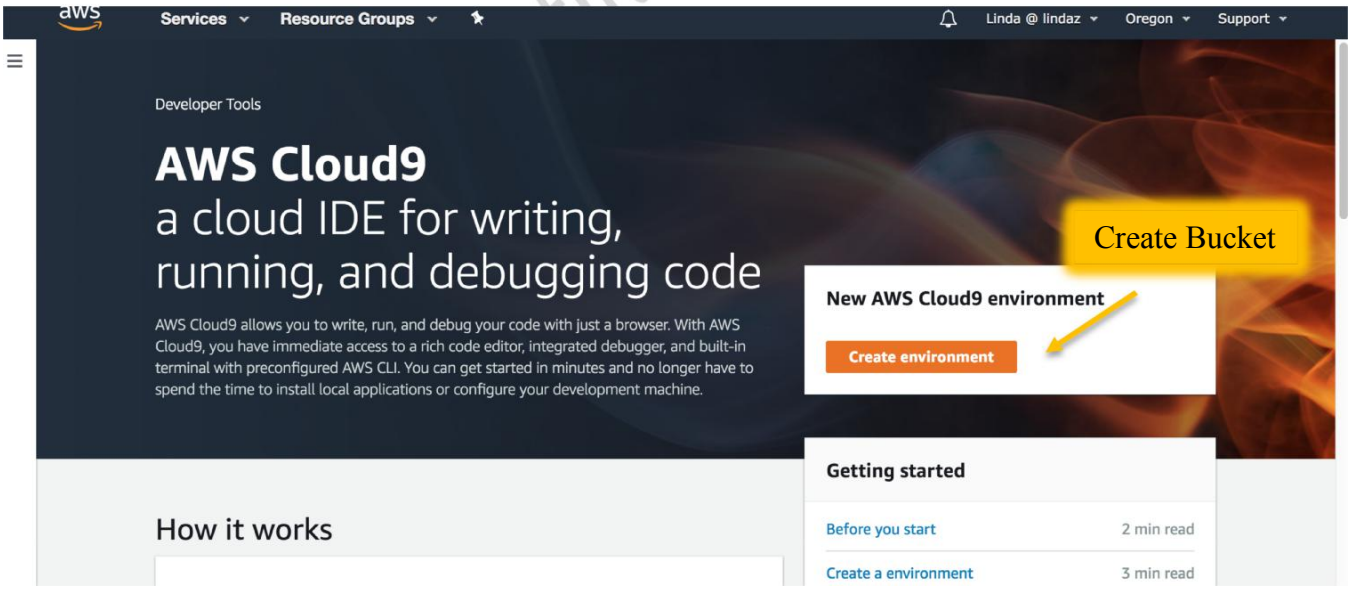
6 AWS Cloud9

This tutorial launches AWS ParallelCluster from the AWS Cloud9 IDE. AWS Cloud9 contains a collection of tools that let you code, build, run, test, debug, and release software in the cloud. The IDE offers support for python, pip, the AWS CLI and provides easy access to AWS resources through the IAM user credentials.

AWS ParallelCluster can also be conveniently launched from the laptop or desk top. AWS ParallelCluster Documentation, Section 10, discusses AWS ParallelCluster and where to find the documentation that explains these steps.

6.1 Launching AWS Cloud9

If not already there, open the AWS Cloud9 console, at https://console.aws.amazon.com/cloud9/

