1. (10 pts) Log in to the XSEDE cluster as shown in class. Run a sequential "hello world" as described in class. Have the program print hello <user-name> where <user-name> is your XSEDE user id. Turn in your code, Unix shell commands, and the program output.

|  |  |
| --- | --- |
| Code | #include <stdio.h>  #include <mpi.h>    int main(int argc, char \*argv[]) {  int namelen;  char username[MPI\_MAX\_PROCESSOR\_NAME];    MPI\_Init(&argc, &argv);  MPI\_Get\_processor\_name(username, &namelen);    printf(“Hello %s\n", username);  MPI\_Finalize();  } |
| job\_script | #!/bin/bash  #PBS -l nodes=1:ppn=20  #PBS -l walltime=00:01:00  #PBS -N hello\_mpi  #PBS -o hello\_mpi.out  #PBS -e hello\_mpi.err  #PBS -q workq  #PBS -A loni\_droz50ksv  #PBS -m e  #PBS -M mst039  mpirun -machinefile $PBS\_NODEFILE -np 1 ./hello\_mpi |
| Commands  Using LONI | mpicc -o hello\_mpi hello\_mpi.c  qsub job\_script |
| Output | Hello qb024 |

1. Write a **sequential** C code program that sums a vector of values **vect1[i]**. The result should be placed in a variable called **totsum**. Initialize your vector to integers such that **vect1[i] = i+1**. When you’re certain the code compiles and runs correctly, add timing commands and report your run times. Turn in your code, Unix shell commands, and the program output.

|  |  |
| --- | --- |
| Code | #include <stdio.h>  #include <mpi.h>  #define ARRAYSIZE 10  int vect1[ARRAYSIZE];  int main(int argc, char \*argv[]) {  double start\_time, end\_time, exe\_time;  MPI\_Init(&argc, &argv);  start\_time = MPI\_Wtime();  int totsum=0;  int i=0;  for(i = 0; i < ARRAYSIZE; i++ )  {  vect1[i] = i + 1;  }  for(i = 0; i < ARRAYSIZE; i++ )  {  totsum = totsum + vect1[i];  }  end\_time = MPI\_Wtime();  exe\_time = end\_time - start\_time;  MPI\_Finalize();  printf("Sum is %d. Took %d sec.", totsum, exe\_time);  } |
| job\_script | #!/bin/bash  #PBS -l nodes=1:ppn=20  #PBS -l walltime=00:01:00  #PBS -N sum\_seq  #PBS -o sum\_seq.out  #PBS -e sum\_seq.err  #PBS -q workq  #PBS -A loni\_droz50ksv  #PBS -m e  #PBS -M mst039  mpirun -machinefile $PBS\_NODEFILE -np 1 ./sum\_seq |
| Commands  Using LONI | mpicc -o sum\_seq sum\_seq.c  qsub job\_script |
| Output | Sum is 55. Took 0.0003 sec. |

1. Run hello world from <your\_username>, based on the code from lecture, on the XSEDE bridges cluster using MPI. Run on 4 and 8 threads. Report your run times. Add timing commands to the MPI code and report your run times. Turn in your code, Unix shell commands, and the program output.

