Open OnDemand Connector for Amazon Elastic Kubernetes Service (EKS)

Faras Sadek[1] [0009-0008-3207-3737], Milson Munakami[2] [0000-0002-3782-4797], Arthur Barrett[3] [0009-0009-9175-8266], Vesna Tan[3] [0009-0001-0432-354X], Jeremy Guillette[3] [], Robert M Freeman Jr[4] [0000-0003-1463-9748], and Raminder Singh[5] [0000-0002-7288-3979]

[1] Harvard University School of Engineering (SEAS Computing)
[2] Harvard University Research Computing (URC)
[3] Harvard University Academic Technology (ATG)
[4] Harvard Business School Research Computing Services
[5] Faculty of Arts and Sciences research computing (FASRC) at Harvard University

Abstract. Demand for computational resources is increasing in the classroom, and instructors want to train users on real-world problems by providing access to the latest resources. The Open OnDemand Connector for Amazon Elastic Kubernetes Service (EKS) is built on top of Open OnDemand (OOD), a web-based platform for accessing and managing Cluster and Cloud resources. In this paper, we have discussed how to leverage the power of Kubernetes and the flexibility of Open OnDemand to deploy a complex cloud-native solution using containerized applications, such as Jupyter, in a secure, scalable, and efficient manner.

Keywords: Open OnDemand (OOD), Amazon Elastic Kubernetes Service (EKS), High-Performance Computing (HPC), AWS Cloud

1. Introduction

In modern learning environments, courses have increased their use of computational resources in the classroom. At the same time more courses are adding increasingly intensive analytical and quantitative elements to their curriculum. Faculty who use tools like Jupyter notebooks and services such as high-performance computing in their research, also desire to teach with these same tools; however, using the traditional HPC resource manager like Simple Linux Utility for Resource Management (SLURM) [4] does not enable Research Computing groups to support the needs of courses specifically, nor the large influx of users at the start of each semester.

At Harvard, in partnership with Academic Technology (ATG) and School of Engineering (SEAS), University research computing (URC) and Faculty of Arts and Sciences research computing (FASRC), we set up an interactive computation platform using Open OnDemand [5]. Open OnDemand is an open-source project designed to lower the barrier to HPC use across many diverse disciplines [1]. In our data center, this web-based platform fronts an Academic Cluster with 50 nodes. This solution supports 30+ courses each term, providing access to Jupyter, RStudio, Stata, Matlab etc. software and statistical tools, and has 1000+ unique users every semester.

2. Problem Statement

A few of the courses require access to GPUs to teach machine learning and deep learning for half a term in a year, which presents a challenge as service providers to provide on-premises support for such expensive resources. This effort aims to build a scalable, cost-effective, and easy-to-use computation environment by integrating and extending the on-premises Academic Cluster with the resources and computational power of the Amazon AWS cloud.

In this project, we aim to connect OnDemand with a fully managed AWS EKS Cluster to support GPU based workloads and users' workflows with high memory requirements. Although the primary objective of this project is to provide the latest GPU resources in the cloud to students through the OnDemand interface, the elasticity and availability of computational resources provided by the cloud enables the project to support a wide range of computational needs on different scales. The support for state-of-the-art GPU resources within Open OnDemand enables users to take advantage of the power of GPU acceleration for their applications such as machine learning, deep learning, natural language processing, and scientific computing.

The Academic Cluster lacks sufficient GPUs to meet the demand of high-enrollment courses like *COMPSCI 109B: Data Science 2: Advanced Topics in Data Science*, which require over 100 GPUs during peak usage periods.