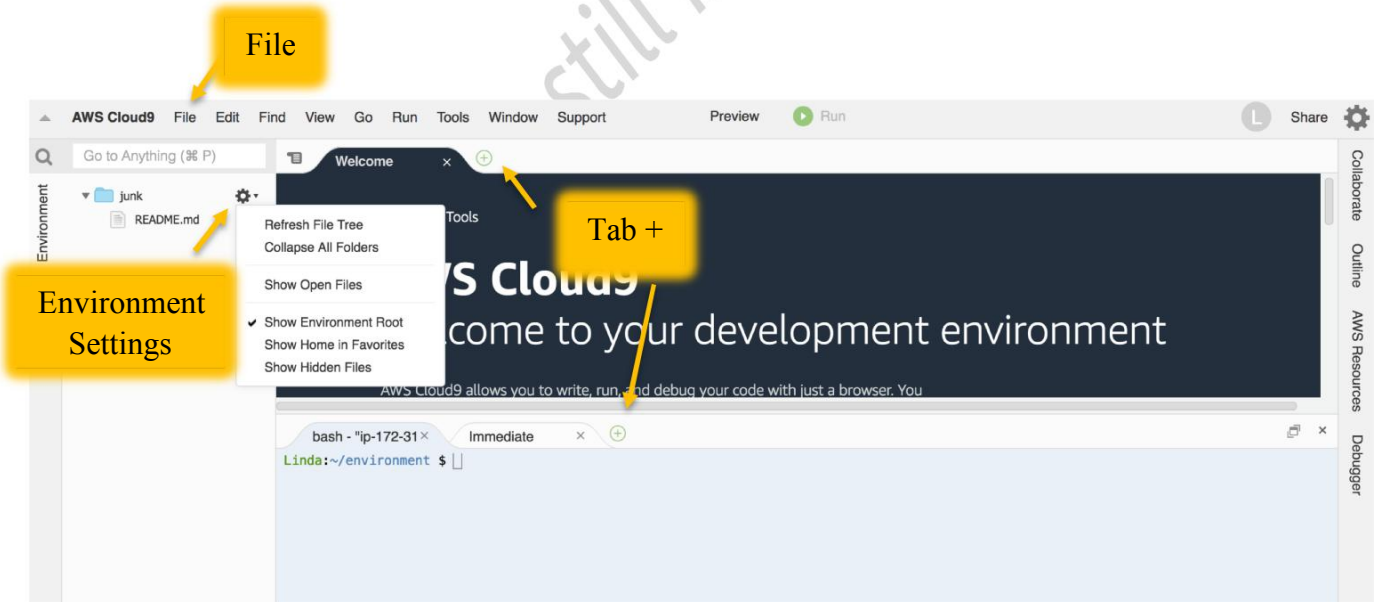


The Steps for launching cloud 9 are as follows:

1. Click on **Create environment**
2. To the right in the top navigation bar, double check that you are in the Oregon AWS Region.
3. Type a **Name** for your environment on the **Name environment** page.
4. To add a description to your environment, type it in the **Description** box.
5. Choose **Next step**.
6. On the **Configure settings** page, for **Environment type**, choose **Create a new instance for environment (EC2)**.
7. For **Instance type**, choose the default choice: t2.micro or t3.micro.
8. Note that for **Cost-saving setting**, the amount of time until AWS Cloud9 shuts down the Amazon EC2 instance is selectable. Leave the default choice.
9. Choose **Next step**.
10. Click on **Create Environment** and wait a few minutes for the environment to launch and set up**.**
11. Spend a few minutes checking out the AWS Cloud9 IDE Dashboard.

6.2 Upload a File from Your Laptop

The steps to upload a file to the Cloud9 IDE are as follows.



Add a file to the AWS Cloud9 environment:

1. From the AWS Cloud9 Dashboard click on the Environment settings gear in the upper left corner.
2. Select **Show Home in Favorites** and **Show Hidden Files.**
3. Select **Upload local files …** from the **File** pull down menu.
4. Upload a file.

Now that the file has been uploaded:

5. Find the files in the environment sidebar, right click and delete the files.

6.3 Edit a File

Cloud9 includes a simple text editor:

1. From the AWS Cloud9 Dashboard select **New File** from the **File** pull down menu.
2. Select a location from inside the **Untilted1** panel, add a few words of encouragement such as “this is great!”, and select **Save As** from the **File** pull down menu.
3. Give the file the name **goodjob.txt** and select **Save.**
4. Note the location of the file from the Environment sidebar.

6.4 Open a terminal window

The Cloud9 IDE allows for the creation of terminal windows:

1. Select one of the tab “+” icons
2. Select **New Terminal**
3. Explore the new terminal with a few linux commands: pwd, ls, etc.
4. Move to the home directory by typing **cd ~** in the terminal window.

7 The AWS Command Line Interface, AWS CLI

The AWS Command Line Interface (CLI) is a tool to manage your AWS services. It easily integrates with bash scripts. Cloud9 installs the AWS CLI and configures programmatic IAM credentials by default. More information on the AWS CLI is found at:
https://aws.amazon.com/cli/

7.1 Working with S3

The AWS CLI is commonly used to move files from S3 to an EC2 instance. Look for the S3 bucket and file created earlier from one of the terminal windows:

```
$ aws s3 ls

$ aws s3 ls s3://bucketname/
```

Copy the file created in section 6 to your s3 bucket:

```
$ aws s3 cp goodjob.txt s3://bucketname/
```

7.2 Creating an SSH Key

Logging on to an EC2 instance requires the use of an SSH Key for passwordless login. The SSH Key can be created from the AWS Management Console, EC2 Dashboard, or through the AWS CLI.

1. Create an SSH Key, using the name “lab” + “your initials” + “a unique number”. For example, “lablsh60”

```
$ aws ec2 create-key-pair --key-name labXXX## --query KeyMaterial --output text >
~/.ssh/id_rsa
```

2. From a terminal window, move to the .ssh subdirectory:

```
$ cd ~/.ssh
```

3. Modify the id_rsa file permissions:

```
$ chmod 600 id_rsa
```

4. Move back to the home directory:

```
$ cd ~
```

The ssh key is now ready to be used.

8 Install and Set up AWS ParallelCluster

AWS ParallelCluster enables you to quickly build an HPC compute environment in AWS. It automatically sets up the required compute resources, a shared filesystem, and offers a variety of batch schedulers such as AWS Batch, SGE, Torque, and Slurm.

8.1 Install AWS ParallelCluster

Python and PIP (Python package management tool) are already installed in the Cloud9 environment. Use the pip install command to install AWS ParallelCluster:

```
$ sudo pip install aws-parallelcluster
```

8.2 Initialize AWS ParallelCluster

Next, configure AWS ParallelCluster.

