Open OnDemand Connector for Amazon Elastic Kubernetes Service (EKS)

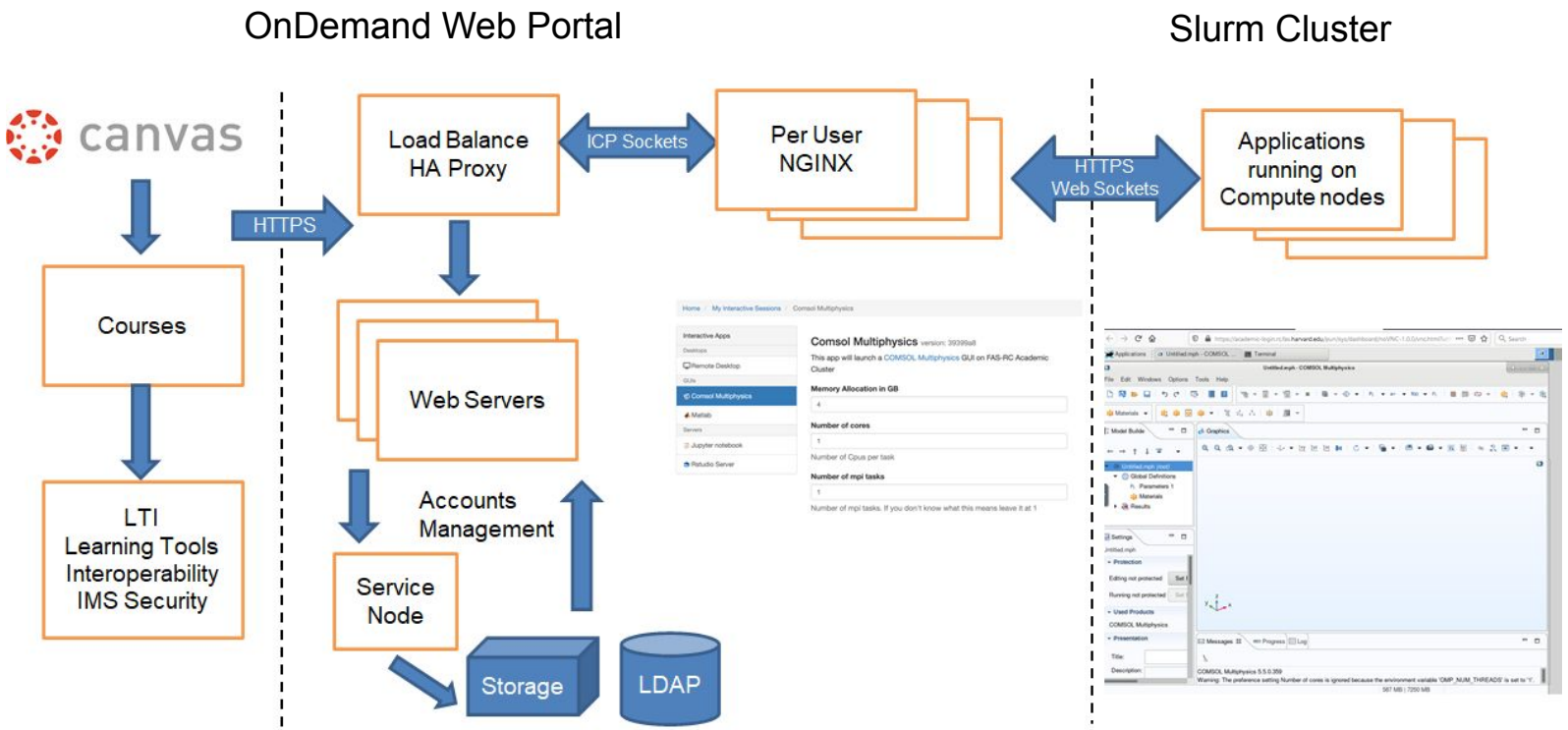
Raminder Singh
Faras Sadek
Harvard University
May 25th 2023, Hamburg, Germany

Background

In partnership with Academic Technologies (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.