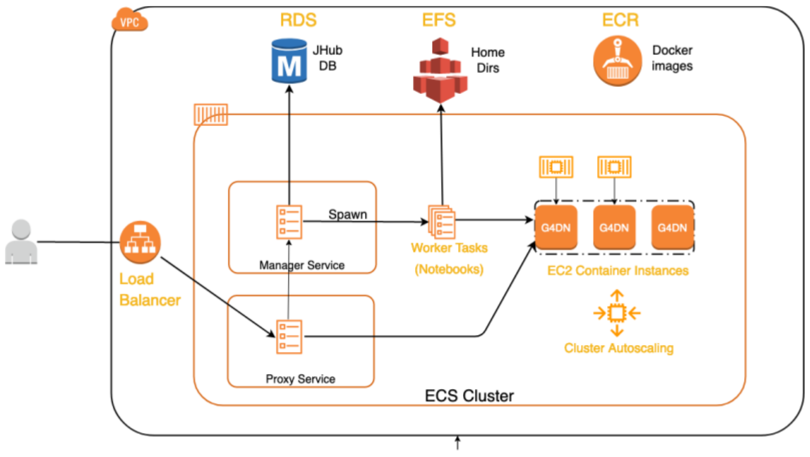


Fig.1 Architecture diagram of JupyterHub based solution.

Previously, Academic Technology deployed a JupyterHub based solution on AWS that could scale GPU instances as needed. However, this solution was separate from the academic cluster and had its own infrastructure, lifecycle, and support. Additionally, it had several limitations:

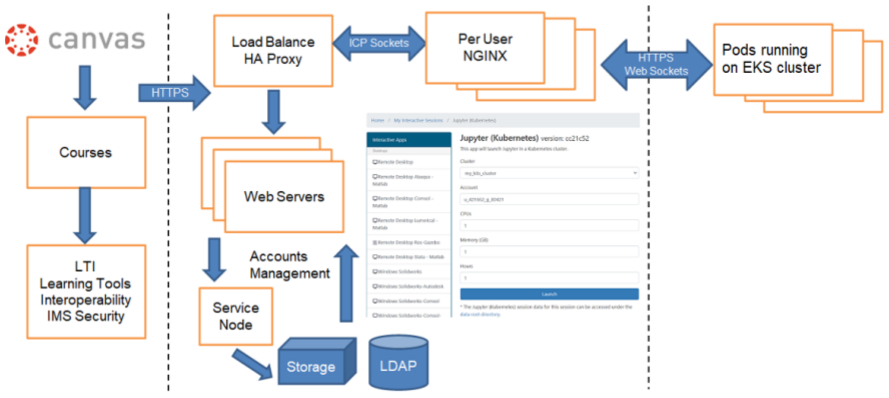
a. The JupyterHub solution uses an AWS account that is not protected by Harvard's AWS Cloud Shield (not a direct connection) and requires a Harvard security policy exception each year to run.

b. Additionally, the JupyterHub cluster is based on an ECS cluster that is not optimized for large amounts of virtualized resources such as computing, and storage required to serve dozens of courses and more than a thousand students at any one time. Therefore, we require a flexible and powerful solution like AWS EKS.

c. Moreover, it does not integrate with the existing OnDemand platform where students/courses have a familiar and uniform interface for accessing compute resources and are already working with a variety of applications beyond Jupyter (e.g. RStudio, etc.).

3. Approach

Our goal is to integrate the existing OnDemand solution with AWS and configure the OnDemand application to launch a Jupyter application with access to GPU resources. Access to the course is controlled through Canvas (a web-based, learning management system, or LMS), and only authorized courses will be granted access to the AWS Jupyter application. Additionally, students will not have direct access to the infrastructure; all underlying resources will be hidden from them.


canvas
Courses
LTI
Learning Tools
Interoperability
IMS Security
HTTPS
Load Balance
HA Proxy
ICP Sockets
Per User
NGINX
HTTPS
Web Sockets
Pods running
on EKS cluster
Web Servers
Accounts
Management
Service
Node
Storage
LDAP
Home / My Interactive Sessions / Jupyter (Kubernetes)
Interactive Apps
Desktops
Remote Desktop
Remote Desktop Abaqus - Matlab
Remote Desktop Comsol - Matlab
Remote Desktop Lumerical - Matlab
Remote Desktop Ros-Gazebo
Remote Desktop Stata - Matlab
Windows Solidworks
Windows Solidworks-Autodesk
Windows Solidworks-Comsol
Windows Solidworks-Comsol-
Launch

Jupyter (Kubernetes) version: cc21c52

This app will launch Jupyter in a Kubernetes cluster.

Cluster: my_k8s_cluster

Account: [illegible]

CPUs: 1

Memory (GB): 1

Hours: 1

* The Jupyter (Kubernetes) session data for this session can be accessed under the data root directory.