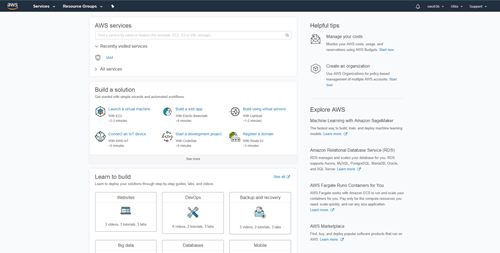
|  |  |
| --- | --- |
| Code | #include <stdio.h>  #include <mpi.h>  int main(int argc, char \*argv[]) {  double start\_time, end\_time, exe\_time;  start\_time = MPI\_Wtime();  int numprocs, rank, namelen;  char username[MPI\_MAX\_PROCESSOR\_NAME];  MPI\_Init(&argc, &argv);  MPI\_Get\_processor\_name(proc\_name, &namelen);  MPI\_Finalize();  end\_time = MPI\_Wtime();  exe\_time = end\_time - start\_time;  printf(“Hello %s. Took %d sec.\n", username, exe\_time);  } |
| job\_script  Needs to be changed for different numbers of nodes | #!/bin/bash  #PBS -l nodes=1:ppn=20  #PBS -l walltime=00:01:00  #PBS -N hello\_mpi  #PBS -o hello\_mpi.out  #PBS -e hello\_mpi.err  #PBS -q workq  #PBS -A loni\_droz50ksv  #PBS -m e  #PBS -M mst039  mpirun -machinefile $PBS\_NODEFILE -np 4 ./hello\_mpi  #!/bin/bash  #PBS -l nodes=1:ppn=20  #PBS -l walltime=00:01:00  #PBS -N hello\_mpi  #PBS -o hello\_mpi.out  #PBS -e hello\_mpi.err  #PBS -q workq  #PBS -A loni\_droz50ksv  #PBS -m e  #PBS -M mst039  mpirun -machinefile $PBS\_NODEFILE -np 8 ./hello\_mpi |
| Commands  Using LONI | mpicc -o hello\_mpi hello\_mpi.c  qsub job\_script  //change number of nodes in hello\_mpi.c and job\_script  mpicc -o hello\_mpi hello\_mpi.c  qsub job\_script |
| Output | Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec.  Hello qb191. Took 0 sec. |

1. Write MPI code to sum a vector of values **vect1[i]**. The result should be placed in a variable called **totsum**. Initialize your vector to integers such that **vect1[i] = i+1**. Your vector should be split among the available processes, and each process should add its own components. Use an MPI reduction operator to sum the results on the master process. Your code should work on any number of processes. Compute the runtime of your code. Run with a different number of threads and different length vectors. Turn in your code, Unix shell commands, and the program output. Give advice to someone running this code about how they should parallelize it. How many threads is optimal? How large does your vector need to be to see speedup?