- 30+ courses per term
- 1000+ unique users.
- Access to ~50 compute nodes
- 1M+ CPU Hours user term
- Access to Linux and Windows applications
    - Jupyter, RStudio, Stata, Matlab, Comsol, Lumerial, CodeServer etc.
    - Solidworks (Windows)

Setup based on Academic Slurm Cluster

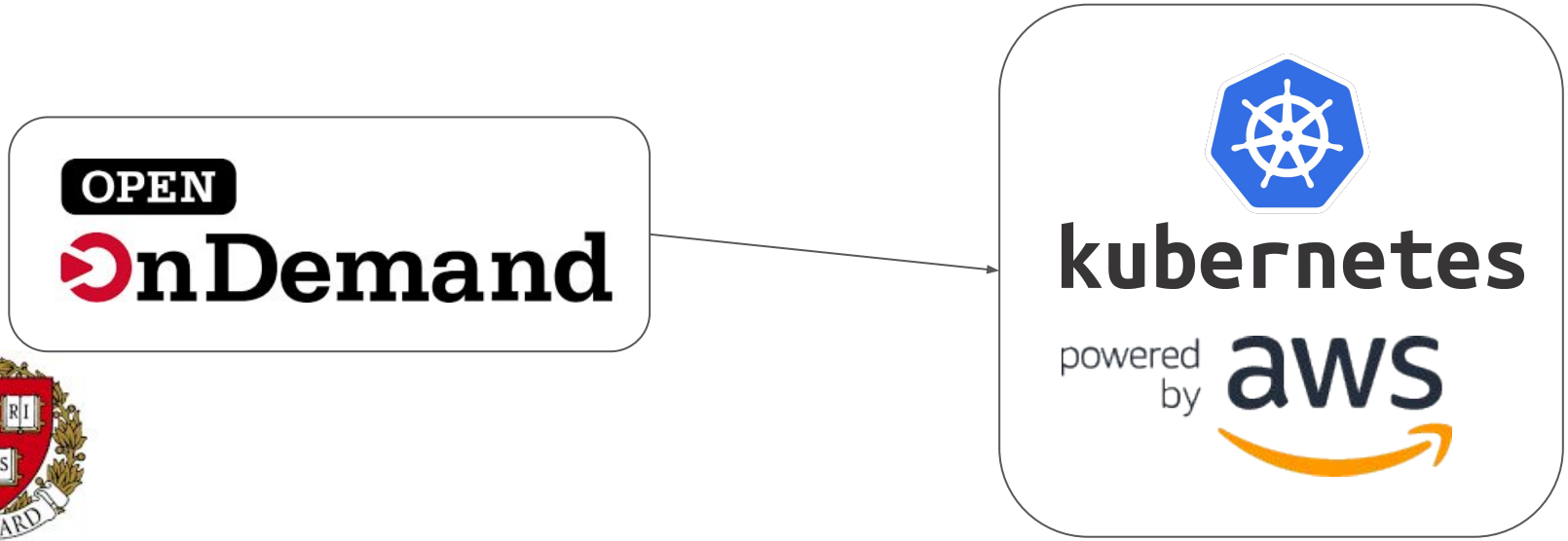


Introduction

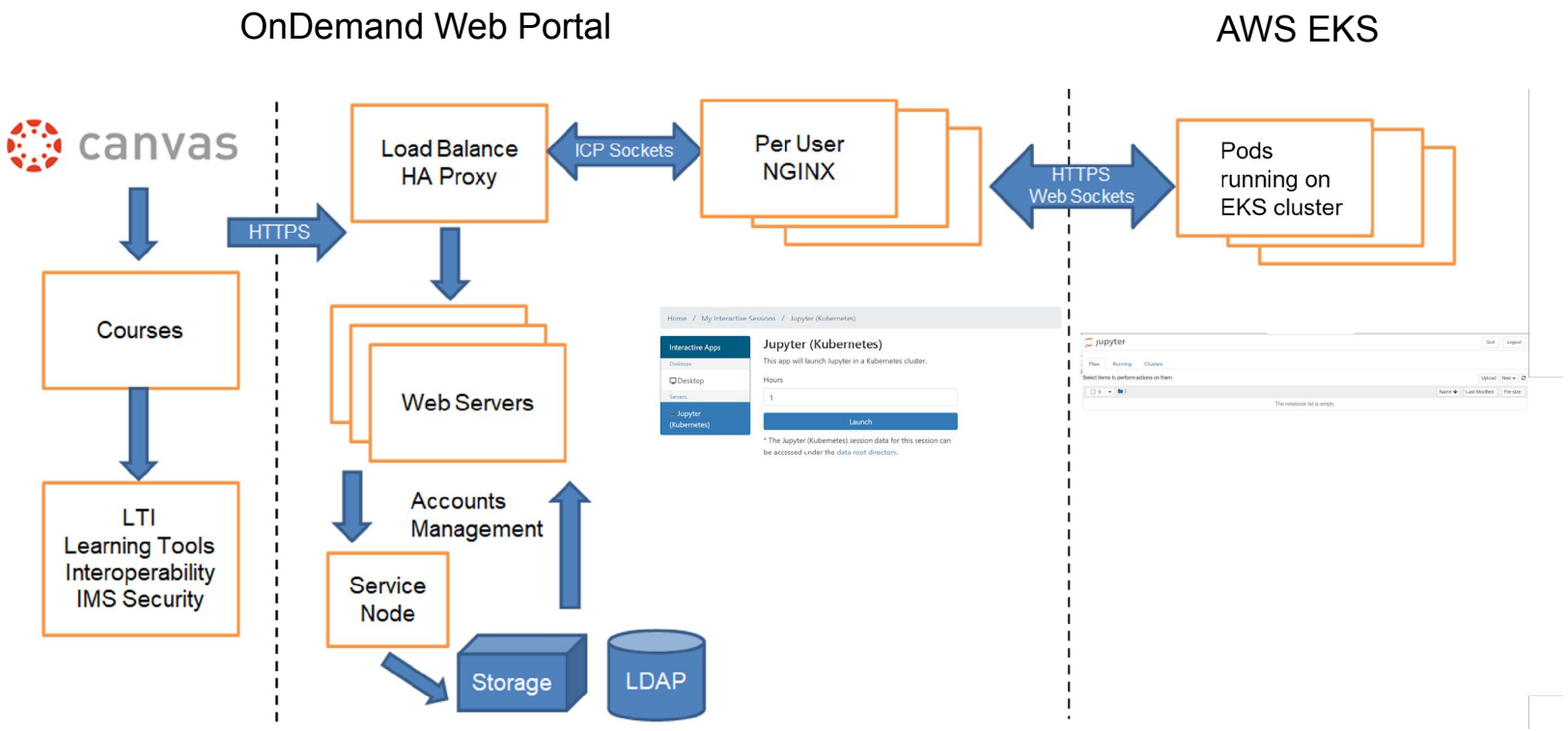
Problem: The on-premise academic cluster that powers our current Open OnDemand does not have enough Graphics processing units (GPUs) to meet the needs of courses like