Fig.2 Integration of Canvas, OnDemand and EKS cluster

Open OnDemand [5] is a web-based portal that provides access to cluster services. Through OnDemand, students are able to upload and download files, submit jobs, and most importantly, run pre-configured interactive applications such as Jupyter or

Rstudio. OnDemand simplifies the complexity inherent in accessing and allocating compute resources, which is typically the responsibility of the Academic Cluster's workload manager, Slurm[4]. Although it is commonly used in high-performance computing and Linux-based clusters, most students, and even some faculty, are not familiar with it, so OnDemand provides the necessary high-level interface to take advantage of cluster resources without needing to understand the underlying details.

Kubernetes is a complete open-source container orchestration tool used to automate the deployment, scaling, and management of containerized applications. It is a popular solution for handling container-based workloads and has several compelling features for this project. Key among those features is the overall flexibility and scalability of the platform, as well as the extensive ecosystem of plugins and strong support from multiple cloud vendors in the form of managed services. This provides a strong base to manage resources for OnDemand application jobs in the cloud, which can then be coordinated with Slurm in the Academic Cluster.

Due to the complexity of this project, which involves integrating several components such as OnDemand, Canvas, AWS Elastic Kubernetes Service (EKS. We initially deployed an OnDemand node in the cloud and configured the system to run computational resources on the AWS EKS cluster. This approach helped us conduct a feasibility study and test different services available in AWS. We also evaluated different security solutions, such as AWS Identity and Access Management (IAM) and AWS Cognito and ensured that they aligned with Harvard's security requirements. The diagram below illustrates the high-level architecture of the solution, and we took an iterative approach to implement the current working solution.

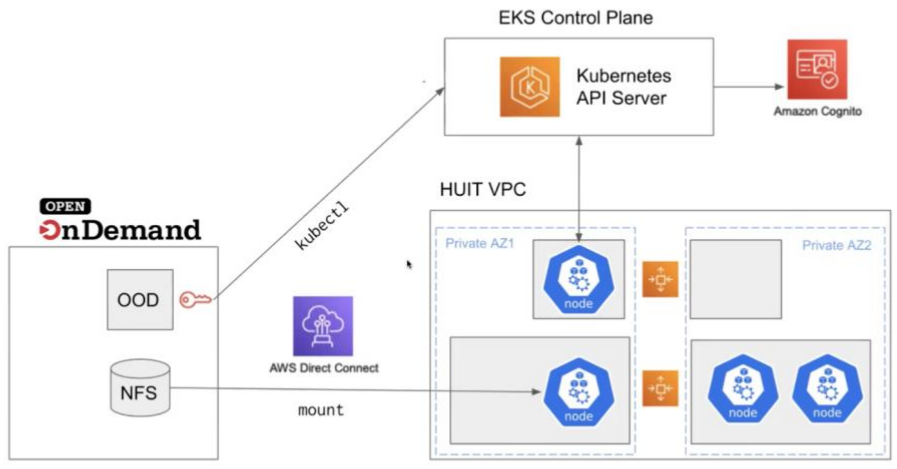


Fig.3 High-level Architecture Diagram

In Kubernetes, a pod is the fundamental deployment unit that represents a group of containers that are closely connected and share common resources, such as network interface and storage. The entire pod, including all its containers, runs on a single EC2 machine allocated by the AWS autoscaler and shares the same private IP address. The kubelet, the primary "node agent" that runs on each node, communicates with the Kubernetes control plane and Kubernetes nodes. In our solution, the resource request from Kubernetes specifies the minimum and maximum amount of resources that the application container will require in a pod. While Kubernetes includes a reservation mechanism to minimize resource contention, it is limited to CPU [3] and maximum memory, potentially leading to reduced resource utilization. We are trying to optimize the GPU-based resource allocation based on student demand to support the class workload that needs to handle data-intensive and computation-intensive jobs.