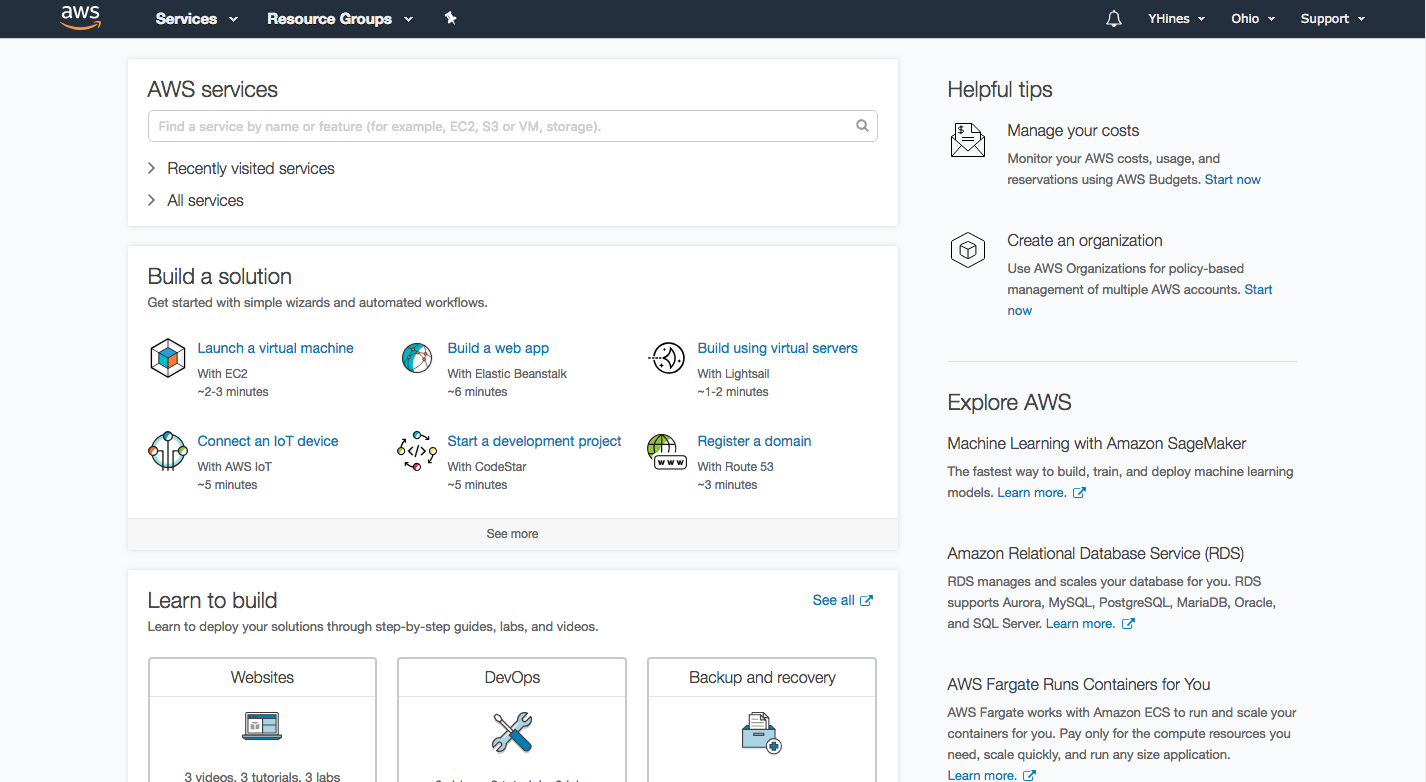
|  |  |
| --- | --- |
| Code | #include <stdio.h>  #include <mpi.h>  #define ARRAYSIZE 10  int vect1[ARRAYSIZE];  int main(int argc, char \*argv[]) {  double start\_time, end\_time, exe\_time;  MPI\_Init(&argc, &argv);  start\_time = MPI\_Wtime();  int totsum, localsum, numprocs, rank;  MPI\_Status status;  int i=0;  for(i = 0; i < ARRAYSIZE; i++ ) {  vect1[i] = i + 1;  }  MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  int delta = ARRAYSIZE / numprocs;  if(rank == 0) {  int count = 0;  int i=0;  for (i = 0; i < numprocs; i++) {  int \*buff = vect1 + count;  count += delta;  MPI\_Send(buff, delta, MPI\_INT, i, MPI\_ANY\_TAG, MPI\_COMM\_WORLD);  }  } else {  int buffer[delta];  int localsum = 0;  MPI\_Recv(&buffer, delta, MPI\_INT, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);  int i=0;  for(i = 0; i < delta; i++) {  localsum += buffer[i];  }  }  MPI\_REDUCE(&localsum, &totsum, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);  end\_time = MPI\_Wtime();  exe\_time = end\_time - start\_time;  MPI\_Finalize();  printf("Sum is %d. Took %d sec.", totsum, exe\_time);  } |
| job\_script  Needs to be changed for different numbers of nodes | #!/bin/bash  #PBS -l nodes=1:ppn=20  #PBS -l walltime=00:01:00  #PBS -N sum\_seq  #PBS -o sum\_seq.out  #PBS -e sum\_seq.err  #PBS -q workq  #PBS -A loni\_droz50ksv  #PBS -m e  #PBS -M mst039  mpirun -machinefile $PBS\_NODEFILE -np 5 ./sum\_seq  #!/bin/bash  #PBS -l nodes=1:ppn=20  #PBS -l walltime=00:01:00  #PBS -N sum\_seq  #PBS -o sum\_seq.out  #PBS -e sum\_seq.err  #PBS -q workq  #PBS -A loni\_droz50ksv  #PBS -m e  #PBS -M mst039  mpirun -machinefile $PBS\_NODEFILE -np 20 ./sum\_seq |
| Commands  Using LONI | mpicc -o sum\_seq sum\_seq.c  qsub job\_script  //change number of nodes in hello\_mpi.c and job\_script  mpicc -o sum\_seq sum\_seq.c  qsub job\_script |
| Output | Sum is 55. Took 00 sec.  Sum is 55. Took 0 sec. |

1. In preparation for Lab #3, have each member of the team make themselves an Amazon AWS account here: [**https://aws.amazon.com/**](https://aws.amazon.com/) Login to your account, and take a snapshot of the login screen with the title AWS services. Of the 20 main categories of “All Services” offered, pick at least 5 that interest you and give a brief description (e.g. one line) of what the specific services are used for. Since the next lab will be about Public Cloud (AWS in particular), it would be a good idea to read as much of the Service guides as you feasibly can.

