AWS re:Invent

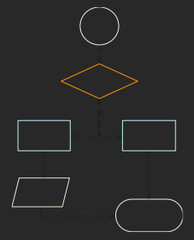
CMP402-R

Setting up and optimizing your HPC cluster on AWS

Francesco Ruffino
Sr. HPC Specialized SA
Amazon Web Services

Pierre-Yves Aquilanti, Ph.D.
Senior HPC Specialized Solutions Architect
Amazon Web Services

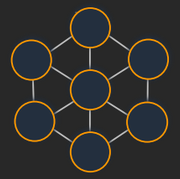
Compute & orchestration building blocks



Workflow, notifications, queues

- Workflow management
- Notification & message queues

Workflow management and communication

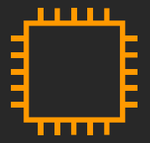


Serverless & containers

- Event-driven functions (AWS Lambda)
- Batch schedulers, containers orchestrators

Provisioning

- Amazon EC2 Auto Scaling groups: scale up & down
- Instance fleets: capacity at scale across AZs



Instances

- Virtualized
- Bare metal

} Different capabilities (CPU, RAM, SSD, network, accelerators)

Abstraction

Focus on the workload and not infrastructure

Compute on events or requests

Base compute layer

Base infrastructure

VPC & subnets

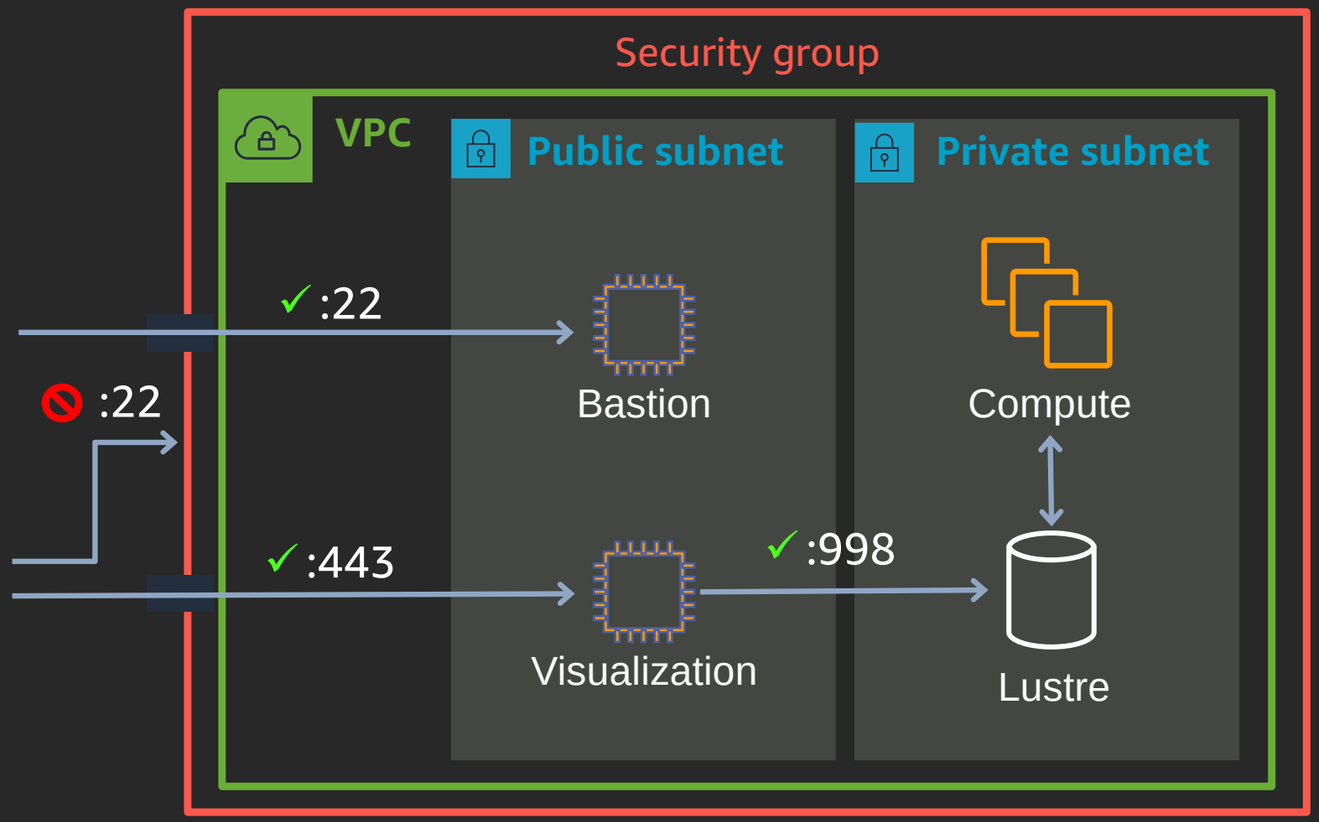
- Virtual Private Cloud: logically section of the cloud provider infrastructure
- Subnet: logical partition of a network