4. Solution

a. Requirements

The solution is driven based on the following functional requirements:

1. User experience should remain the same as the previous Open OnDemand interface, and the underlying system and infrastructure complexity should be hidden from users.
2. Existing Open OnDemand users should be authenticated to Kubernetes and authorized to access their own servers/containers only.
3. On-premises users' home directories and files should be mounted on their Kubernetes nodes to ensure seamless access and data transfer.
4. Quality attributes like security, availability, reliability, and scalability are also considered during the design phase.

b. Implementation

The EKS cluster uses AWS Cognito as an OpenID Connect (OIDC)-compatible identity provider and is set up with two node groups, GPU computation and Kubernetes autoscaler. Each is a logical grouping of nodes that share common attributes, such as instance type, availability zone, and scaling configuration managed by the AWS autoscaling service. Kubernetes autoscaler node group has a minimum capacity of at least one node, while the other node group handles and manages the users' GPU compute nodes used by the EKS cluster. This GPU compute node group template also includes user data or a custom bootstrap script that connects each node provisioned and imaged using this template to the EKS

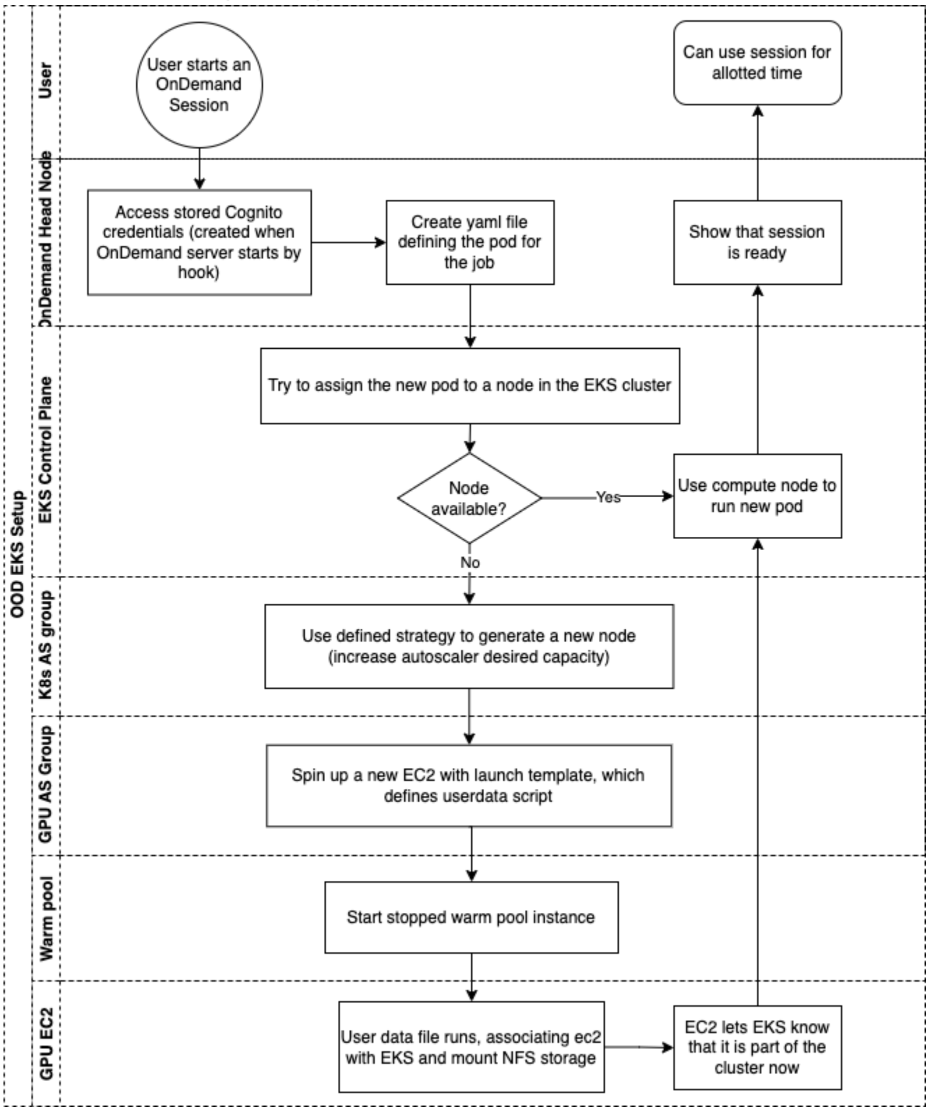

OOD EKS Setup
User
OnDemand Head Node
EKS Control Plane
K8s AS group
GPU AS Group
Warm pool
GPU EC2
User starts an OnDemand Session
Can use session for allotted time
Access stored Cognito credentials (created when OnDemand server starts by hook)
Create yaml file defining the pod for the job
Show that session is ready
Try to assign the new pod to a node in the EKS cluster
Node available?
Yes
No