- CS109b ~ 200 Students
- Each student needs a GPU card

Solution: Connecting existing OnDemand portal using Kubernetes plugin to Amazon EKS



Setup with AWS Elastic Kubernetes Service (EKS) Cluster



Functional Requirements

- Must provide same OOD frontend experience to users as on-prem (hide infrastructure).
- Must be able to authenticate existing OOD users with Kubernetes and authorize them to run their own pods.
- Must be able to mount NFS storage (home directories) from on-prem storage
- Must be able to run GPU workloads and scale capacity up/down.
- Must have quality attributes like security, availability, reliability, and scalability

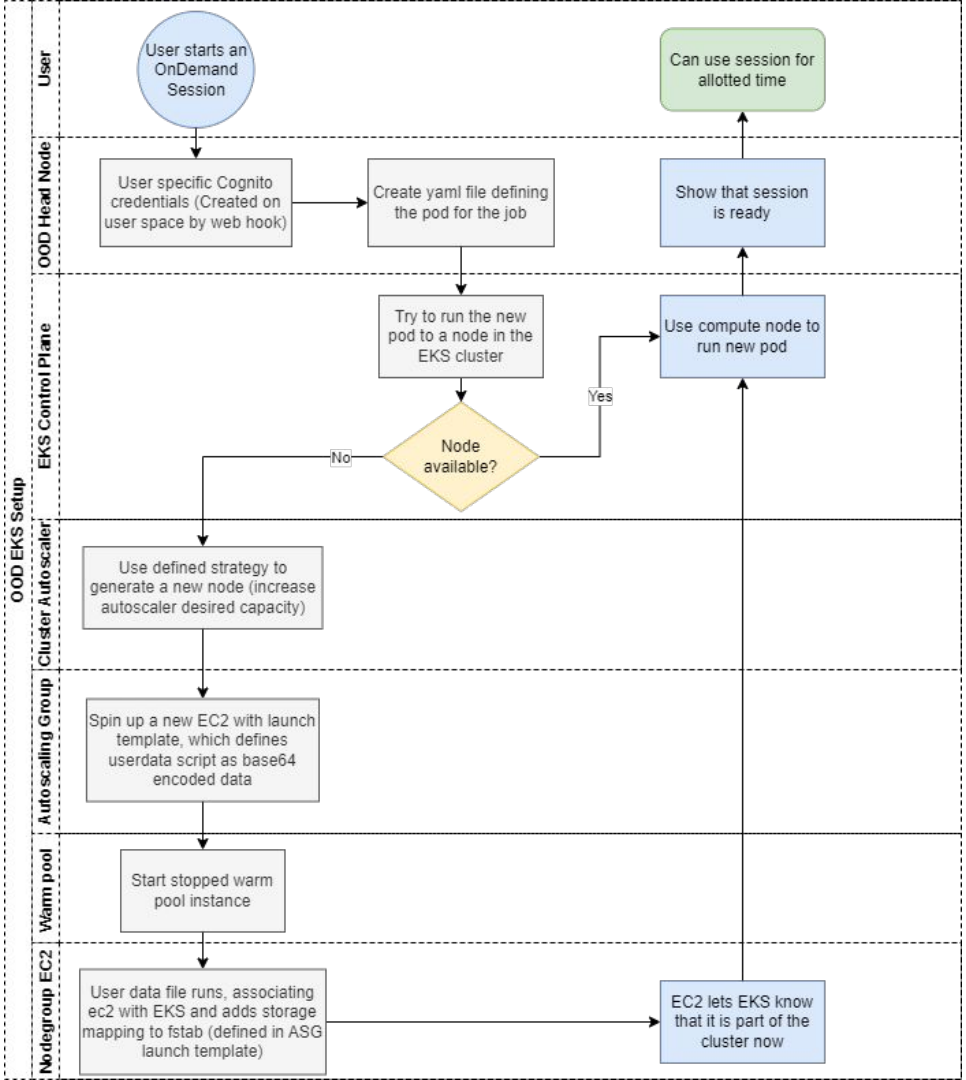
Infrastructure (high-level)