Chris Conway

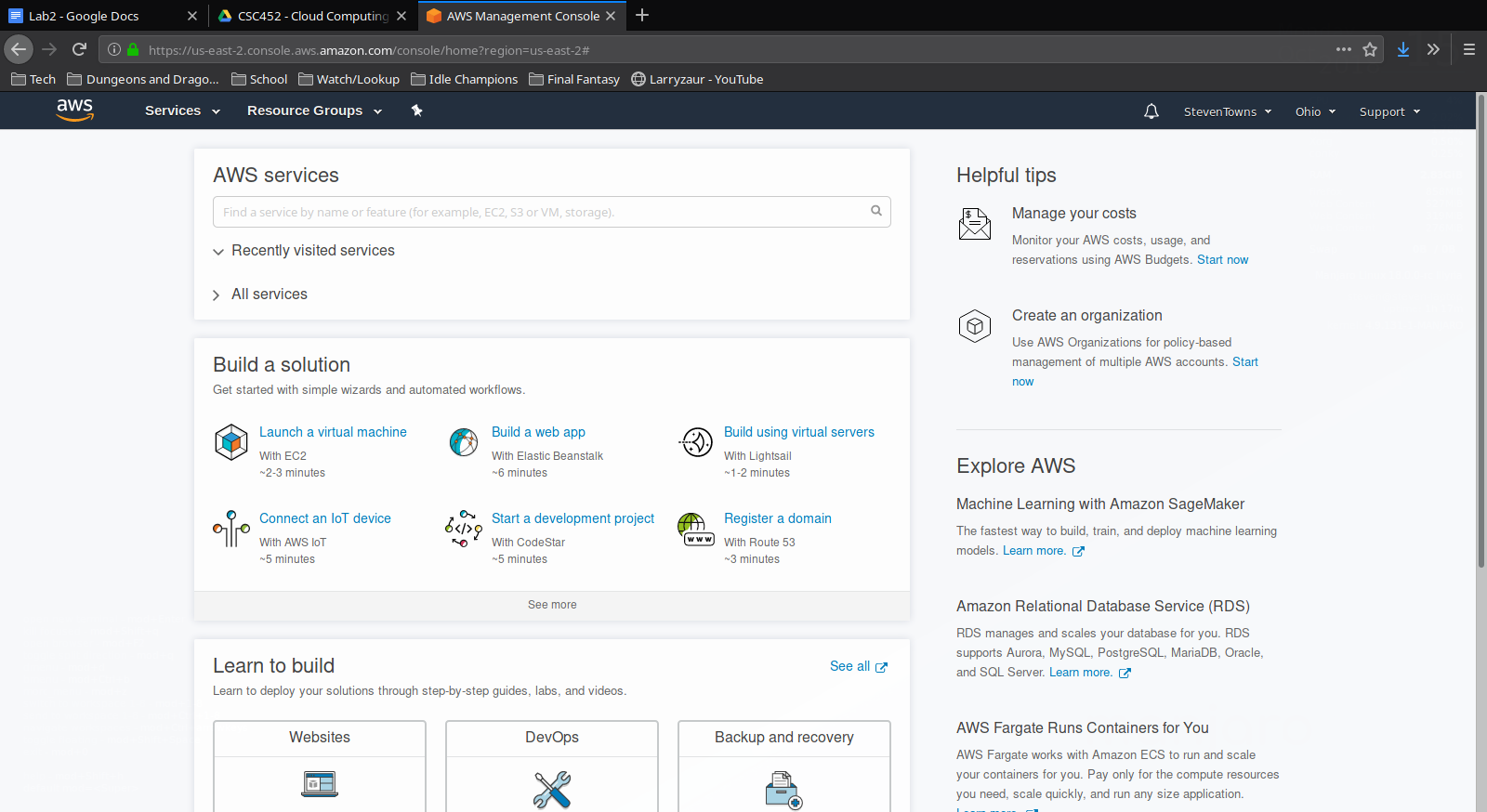
a.   
b. Amazon Sumerian: Online tool for building VR, AR, and 3D applications.  
c. Amazon GameLift: Deploy and scale session-based Multiplayer Games.  
d. CodeStar: Tool for developing, building, and deploying applications quickly.  
e. Amazon SageMaker: Tool to build, train, and deploy Machine Learning Models.  
f. RDS: Managed Relational Database Service.