Use compute node to run new pod
Use defined strategy to generate a new node (increase autoscaler desired capacity)
Spin up a new EC2 with launch template, which defines userdata script
Start stopped warm pool instance
User data file runs, associating ec2 with EKS and mount NFS storage
EC2 lets EKS know that it is part of the cluster now

Fig.4 An overview of solution workflow

cluster and also mount the on-premise NFS file system to the new node. The auto-scaling node runs a Kubernetes autoscaler pod that has an IAM role to adjust the number of running nodes in the users' computation node group. When a user requests a GPU node from the computation node group, AWS autoscaler checks the available nodes to fulfill the user request. If no nodes are available, Kubernetes autoscaler requests a new node from the computation node group to be attached to the EKS cluster. Once the node becomes ready, Kubernetes auto-scaler assigns the node to the requested workloads. This means that the configured autoscaling group can scale nodes in a node group as needed to optimize the cluster's performance and cost. To reduce the nodes provisioning times, warm pool is configured for the GPU node group.

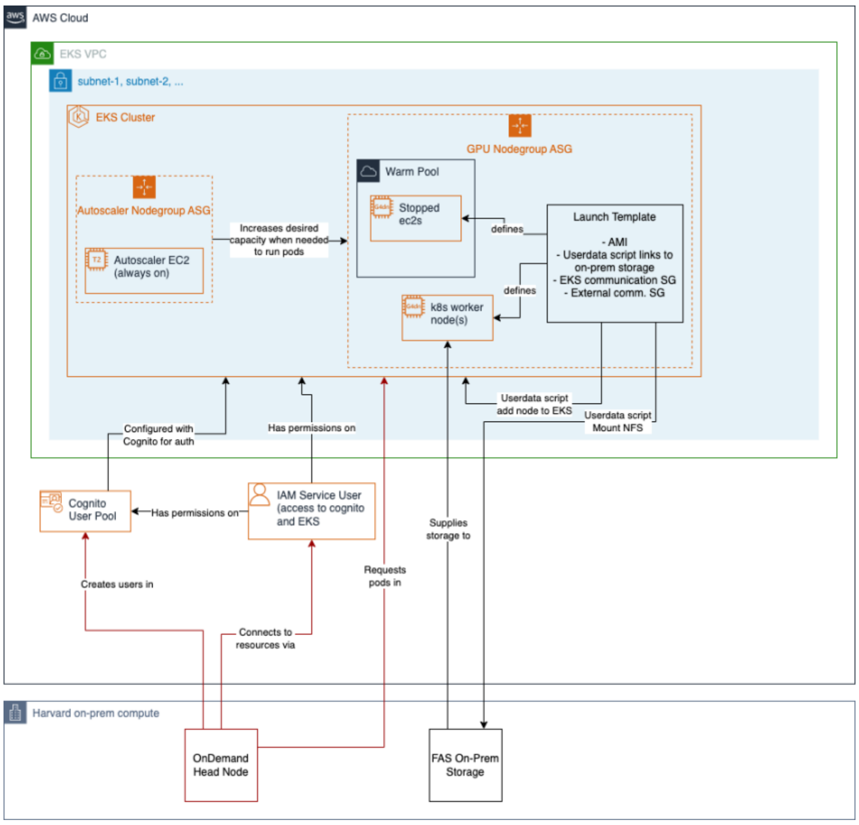


Fig.5 Implementation Flow Diagram.

c. User Access

Students are expected to access GPU resources by logging into Canvas and using an LTI tool installed in their Canvas course to connect to OnDemand. This is the same process they already use to access non-GPU resources provided by OnDemand. The main difference is that in order to access GPU resources managed by Kubernetes, the user must be able to authenticate with AWS Cognito and then be authorized by Kubernetes RBAC. This part of the process is intended to be handled automatically and hidden from the end user.