Enter **pcluster configure** in a terminal window and walk through the configure menu. Accept the defaults except where text is added in bold below.

```
$ pcluster configure
Cluster Template [default]:
AWS Access Key ID []:
AWS Secret Access Key ID []:
Acceptable Values for AWS Region ID:
    ap-south-1
    ...
    us-west-2
AWS Region ID [us-east-1]:us-west-2
VPC Name [public]:
Acceptable Values for Key Name:
  keypair1
  keypair-test
  production-key
Key Name []:labXXX##
Acceptable Values for VPC ID:
  vpc-blk4942q
```

```
VPC ID []:Add_the_VPC_ID_suggested
Acceptable Values for Master Subnet ID:
  subnet-9k283a6g
Master Subnet ID []:Select_a_Subnet_ID_offered
```

8.3 Modify the default AWS ParallelCluster Config File

Upon initial configuration, AWS ParallelCluster places a default configuration file, "config", into the ~/.parallelcluster directory. This config files creates a simple cluster with the minimum required information. In this section, the config file is replaced, to create a config file that creates a cluster typical of one used to run "tightly coupled" HPC applications.

1. Start by renaming the config file created in the last section. From a terminal window, move the the .parallelcluster directory and rename the config file.

   ```
   $cd ~/.parallelcluster
   $mv config config_simple
   ```

2. Using the Cloud9 text editor, find the file called config_simple in the left hand environment panel, and open the file with a right click. Next, move to the tab with the open file (called config_simple). Inspect the file and leave it open for step 4. Notice the location of the vpc and subnet id's.

3. From a terminal window in the cloud9 ide, move to the ~/.parallelcluster location and copy the following config file to the ~/.parallelcluster location using the wget command:

```
$ cd ~/.parallelcluster
$ wget https://s3-us-west-2.amazonaws.com/kevin-public/config_start
```

4. Using the Cloud9 IDE text editor (as described in step 1), open **config_start**. The file, **config_simple**, should already be open. Modify **config_start** with the (1) vpc id, (2) subnet id found in the **config_simple** file, and (3) the keyname **labXXX##** found in the config_simple file. Copy and paste the entire lines, to ensure that the names are copied exactly. Save the **config_start** file with the new name **config** in the ~/.parallelcluster directory. It is okay to overwrite an existing config file, if it exists.
5. Double check that the ~/.parallelcluster config file is the merged file that includes both specification of master and compute instance types, as well as your vpc, subnet id's and the correct keyname.
6. Open the config file again, and make some further changes. (1) Currently, we are requesting a 20GB empty EBS volume for the cluster. Change this to a 400GB EBS volume (of type gp2) from the EBS snapshot with snapshot_id snap-0388ba4ffebb19c98 . (2) Change the maximum number of compute instances to 4. You'll probably want to get to know the documentation at https://aws-parallelcluster.readthedocs.io/en/latest/configuration.html to solve this. When you are

finished, compare your result to the solution in the Section 10.1 of this document, then save the config file, and proceed.

9 Working with AWS ParallelCluster

AWS ParallelCluster is now installed and configured. In day-to-day application, it is possible to create many clusters quickly. It is common to create one cluster per application, such as one cluster for LS-Dyna and one cluster for ANSYS Fluent. Clusters can also be created for different needs, such as one for production and one for a POC.

9.1 Launch an AWS ParallelCluster

Issue the following command to create a cluster, pick a cluster name as desired:

$ pcluster create *UniqueClusterNam*e

The launch process takes about 10+ minutes.
When the cluster is complete, the following information is displayed.

```
Beginning cluster creation for cluster: UniqueClusterName
Creating stack named: parallelcluster-UniqueClusterName
Status: parallelcluster-round2 - CREATE_COMPLETE
MasterPublicIP: 54.148.206.47
ClusterUser: ec2-user
MasterPrivateIP: 172.31.40.23
```

9.2 While waiting for the cluster to come up, turn to the documentation in Section 10, AWS ParallelCluster Documentation

Note how the config file calls the post install script. While the cluster is being created, review the AWS Parallel Cluster documentation in Section 10.

9.3 Logging on to the AWS ParallelCluster Head node

Log into the new ParallelCluster with the following command:
`$ pcluster ssh *UniqueClusterName*`

If you have forgotten the unique cluster name, it can be retrieved with this command:
`$pcluster list`

The EC2 instance asks for confirmation of the ssh login the first time you log onto the instance. Answer with "**yes**".
`Are you sure you want to continue connecting (yes/no)? **yes**`

The ls command will reveal the files added by the post_install script:

```
[ec2-user@ip-172-31-40-92 ~]$ ls
bin  hpcg.qsub
[ec2-user@ip-172-31-40-92 ~]$ ls bin
```

```
xhpcg
```

9.4 Running your first job: Launch the HPCG Benchmark

The qsub command is used to launch the hpcg benchmark. A job submittal file is used and was created by the post install script. Check out the hpcg.qsub file:

```
[ec2-user@ip-172-31-40-93 ~]$ cat hpcg.qsub
#!/bin/bash
#$ -pe mpi 4
#$ -j Y

/usr/lib64/openmpi/bin/mpirun -np 4 ~/bin/xhpcg --nx=32 --ny=32 --nz=32 --rt=20
```

Submit the job by entering the command: qsub hpcg.qsub.

```
[ec2-user@ip-172-31-40-93 ~]$ qsub hpcg.qsub
Your job 1 ("hpcg.qsub") has been submitted
```

9.5 The SGE Scheduler

The SGE scheduler is managed by the user. Useful SGE commands include **qhost** and **qstat**.