Security groups

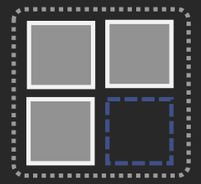
- Virtual firewalls
- VPC & instances

Instances & services

- Instances
- Managed services
- …

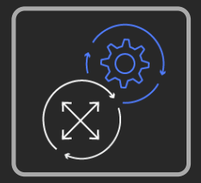


EC2 Auto Scaling in more detail



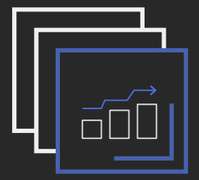
Logical unit

Purpose of scaling or management



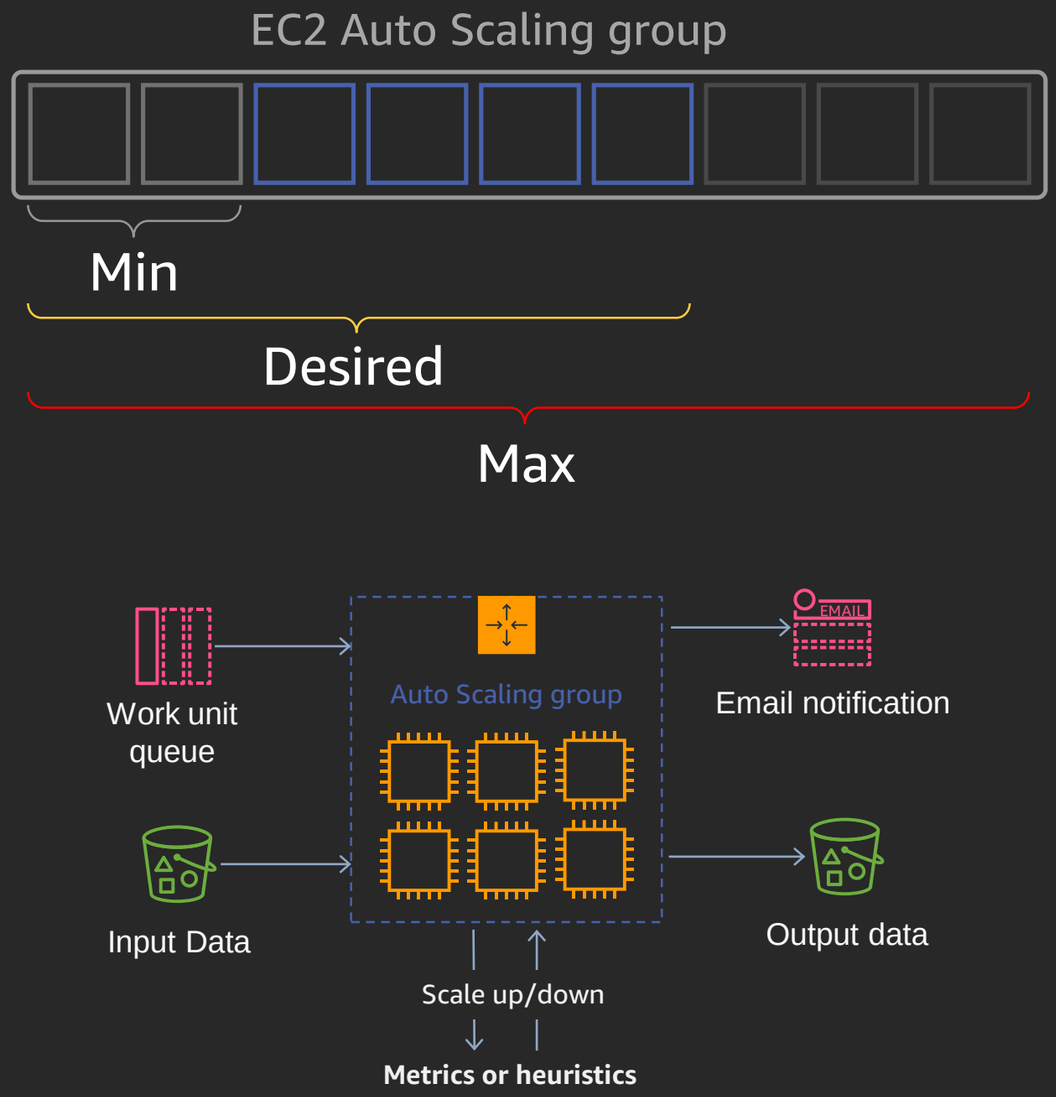
Launch templates

Kind, size, storage, ssh keypair, user data, security groups



Scaling options

- Manual, schedule, predictive
- Notify on start, stop, terminate…