- AWS Accounts:
  - Two accounts (DEV and PROD)
  - Standard VPC with DirectConnect setup by HUIT (everything in private subnets)
- EKS
  - Runs the Kubernetes *control plane* in AWS and integrates with AWS services
  - Connects with Cognito for authentication using OIDC authenticator
  - Authorization is handled by Kubernetes natively
- Cognito
  - Handles authentication of users with Kubernetes
  - OpenID Connect (OIDC) provider to EKS
- EC2
  - Runs the Kubernetes *data plane* and provides the nodes/compute for the cluster
  - Uses Autoscaling Groups and Warmpools
- Identity and Access Management (IAM)
  - A single IAM service account is needed by OnDemand servers to manage EKS and Cognito

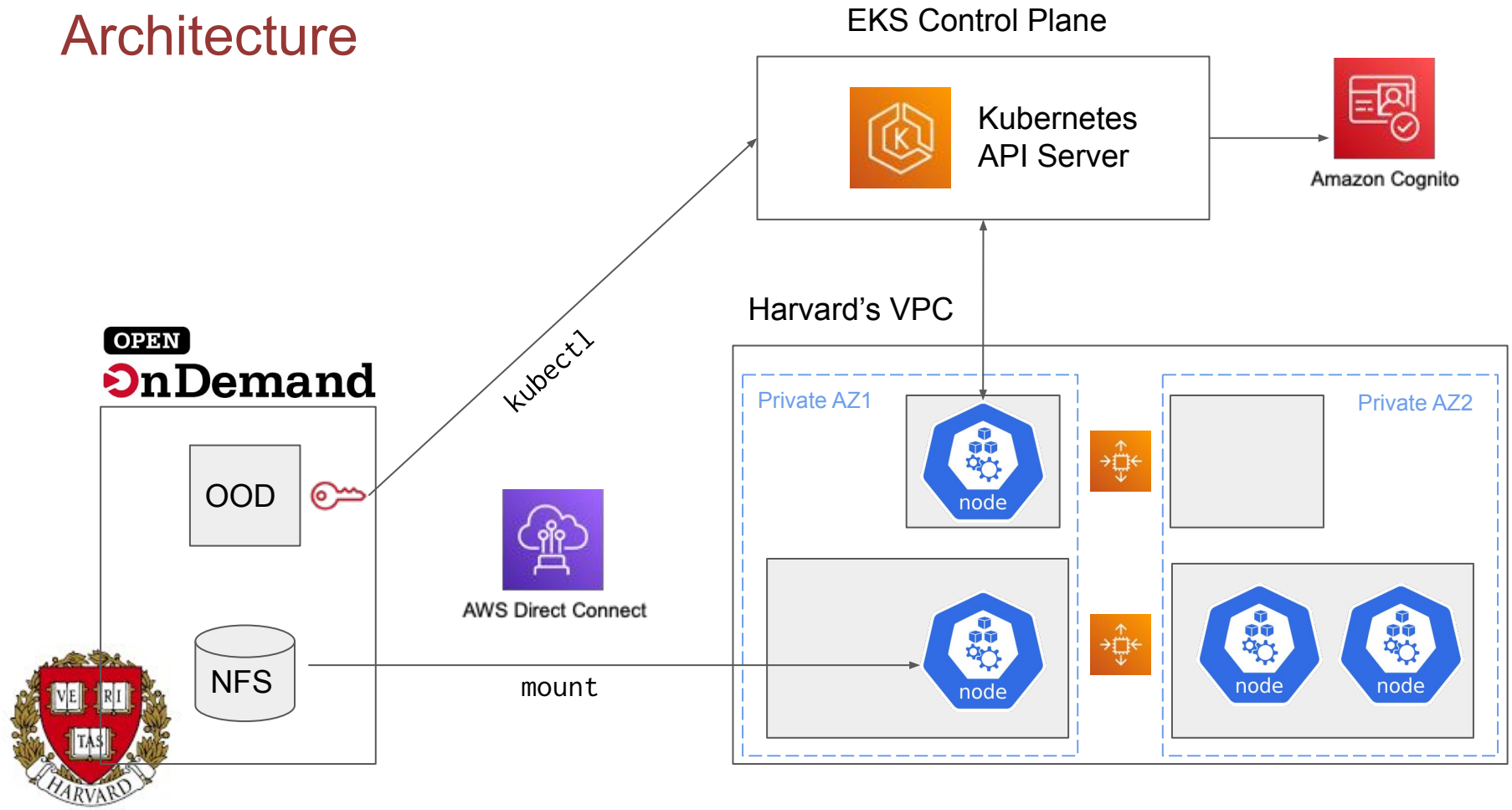
Tools

- **AWS CLI** – A command line tool for working with AWS services, including Amazon EKS.
- **kubectl** – A command line tool for communicating and working with Kubernetes clusters.
- **eksctl** – A command line tool for working with EKS clusters that automates many individual tasks.
- **Helm** - The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster using Helm charts.

Solution Overview



Architecture

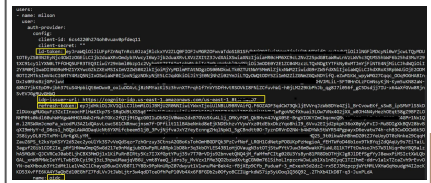


Cognito Flow

When user logs in….

setupUser(u27g111835)

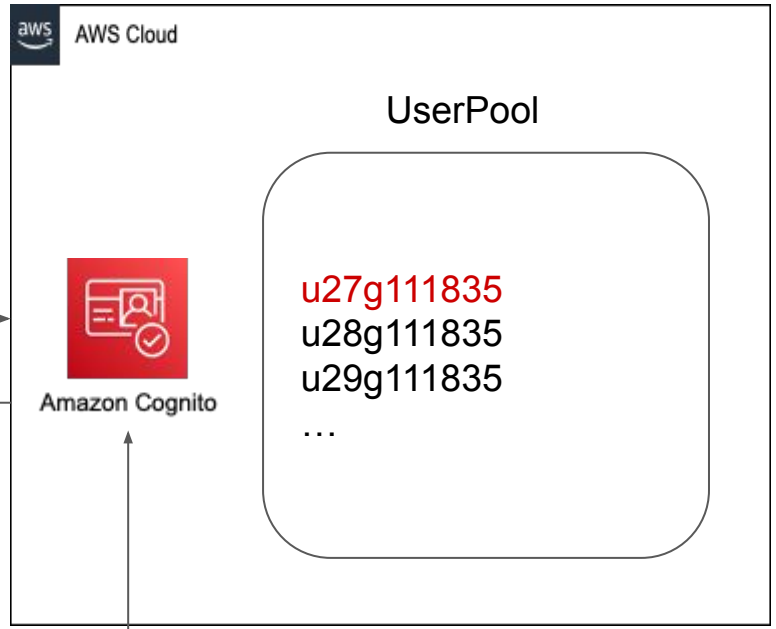
OIDC refresh & access token



OIDC
id-token

Amazon Elastic Kubernetes Service (Amazon EKS)

authenticate



EKS Control Plane