The command, **qstat:** The **qstat** command gives information on the submitted jobs, including the state of the job. Initially the job will be in the "qw" state (queue, wait). This is followed by the "r" state (run). The job id number is provided. The task is removed from the queue when a job is completed.

```
[ec2-user@ip-172-31-40-93 ~]$ qstat
job-ID  prior   name       user         state submit/start at     queue                          slots
ja-task-ID
------------------------------------------------------------------------------------------------------------
      1 0.55500 hpcg.qsub  ec2-user     r     12/08/2018 00:49:37 all.q@ip-172-31-45-182.us-west     4
```

The command, **qhost:** The **qhost** command lists the compute node status and id's. Note that the ip address can be used to log onto the compute nodes.

```
[ec2-user@ip-172-31-40-93 ~]$ qhost
HOSTNAME                ARCH         NCPU NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------------------
global                  -               -    -    -    -     -       -       -       -       -
ip-172-31-44-191        lx-amd64        2    1    2    2  0.24    7.4G  328.6M     0.0     0.0
ip-172-31-45-182        lx-amd64        2    1    2    2  0.22    7.4G  328.7M     0.0     0.0
```

Use the Linux SGE man pages to learn more about the SGE scheduler.
You can use **qdel** to delete a job from the queue.

9.6 Autoscaling the ParallelCluster

In 9.5 we submitted a single job that required 4 cores, and our cluster had exactly 4 cores available, so it was the perfect size. But oftentimes you will put much more work in the queue than can be computed all at once. If you wish, you can allow the cluster to adjust its size accordingly: "growing" more compute nodes when there are lots of jobs, and "terminating" the compute nodes when they are idle.

We will run a different application, a physics HPC code called FEFF that calculates the absorption of light by materials.

```
cd ~/FEFFruns
```

There is a mess of files here that you don't need to fully understand, but essentially each "*.cif" file contains the atomic structure of a material. We will simply run the FEFF application once for each of the ~20 materials[1]. The key thing is that this will put a lot of work in the job queue. In response to that abundance of work, the cluster will scale and create new EC2 compute instances.

```
./ task.sh
```

You can use qstat and qhost again to see what's happening. After a few minutes, check the EC2 console if you see new instances being created. (If not, wait a little while longer.)

Check the qhost command again to see the new hosts appear in the cluster. Now that we have gone from 2 to 4 compute nodes, we will crunch through the backlog of jobs twice as fast as before. Nice! Since we set a minimum of 2 and maximum of 4 compute nodes, the autoscaling mechanism can only operate between those boundaries. That is, we cannot process faster than with 4 nodes; and we can never scale the capacity down below 2 nodes, even if there is no work left to do. It is very common to instead use a minimum of 0 nodes, so that the cost of the cluster goes to a minimum when there is no work left in the job queue; and to have a higher maximum, to allow more speedup. The total cost will be in the volume of core hours used, so it isn't more expensive to run all the members of a statistical ensemble (e.g. weather forecasting) or high-throughput case all at the same time.

We've seen all we want to see here, so you're welcome to delete any remaining jobs from the queue if you like, or you can just leave them be – they'll all get done in a few minutes anyway. (If you waited long enough, you'd first see all the jobs finish, and finally some time after that you'd see the number of nodes drop to the minimum again.)

BONUS POINTS: If you were thinking to yourself, "hmm, wouldn't this be a perfect candidate for AWS Batch?", then double ice cream for you tonight.

9.7 Benchmarking on the ParallelCluster

People often want to benchmark their application. The benchmark tells us how long it takes to finish a given job on the chosen compute platform. This is useful for a few reasons. First, you can compare different platforms that way, for example maybe the AWS cluster is 20% faster than your on-premise cluster if it's an older cluster. Second, since on AWS you pay exactly for the time you use the EC2 instances, the benchmark also tells you the cost of each simulation. For example, if each instance costs $1 per hour, and your job takes 1.5 hours on 8 instances, then the cost per job is roughly $1 x 1.5 x 8 = $12. ("Roughly" because there is also the cost of the head node and some (usually very minor) costs related to storage etc.) Finally, during the benchmark you typically try out several configurations, and the results allow you to choose the configuration that works best for you.

You repeat the same job on clusters of increasing size and see how much faster it gets. For example: (1 node, 8 minutes), (2 nodes, 4 minutes), (4 nodes, 2:10 minutes), (8 nodes, 1:12 minutes). Normally it gets faster as you use more cores/nodes. But you'll already notice from this made-up example that the speedup isn't completely linear. There are several reasons for this, including Amdahl's Law (every application contains non-parallellizable code that limits the theoretically achievable speedup) and infrastructure limitations (the speed of light, network latency, memory access etc. - keep speedup below the theoretical limit).

[1] Depending on the version of this tutorial you're using, you may see ~20 cases here or ~120.