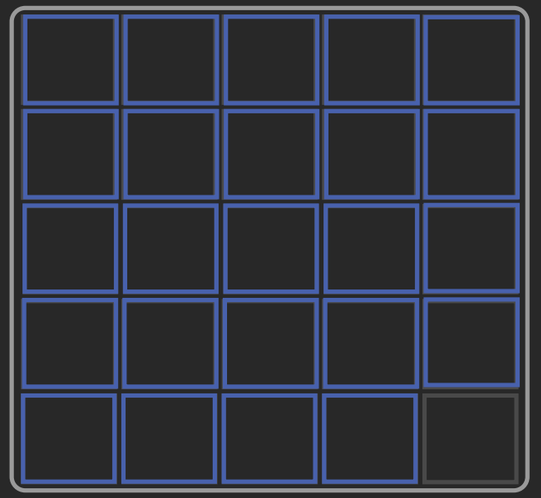


EC2 Auto Scaling compute system?

Imagine that nodes are added when jobs are submitted and removed when they finish

	Start	0 instances in the cluster, 25 max authorized
	Submit	Job 1, request 288 cores, 8 instances
	Provision	8 instances and execute Job 1
	Finalize	Job 1
		Idle 5 min
But after 2 minutes	Submit	Job 2, request 864 cores, 24 instances
	Provision	16 new instances and execute Job 2
	Finalize	Job 2
		Idle 5 min
	Terminate	24 instances
	Stop	0 instances in the cluster