AWS Cognito provides federated identities service to users registered in user pool, which provides a unique token credential to the authenticated user via OIDC to grant access to the AWS EKS. The token identifier provider (IdP) is trusted by AWS Security Token Service (STS) to access temporary, limited-privilege AWS credentials.

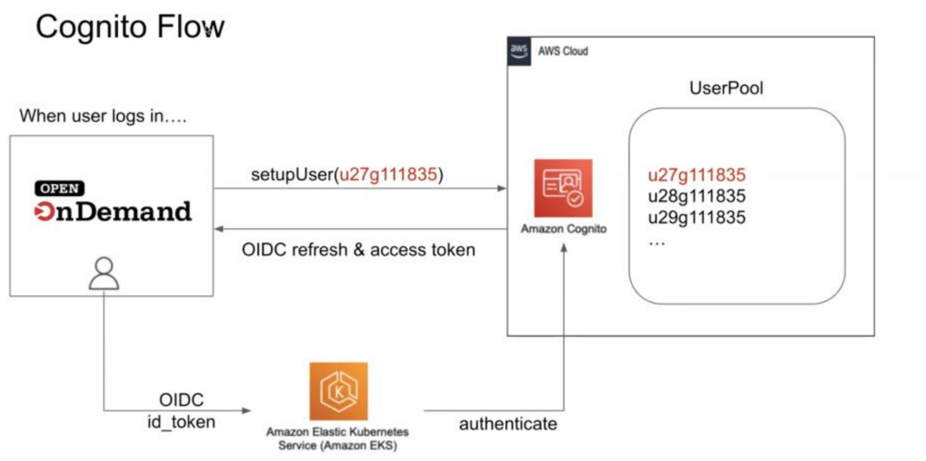


Fig.6 An overview of Amazon Cognito security workflow

Student accounts are automatically added to the Kubernetes cluster by OnDemand as part of a pre-hook. It needs to initialize root-user credentials, which is executed when the user's per-user Nginx (PUN) process starts. Furthermore, users are created in AWS Cognito user pools as part of the pre-hook workflow. In other words, these accounts are not created until a user accesses OnDemand from Canvas. When a user accesses OnDemand, the pre-hook script checks whether the user exists in Amazon Cognito. If not, the pre-hook creates the user programmatically using the AWS CLI and IAM service account and returns a user token identifier. If the user already exists, the pre-hook simply returns a Cognito token for that user. The user's token is stored in the corresponding OnDemand user's KubeConfig file (e.g., ~/.kube/config) located in the user's home directory.

Once the pre-PUN hooks perform a token exchange, the user can authenticate to the Kubernetes cluster. User authorization is handled internally by Kubernetes RBAC, which is also bootstrapped by the pre hook script upon user login to ensure security. All user pods run in a user-specific, unique, and isolated namespace. Then, the authorized user can request computational resources from the EKS cluster via OnDemand apps such as the Jupyter application.

d. Optimization

The importance of optimized resource usage cannot be overstated, and crucial to achieving this is deploying a service that can automatically clean up (deprovision? kill?) pods that have exceeded their allotted walltime. In this solution, we use the job-pod-reaper service provided by Open OnDemand [1] to kill pods that have reached their walltime and ensure that OnDemand pods are shut down within a specific time frame. To achieve this, the job-pod-reaper service runs as a Deployment inside the Kubernetes cluster and kills pods based on the lifetime annotation noted on deployed pods. Once the pods are removed from the nodes, the corresponding AWS Elastic Compute Cloud (EC2) instances are scaled down and returned to the warm pool of the Users' GPU computation node group.

5. Challenges

During the development, deployment, and testing, we ran into several challenges.