Now, a note: Because this is a tutorial and we wanted to keep it as cheap as possible, our cluster is comprised of tiny c4.xlarge instances. These won't do very well on a parallel MPI benchmark; the MPI communications will overwhelm them pretty quickly. If you want to do this benchmark exercise more properly, you can now upgrade your cluster using the "update" command (https://aws-parallelcluster.readthedocs.io/en/latest/commands.html#update).
To do that, you edit the "config" file in the cloud9 console, and change the line
`compute_instance_type = c4.8xlarge`
Then you do the command
`pcluster update UniqueClusterName`
And then it will take a few minutes for your old c4.xlarge compute instances to be terminated and replaced by new c4.8xlarge instances. After a few minutes, that will be reflected by the **qhost** command on your cluster.
But if you don't want to take the time for this, then you can do the test on the c4.xlarge instances. It just won't be very good.

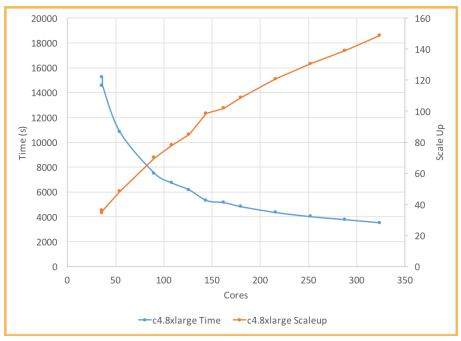
`cd /codes/WRF-benchmarks/CONUS12`
This directory contains a weather forecast. We will use it benchmark the WRF weather model on our cluster. Adapt the job file to run the forecast multiple times:
Run on 4 nodes (edit wrfjob.sh ; `qsub wrfjob.sh`)
Run on 2 nodes (edit wrfjob.sh ; `qsub wrfjob.sh`)
Run on 1 node (edit wrfjob.sh ; `qsub wrfjob.sh`)
Run on 1 core (edit wrfjob.sh ; `qsub wrfjob.sh`)

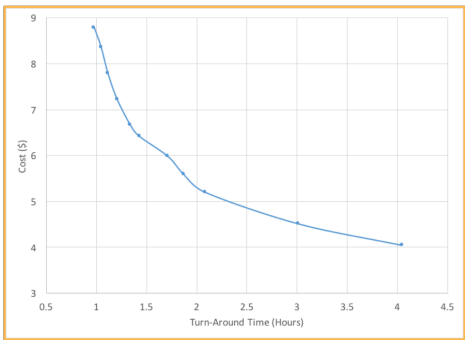
(Pro tip: you will find the best performance if you combine MPI and OpenMP. You can use OMP_NUM_THREADS=9 and use 2 MPI threads per c4.8xlarge instance (=1 per socket). This is the most efficient balance between CPU utilization and MPI communication overhead.)

Each time, write down the runtime it took to complete the compute. The output should tell you this as "compute 87.34252s" or so. (The slowest one will take about 8 minutes.) Make a table or plot of the result, with the number of cores on the x-axis, and 1 over the runtime on the y-axis. This is what's called a "speedup curve". It tells you how much the application speeds up as you increase the number of nodes. (See example below.)

Now change the plot or table, and instead plot cost in $ on the x-axis and time on the y-axis. This is the same information as in the previous graph, but it shifts the attention away from infrastructure to real-life decisions. "How fast do I want this result, and how much am I willing to spend to get it that fast?" For example, what point on this graph would you choose to do non-urgent research, and what point would you choose for forecasts where human lives depend on it? (Like knowing when the wind is about to change direction and push a forest fire towards firefighters.) Example below:

Another note: Our test is a very small weather forecast, and it doesn't have much predictive power for much larger weather forecasts. Always try to run a benchmark that's representative of what you're interested in. Don't try to stretch tiny benchmarks out to huge clusters: there's just "not enough meat on the bone" and it's not a very relevant exercise.





TBD: Visualize the weather forecast on a map.
Make sure XWindows is available on the machine you're connecting from (Cloud9? Or laptop?)
Install on cluster head node packages

```
sudo yum install ncview xterm xauth
```

Open a new connection to the head node of the cluster.