Auto Scaling compute cluster



Min: 0 **Desired: 04** Max: 25

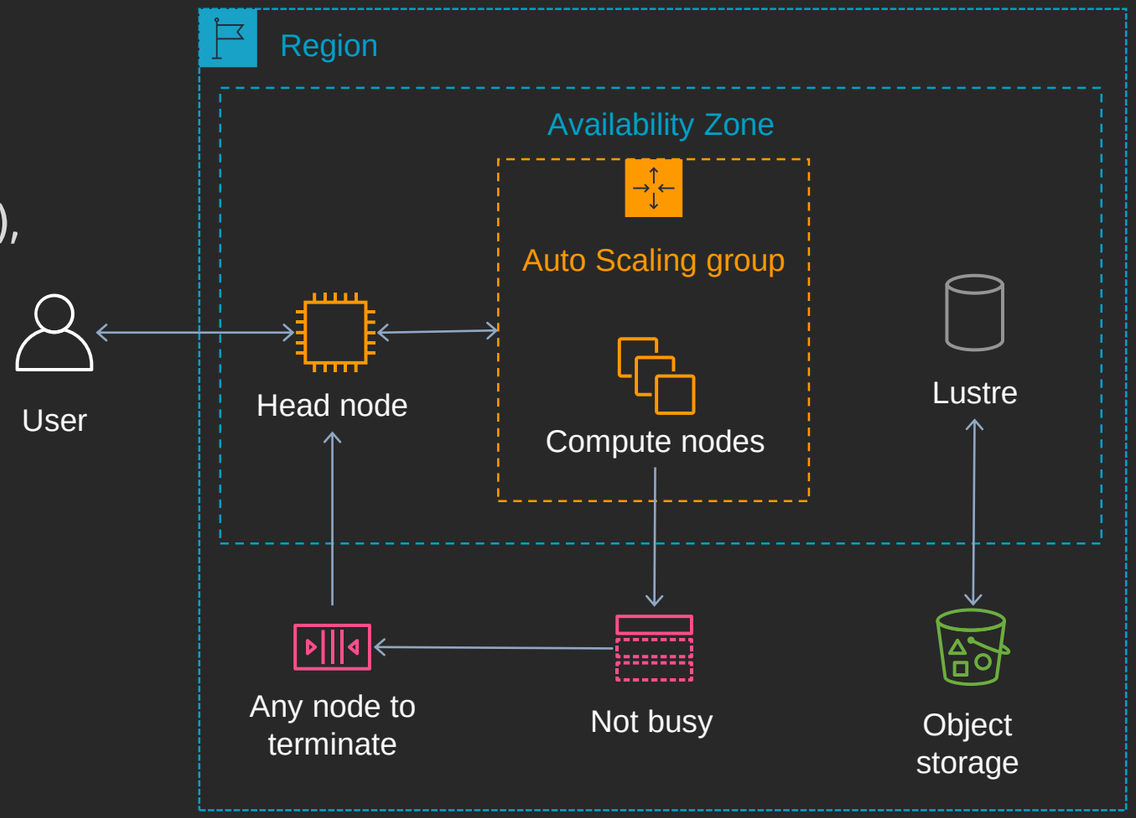
cores / node: 36

```
aws autoscaling delete-auto-scaling-group \
    --auto-scaling-group-name my-asg-scaling-group \
    --desired-capacity 04
    --min-size 1 --max-size 25
    --vpc-zone-identifier "subnet-5ea0c127”
```

Putting it all together

Building an auto-scaling HPC cluster

- Similar to on-premises but with auto-scaling
- Still a classical HPC system with a scheduler (SLURM, SGE…), Lustre, placement groups (tightly coupled)
- The same familiar interface with an elastic capacity



Additional technical considerations

When ready, instances send notifications to a message queuing service. The scheduler watch this queue and add the compute nodes as they appear

When not busy, they will lock themselves up, check the scheduler queue, send a notification and terminate.