1. Initially, we used the built-in EKS IAM authenticator, but due to security concerns and policy constraints on the programmatic creation of AWS IAM accounts associated with students, we had to explore alternative solutions like Cognito. Using Cognito, we were able to apply proper policies to those accounts to restrict access.
2. While testing the solution, we noticed that launching a container (Jupyter job) often took 5+ minutes from a cold start (no running instances available for the job to be placed on immediately). One immediate improvement was preloading the docker image on the EC2 instances (Kubernetes nodes). Since the Jupyter docker images were often 10+ GB in size, skipping the image pull saved 1-2 minutes on average. We also explored the idea of warm pools to further reduce the launch time to ~3 minutes for a new container; if an instance is already running in the warm pool, a user's job starts immediately. In future work, we will be exploring other solutions like AWS Karpenter to further optimize the runtimes.
3. During stress testing, we ran into the problem of IPv4 address exhaustion. This is not an uncommon issue with Kubernetes solutions. Initially, we were given two subnets of CIDR /24 with 508 usable IPs. Given that each jupyter job in our solution is assigned to a single g4dn.xlarge instance in order to use its GPU, that meant that each jupyter job effectively consumed 5 IP addresses (kubelet, kube-proxy, aws-node, and nvidia-device-plugin-daemonset, jupyter job). This limited the maximum capacity of the cluster to 101 concurrent users. This was further complicated by the fact that by default, the Amazon VPC CNI plugin automatically attempts to reserve one full ENI of IP addresses so that pods can be immediately assigned an IP address. But in our case, this caused instances to rapidly consume IP addresses. We were able to disable this behavior and only reserve the

minimum required IP addresses per node, but that still imposed a hard limit on the number of possible Jupyter jobs. We worked with the network team to set up two /23 subnets, which allowed us to have 1024 IPs to support a class of ~100 active students with some headroom. For future expansion, we would need to consider an alternative network architecture such as using a private NAT gateway solution.

4. A user's session time is set up during the resource request, which for us is the GPU node.. If the user computation time requires more than the requested time, we need to find a way to enable the extension of the session as needed.
5. EKS nodes have an IAM Role with a permission policy to connect to the EKS cluster. Users on any given node thus will have access to the nodes' set of roles, giving them the ability to make AWS API calls to run other services. We needed to block access to the node's role in order to prevent this.

6. Conclusion

In summary, due to the increasing demand for specialized and specific computational resources in academic courses, which are not readily available through conventional cluster resources, we propose a solution that provides an extensible and user-friendly connection to the AWS-managed EKS cluster. This solution overcomes the limitations of the previously used JupyterHub approach and the deficiencies of traditional resource managers like SLURM by seamlessly integrating an on-premises Academic Cluster and Open OnDemand with the extendable and diverse computational power of the Amazon AWS cloud. Thus, by integrating Open OnDemand with a fully managed AWS EKS Cluster, the solution ensures adherence to secure and scalable infrastructure and policies, offering students a safe and reliable experience via on-demand teaching tools. For future work, we have already started collaborating with Harvard Business School Research Computing services to generalize the solution for their application needs. We plan to open-source the code and improve on documentation for better adoption.

References

1. Settlage, R., Chalker, B.A., Franz, E., Gallo, S., Moore, E., and Hudak, D.. Open OnDemand: HPC for everyone. Retrieved from https://par.nsf.gov/biblio/10122554. ISC 19.
2. Hudak, D., Johnson, D., Chalker, A., Nicklas, J., Franz, E., Dockendorf, T. and McMichael, B.L.: Open OnDemand A web-based client portal for HPC centers. Journal of Open Source Software. 3(25), 622 (2018). https://doi.org/10.21105/joss.00622.
3. Medel, V., Tolón, C., Arronategui, U., Tolosana-Calasanz, R., Bañares, J.Á., Rana, O.F. (2017). Client-Side Scheduling Based on Application Characterization on Kubernetes. In: Pham, C., Altmann, J., Bañares, J. (eds) Economics of Grids, Clouds, Systems, and Services. GECON 2017. Lecture Notes in Computer Science(), vol 10537. Springer, Cham. https://doi.org/10.1007/978-3-319-68066-8_13
4. Simple Linux Utility for Resource Management (SLURM): https://slurm.schedmd.com/
5. Open OnDemand: https://openondemand.org/
6. OnDemand Kubernetes connector: https://osc.github.io/ood-documentation/develop/installation/resource-manager/kubernetes.html