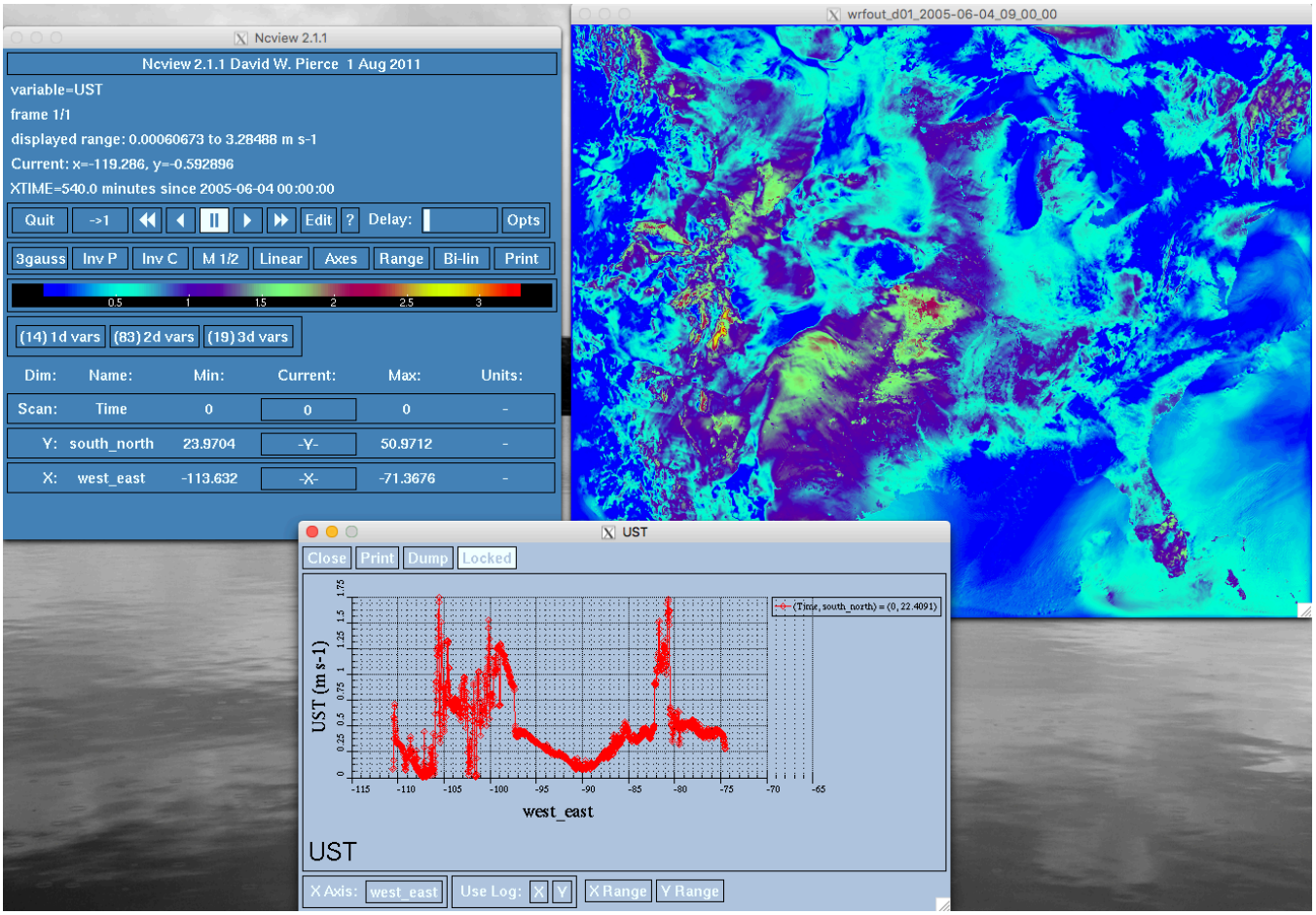
```
$ ssh -X -A -i key_1.pem ec2-user@<ip_address>

[ec2-user@ip-address]$ xterm
```

Plot the WRF output file in the ncview application

```
[ec2-user@ip-address]$ ncview wrfout_d01_2005-06-04_09_00_00
```

You'll see something similar to this:



9.8 Delete the AWS ParallelCluster

Please remember to save valuable software setups and data before deleting any ephemeral cluster. For example, data can be saved directly to S3, snapshots can be created from EBS volumes, such as of the shared file directory (created by default with AWS ParallelCluster), and Amazon Machine Images, AMI's, can be made of instance set ups. For this lab, there is no need to save any data.

From a terminal window in the IDE. List the running ParallelClusters:

```
$pcluster list
```

A listing of all clusters associated with the AWS credentials will be provided.

Delete the ParallelCluster:

```
$pcluster delete unique_cluster_name
```

Clusters take several minutes to delete as the entire AWS ParallelCluster is removed from the AWS account.

10 AWS ParallelCluster Documentation

Notes on AWS ParallelCluster are provided here.

10.1 Notes on the config file

Note some features of the config file.

- The instance type is set to the c5.xlarge for both the master node and compute nodes.

```
compute_instance_type = c5.xlarge
master_instance_type = c5.xlarge
```

- A placement group is set with the “DYNAMIC” option to allow for low latency between compute nodes.

```
placement_group = DYNAMIC
```

- The max cluster queue size and the initial cluster queue size are set to two compute nodes/instances.

```
max_queue_size = 4 initial_queue_size = 2
```

- The following line tells the scheduler that hyperthreading will be disabled and to schedule by cores rather than hyperthreads.

```
extra_json = { "cluster" : { "cfn_scheduler_slots" : "cores" } }
```

- The following line calls a post install script. The post install script also allows for the master and compute nodes to be modified, such as to disable hyperthreading as well as to add libraries.

```
post_install = https://s3-us-west-2.amazonaws.com/kevin-public/cluster-setup-
script.sh
```

The entire Lab config file is listed here:

```
[global]
update_check = true
sanity_check = true
cluster_template = default

[aws]
aws_region_name = us-west-2

[cluster default]
vpc_settings = public
ebs_settings = custom
scaling_settings = custom
key_name = labXXX###
compute_instance_type = c4.xlarge
master_instance_type = c4.xlarge
cluster_type = ondemand
placement_group = DYNAMIC
max_queue_size = 4
initial_queue_size = 2
extra_json = { "cluster" : { "cfn_scheduler_slots" : "cores" } }
```

```
post_install = https://s3-us-west-2.amazonaws.com/kevin-public/cluster-setup-
script.sh
shared_dir = /codes

[scaling custom]
scaledown_idletime = 60

[vpc public]
master_subnet_id = subnet-XXXXXX
vpc_id = vpc-XXXXXX

[ebs custom]
volume_type = gp2
volume_size = 400
ebs_snapshot_id = snap-0388ba4ffebb19c98

[aliases]
ssh = ssh {CFN_USER}@{MASTER_IP} {ARGS}
```