- Amazon EKS provides a scalable and highly-available Kubernetes control plane running across multiple AWS Availability Zones (AZs).
- API endpoint to connect
- Cognito as OIDC authentication provider
- Nodegroups are Amazon EKS managed worker nodes
- Nodegroups run Amazon EC2 instances using the latest EKS-optimized or custom Amazon Machine Images (AMIs) in your AWS account

Autoscaling

- Autoscaling Kubernetes supports scaling workloads based on demands.
- Primarily rely on the Kubernetes Cluster Autoscaler (CAS) and EC2 Auto Scaling groups (ASG)
- Two key Node Groups:
  1. Cluster Autoscaler Node Group:
     - An autoscaling group that runs a cluster-autoscaler pod
     - The sole responsibility of this pod is to increase/decrease the data plane size
     - ASG size = 1 to ensure it's always running
  2. Worker Node Group (GPU or General CPU):
     - An autoscaling group that manages the worker instances
     - ASG size = 0 - 200 depending on resources needed

     Warmpools

Applications / data plane

- To run highly available and resilient application and data plane on Amazon EKS
- Spread workloads across multiple data plane nodes
- Assigning pods to nodes
- Each pod with unique private IP address from the VPC
- Each pod has multiple containers to run Jupyter app
- Application loads container images are stored in a container registry
- Container includes Cuda drivers, TensorFlow and Pytorch with GPU support