Yasmine Hines

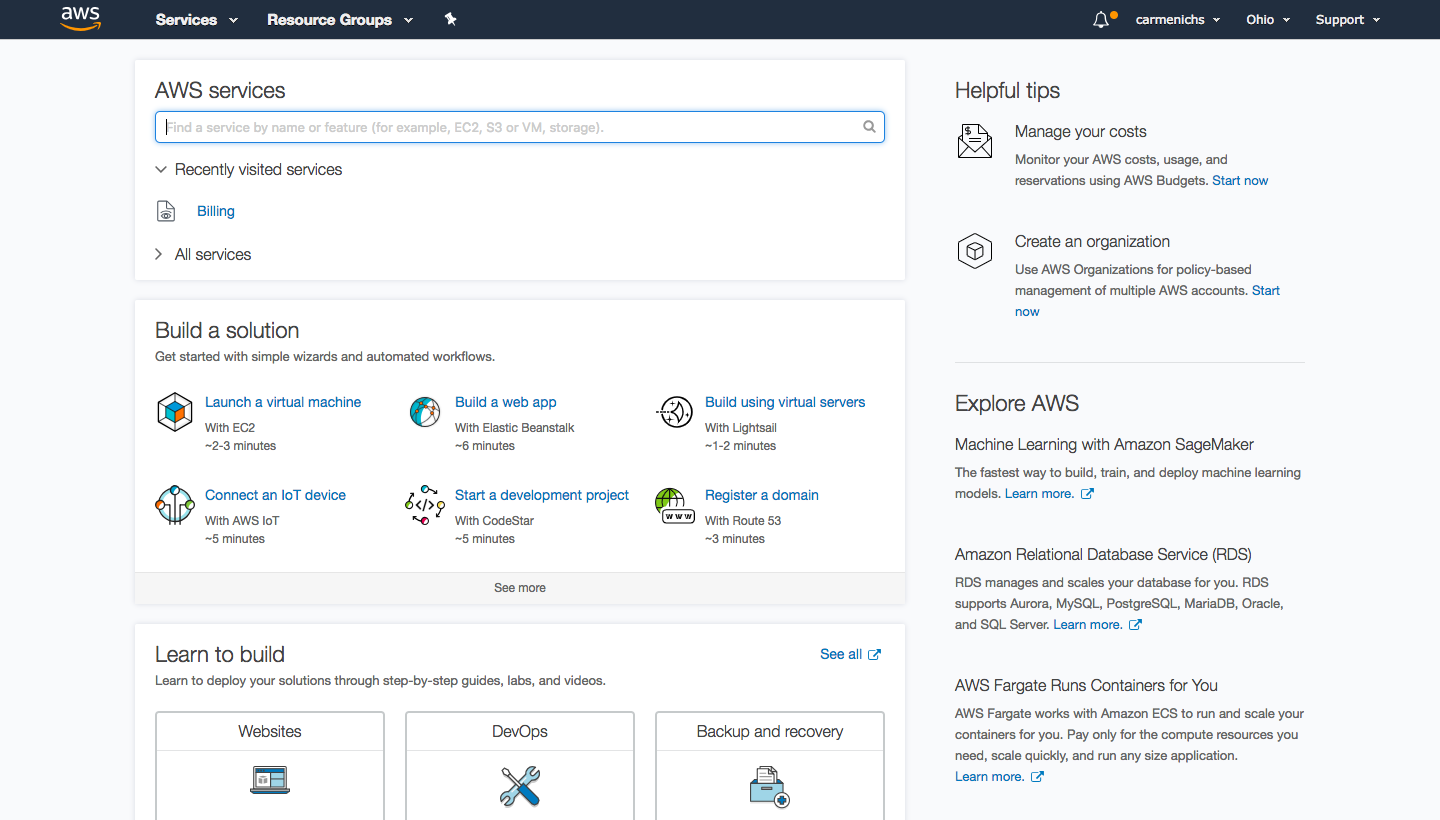


1. Kinesis - works with real-time streaming data
2. Amazon Summerian - online tool used to build VR, AR, and 3D applications
3. Cloud9 - allows you to write, run, and debug your code with just a browser.
4. API Gateway - Build, deploy, and manage APIs
5. Amazon Polly - a tool that turns text into Lifelike speech