10.2 Notes on the post install script

The post install script is found here:
`$ wget https://s3-us-west-2.amazonaws.com/kevin-public/cluster-setup-script.sh`

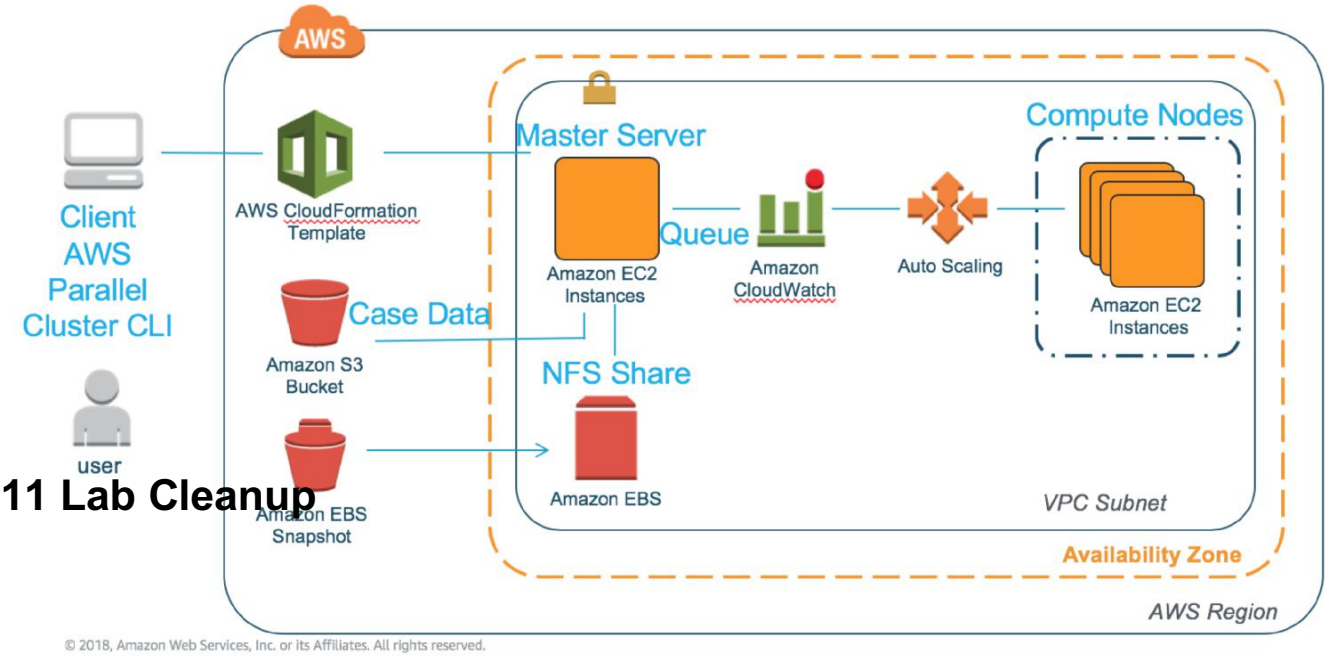
It is called by the AWS ParallelCluster config file, and is listed here.
It writes a log to /var/log/postinstall.log.

10.3 AWS ParallelCluster "Read the Docs"

AWS ParallelCluster documentation is found here:
https://aws-parallelcluster.readthedocs.io/en/latest/

10.4 The AWS ParallelCluster Architecture

The AWS ParallelCluster architecture is represented here:



11 Lab Cleanup

Delete the file and bucket created on S3 with the AWS CLI.

```
aws s3 rm s3://bucketname/goodjob.txt

aws s3 rb s3://bucketname
```

AWS ParallelCluster is best deleted from the Cloud9 IDE as follows:

Delete the parallelcluster:
`$pcluster delete unique_cluster_name`

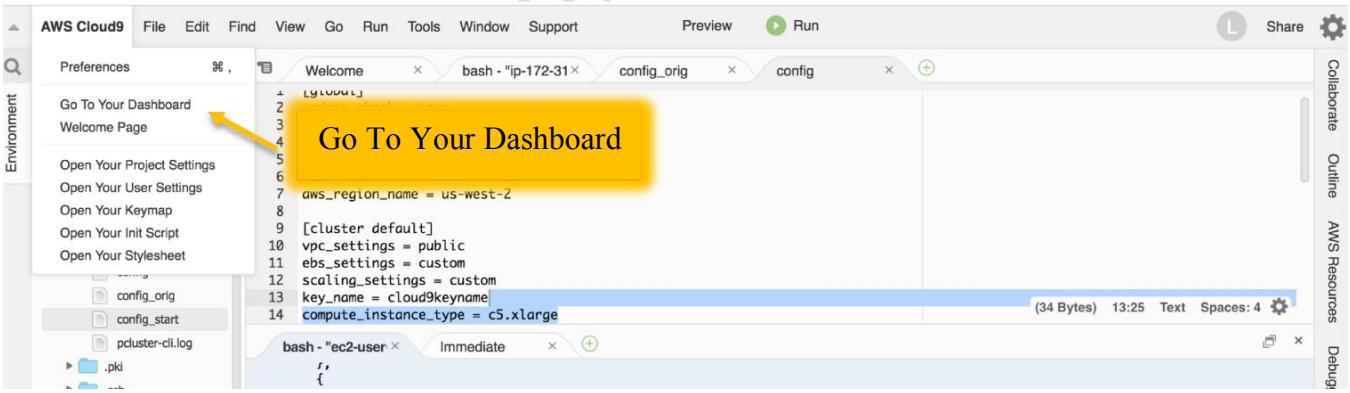
It can also be deleted from the AWS Management Console by deleting the cloudformation stack tied to the cluster. The cluster itself is not tied to the Cloud9 IDE or the location of the terminal or machine from which it is launched. Any AWS ParallelCluster install, tied to the AWS account, will view the clusters through the pcluster commands, if AWS ParallelCluster is installed. For example, the following AWS ParallelCluster command lists the name of all running clusters:

```
$pcluster list
```

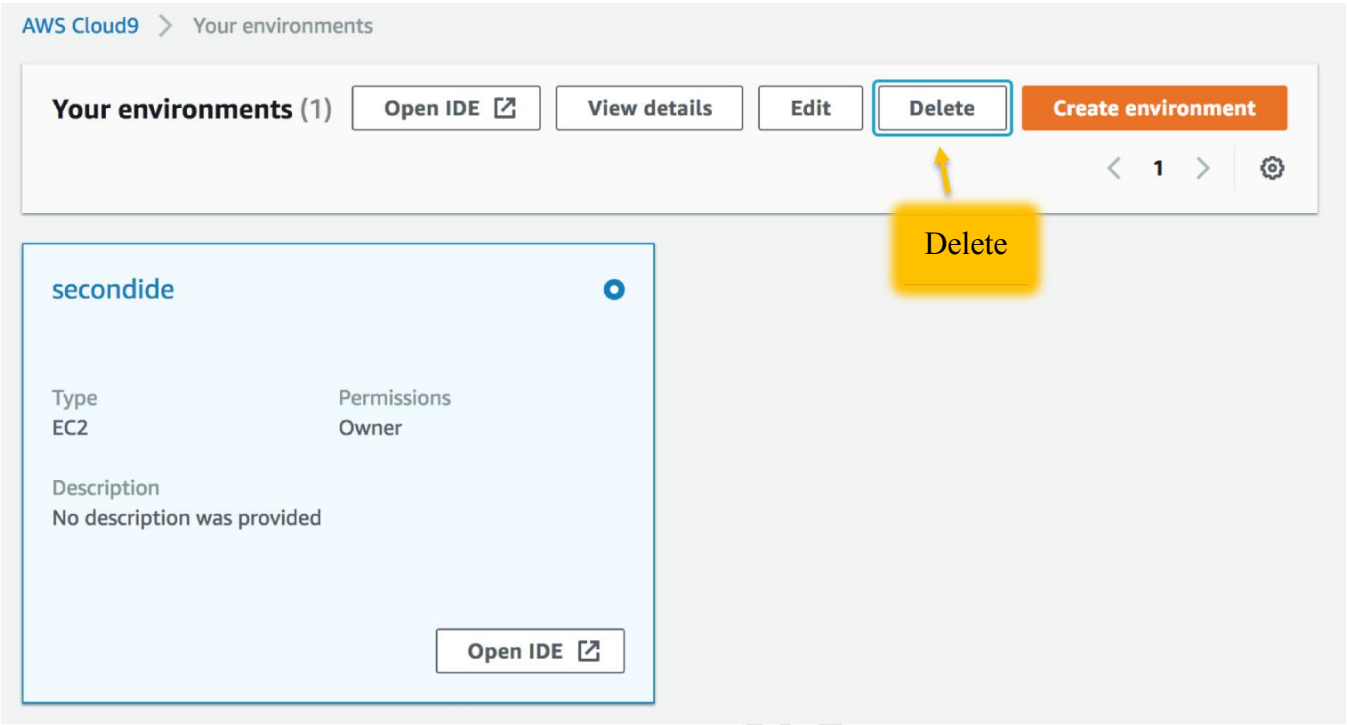
Once at the cloud9 IDE, delete the keypair from AWS before deleting the cloud9 IDE. The private key is required to reach instances launched with that key. Lost private keys cannot be recovered and the public portion of the key remains listed in the AWS console until it is deleted.

Delete the key:
`$aws ec2 delete-key-pair --key-name labXXX###`

Finally, remove the cloud9 IDE by selecting the "**Go To Your Dashboard**" item from the **AWS Cloud9** pulldown menu.



Click Delete in the upper right-hand corner of the Cloud9 Dashboard, and enter the word "**Delete**" into the text box.



If desired, return to the EC2 Dashboard to confirm that the Cloud9 instance and all instances related to AWS ParallelCluster are deleted.