Networking and Security

- Containers run as Non-Root User
  - Avoid privilege escalation
  - Given the appropriate user privileges
- IAM provides fine-grained access control
- Amazon VPC isolates Kubernetes clusters from other customers
- Users & Permissions with RBAC for managing user access to templates, roles, users, namespaces, serviceaccounts, etc.
- Per user Namespace
- Enforce Network Policies using Calico network policy engine
- Monitoring and logging
- Private Subnets, Security Groups and network ACLs

Namespace

- To organize the K8s resources.
- Each namespace provides a separate/ isolated scope for the resources in the cluster, including pods, services, and other objects.
- Attributes:
  - Resource isolation
  - Access control
  - Resource quotas: Limit Tenant’s use of Shared resources
  - Multitenancy: concurrent processing and isolates the application and data from one user to another

Persistent storage

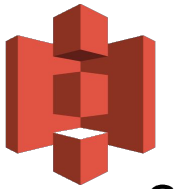
- Stateful Kubernetes workloads persist data using external storage platforms.
- **AWS Direct Connect** and VPN connections are two ways to provide network connectivity between applications running on Amazon EKS and local storage VPCs.
- Requires performant connections to provide consistent network experience

Amazon Elastic File System (EFS)

- A fully managed network file system that can be accessed via the Network File Share (NFS) protocol
- Mounting into both to Open OnDemand web portal as well as the Worker nodes to provide consistent shared user home directory
- Less input/output operations per second (IOPS)
- May be Luster can be more optimized performant file system or even S3

Amazon Elastic File System

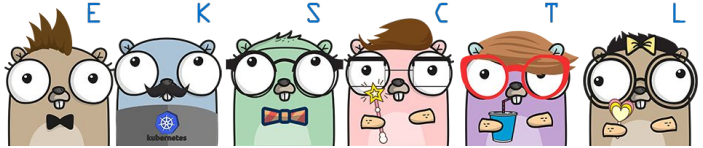
FSx for Lustre



Amazon S3

Deployment

1. Deployment of the Amazon EKS cluster



- Bootstrap Amazon EKS cluster using eksctl
- Simplify cluster management/ operations including managing nodes and addons.
- Running **deploy.sh** bash script using **.env file** Or **command line arguments**

```
REGION=us-east-1
VPC_ID=vpc-xxxxxxxxxxxxxxxxx
SUB_ID_1=subnet-xxxxxxxxxxxxxxxxx
SUB_ID_2=subnet-xxxxxxxxxxxxxxxxx
SSH_KEY=mysshkeyxx
IAM_USER=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
AWS_ACCOUNT_ID=123456789012
OOD_ENV=dev
OOD_CIDR=10.31.0.0/16
SGS_STACK_NAME=my-sgs-stack
SG_ID_NODES=sg-xxxxxxxxxxxxxxxxx
SG_ID_CONTROL_PLANE=sg-xxxxxxxxxxxxxxxxx
SG_ID_SHARED_SERVICES=sg-xxxxxxxxxxxxxxxxx
COGNITO_STACK_NAME=my-cognito-stack
NODE_TYPE=general
UserPoolId=us-east-1_xxxxxxxxx
UserPoolClientId=xxxxxxxxxxxxxxxxxxxxxxxxxx
UserPoolIssuer=https://cognito-idp.us-east-1.amazonaws.com/us-east-1_xxxxxxxxx
```

```
./deploy.sh CLUSTER_NAME
```

```
./deploy.sh CLUSTER_NAME OOD_ENV REGION VPCTAG SUB1TAG SUB2TAG SSH_KEY IAM_USER IAM_ROLE OOD_CIDR SGS-STACK-NAME COGNITO-STACK-NAME gpu|general
```

2. Deployment of the Open OnDemand Web Node

- On Academic cluster, we don’t need to re-deploy the OOD web node.
- The output of the previous cluster deployment are cluster configuration files, web hook file and certificate
- Need to add them to the puppet configuration for Academic cluster

- To deploy a new OOD web node:

```
./ood-installation.sh <HOSTNAME> <CLUSTER_CONFIG_FILE> <K8S_CERTIFICATE> <HOOKENV>
```

Clean up/Deprovision cluster

- Clean up/Deprovision cluster infrastructure

```
./delete.sh CLUSTER_NAME REGION
```

- Cleanup by deleting all cluster related CloudFormation stacks

Future Works / Open Issues

- Best practices for seamlessly implementing GitOps: resulting in enhanced performance and substantial cost savings.
- Using Spot instances or even serverless compute engine Fargate
- Cost monitoring: **Kubecost**
- Security hardening tool: **Datree**
- Monitoring and Observability
- Karpenter based cluster autoscaling: no need to use Nodegroups, which scales nodes based on resource requirements
- Extending the submitted job time
- Automated backup & restoring