Michael Towns



1. Secrets Manager - Tool for managing secure information such as database credentials, API keys, and authorization tokens.
2. Cloud9 - Tool for creating, running, and debugging code using a browser.
3. Amazon Pinpoint - Tool for contacting customers using email, text, or push notifications.
4. Elastic Load Balancing - Tool for managing web traffic to prevent slowdowns.
5. Amazon Inspector - Tool for improving the security and compliance of applications on AWS.

Carmen Nicholsona.MediaTailor - Tool used to Personalize and monetize multiscreen content with server-side ad insertion

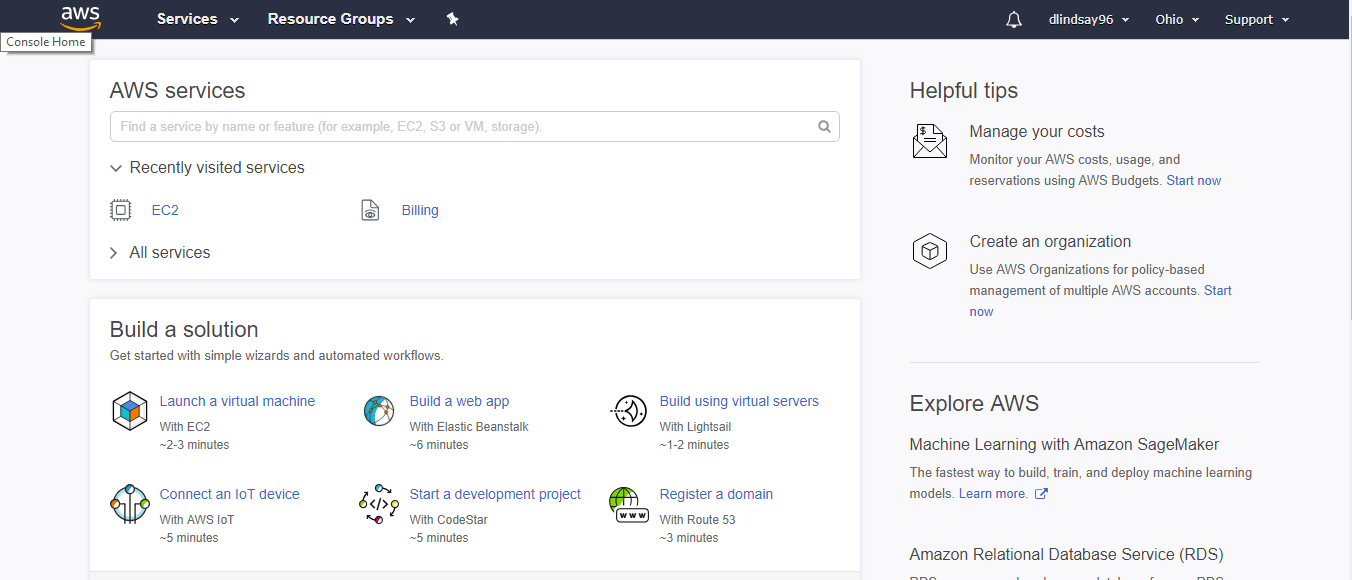
b.Glacier - Archive storage in the cloud

c. AWS DeepLens - Deep learning enabled video camera

d. Mobile Hub - Build, test, and monitor mobile apps

e. CloudWatch - Monitor Resources and Applications

Doug Lindsay



1. **Amazon Sumerian**: is a set of tools for creating high-quality virtual reality (VR) experiences on the web.
2. **Amazon GameLift**: is a fully managed service for deploying, operating, and scaling session-based multiplayer game servers in the cloud.
3. **AWS Auto Scaling**: monitors your applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost.
4. **Cloud9**: cloud-based integrated development environment (IDE) that lets you write, run, and debug your code with just a browser.
5. **AWS Deep Lens**: a fully programmable video camera designed that contains tutorials, code, and pre-trained models made to expand deep learning skills.