Demo

Links

Code Repositories:

- https://github.com/fasrc/OnDemand_Kubernetes

Kubernetes and EKS:

- https://kubernetes.io/
- https://docs.aws.amazon.com/eks/latest/userguide/getting-started.html
- https://www.eksworkshop.com/

Open OnDemand:

- https://openondemand.org/
- https://osc.github.io/ood-documentation/latest/
- https://osc.github.io/ood-documentation/develop/installation/resource-manager/kubernetes.html

Credits

OOD-K8s Project Team:

- Faras Sadek (SEAS)
- Raminder Singh (FASRC)
- Milson Munakami (URCDS)
- Artie Barrett (AT)
- Jeremy Guillette (AT)
- Vesna Tan (AT)
- Francesco Pontiggia (FASRC Alumni)

Questions?

Faras Sadek (fadel@g.harvard.edu) https://seas.harvard.edu/person/faras-sadek
Raminder Singh (r_singh@g.harvard.edu) https://www.rc.fas.harvard.edu/

Plug: FASRC has 3 open positions
https://www.rc.fas.harvard.edu/about/employment/

Reference

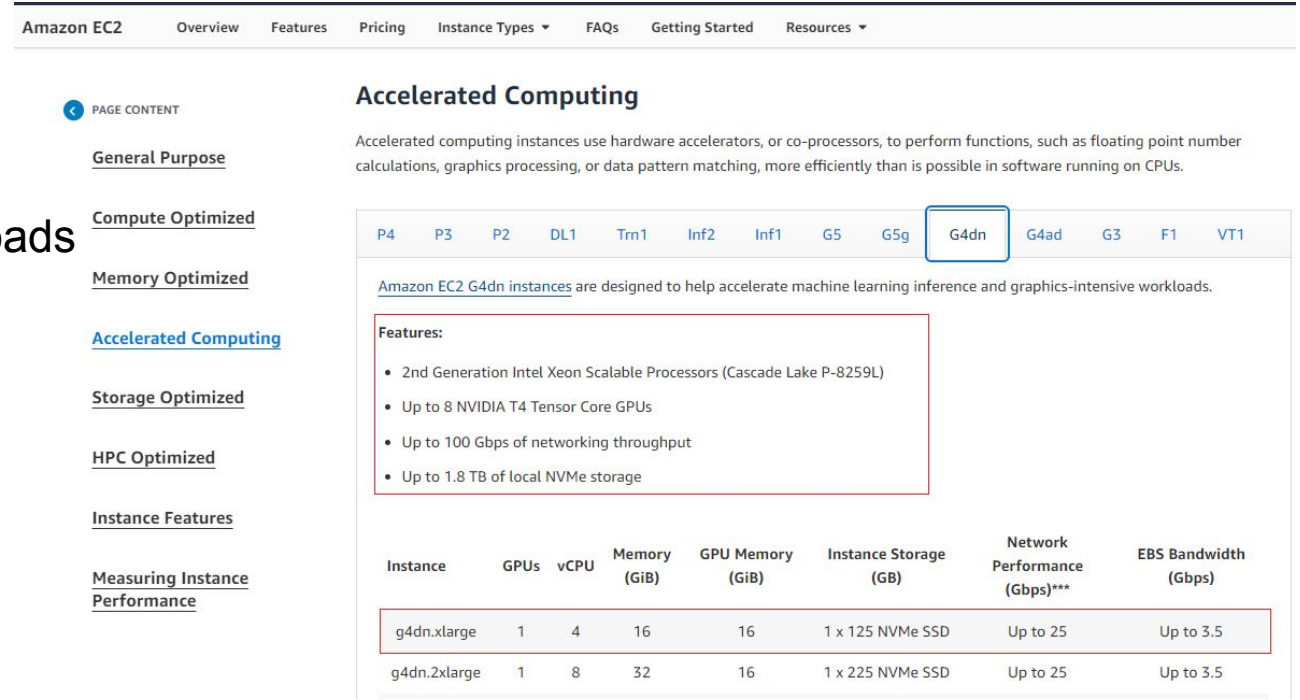
Why Kubernetes (K8s)?

- Cloud-agnostic
  a. Potential to integrate Open OnDemand with other clouds such as Harvard's NERC OpenShift.
  b. Used by HPC centers for similar use cases.
- Extremely flexible and scalable
  a. Provides extensive controls for scheduling, scaling, etc
  b. Has a large ecosystem of plugins i.e. security, monitoring, etc.
- Almost all public cloud providers has **Kubernetes-as-a-service** platform.
- AWS offers a managed service (EKS)
  - AWS manages the cluster control plane.
  - Can take advantage of Elastic Compute Cloud (EC2) for compute and GPUs.
  - Regulatory Compliance

Amazon EKS

g4dn.xlarge

- g4dn.xlarge
- Single GPU VM
- running GPU accelerated workloads

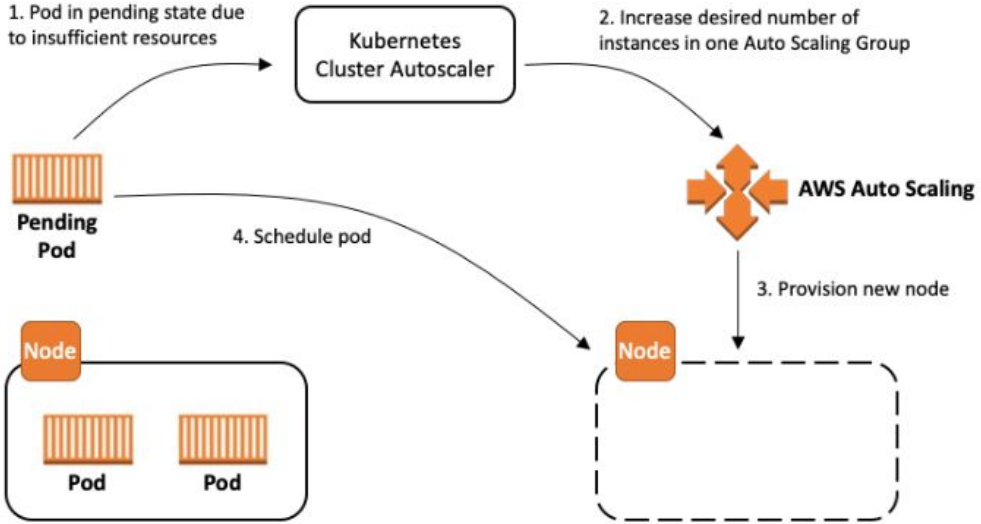


Base EKS-optimized Amazon Machine Image (AMI)

- Packer to bake a custom EKS AMI.
- https://github.huit.harvard.edu/Academic-Computing/OnDemand-eks-base-ami: is configured with HUIT security components prescribed by the HUIT SMVP Policy.
- GPU AMI adds **nvidia-container-runtime** and **nvidia driver** on top of Linux AMI.
- Customized to load all required docker images on AMI to run Jupyter app.

	Instance Size	GPU	vCPUs	Memory (GiB)	Instance Storage (GB)	Network Bandwidth (Gbps)	EBS Bandwidth (Gbps)	On-Demand Price/hr*	1-yr Reserved Instance Effective Hourly* (Linux)	3-yr Reserved Instance Effective Hourly* (Linux)
G4dn										
Single GPU VMs	g4dn.xlarge	1	4	16	1 x 125 NVMe SSD	Up to 25	Up to 3.5	$0.526	$0.316	$0.210

Worker Node Group (GPU or General CPU)




```
Events:
  Type     Reason            Age                  From                Message
  ----     ------            ----                 ----                -------
  Normal   TriggeredScaleUp  6m59s                cluster-autoscaler  pod triggered scale-up: [{eksctl-ood-dev-eks-nodegroup-gpu-nodegroup-NodeGroup-1HHXQ9WKZF33R 0->1 (max: 5)}]
  Warning  FailedScheduling  5m56s (x2 over 7m6s) default-scheduler   0/1 nodes are available: 1 Insufficient cpu, 1 Insufficient memory, 1 Too many pods.
  Warning  FailedScheduling  4m56s                default-scheduler   0/2 nodes are available: 1 Insufficient cpu, 1 Insufficient memory, 1 Too many pods, 1 node(s) had taint {node
.kubernetes.io/not-ready: }, that the pod didn't tolerate.
  Normal   Scheduled         4m21s                default-scheduler   Successfully assigned milson/jupyter-rzys60or to ip-10-140-180-73.ec2.internal
  Normal   Pulling           4m20s                kubelet             Pulling image "huitacademictechnology/ood-k8s-utils"
  Normal   Pulled            4m19s                kubelet             Successfully pulled image "huitacademictechnology/ood-k8s-utils" in 1.439325091s
  Normal   Created           4m19s                kubelet             Created container init-secret
  Normal   Started           4m19s                kubelet             Started container init-secret
  Normal   Pulling           4m10s                kubelet             Pulling image "huitacademictechnology/jupyter-tensorflow-pytorch-gpu"
  Normal   Pulled            4m10s                kubelet             Successfully pulled image "huitacademictechnology/jupyter-tensorflow-pytorch-gpu" in 122.795122ms
  Normal   Created           4m6s                 kubelet             Created container jupyter
  Normal   Started           3m13s                kubelet             Started container jupyter
```