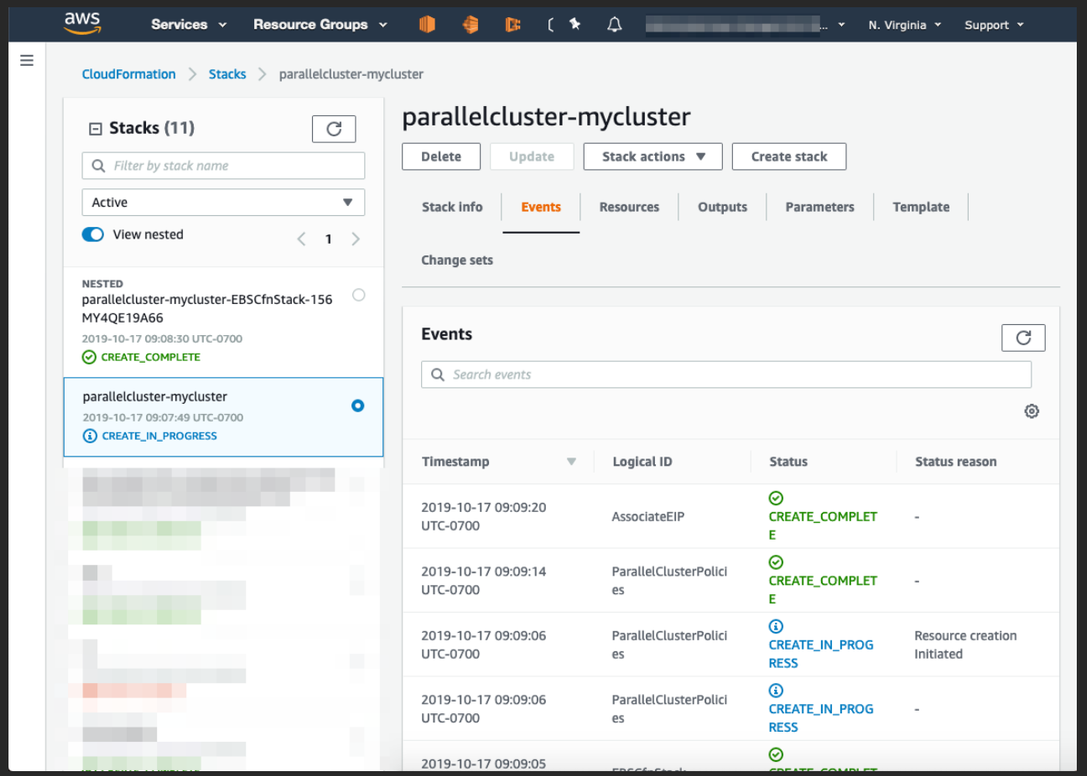
Part of the next hands-on

Launching a cluster in minutes

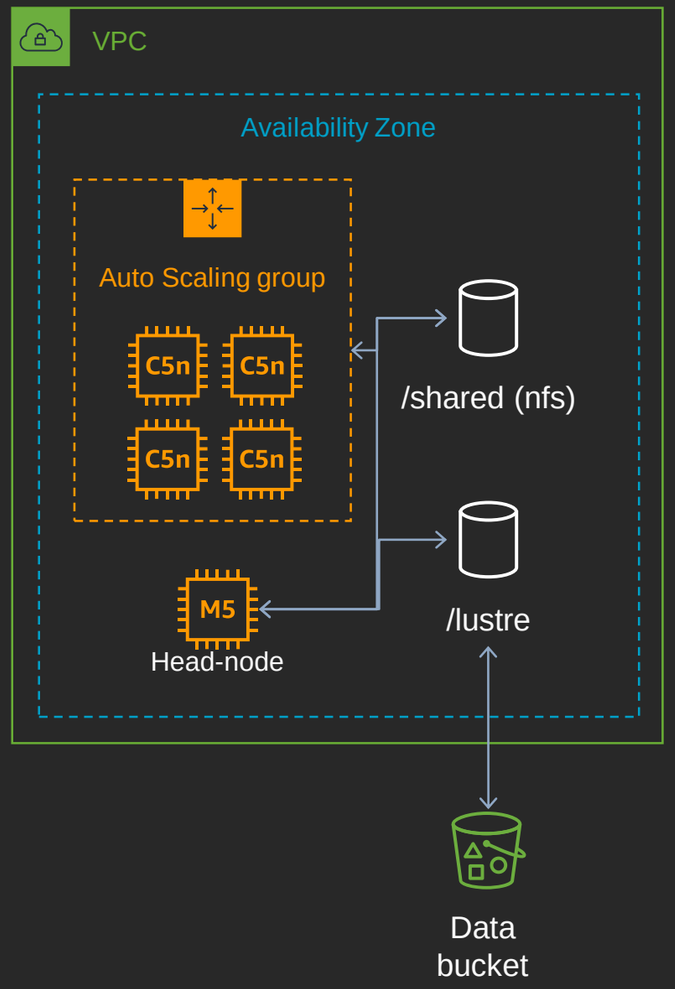
- Key commands to manage a cluster

```
$ pcluster create mycluster

$ pcluster update mycluster

$ pcluster stop mycluster

$ pcluster start mycluster

$ pcluster delete mycluster
```

- AWS CloudFormation does the heavy lifting on your behalf



Simple architecture



- Post install configuration
  - Install applications scripts
- Amazon Elastic Block Store (Amazon
- EBS) Snapshot bootstrap
  - Application installations or static configurations
- Other details
  - Amazon Elastic File System (Amazon EFS) can be shared across clusters
  - Lustre partition can be mounted but per AZ
  - Public/private subnets for head/compute
  - Link to AD for user mapping if required

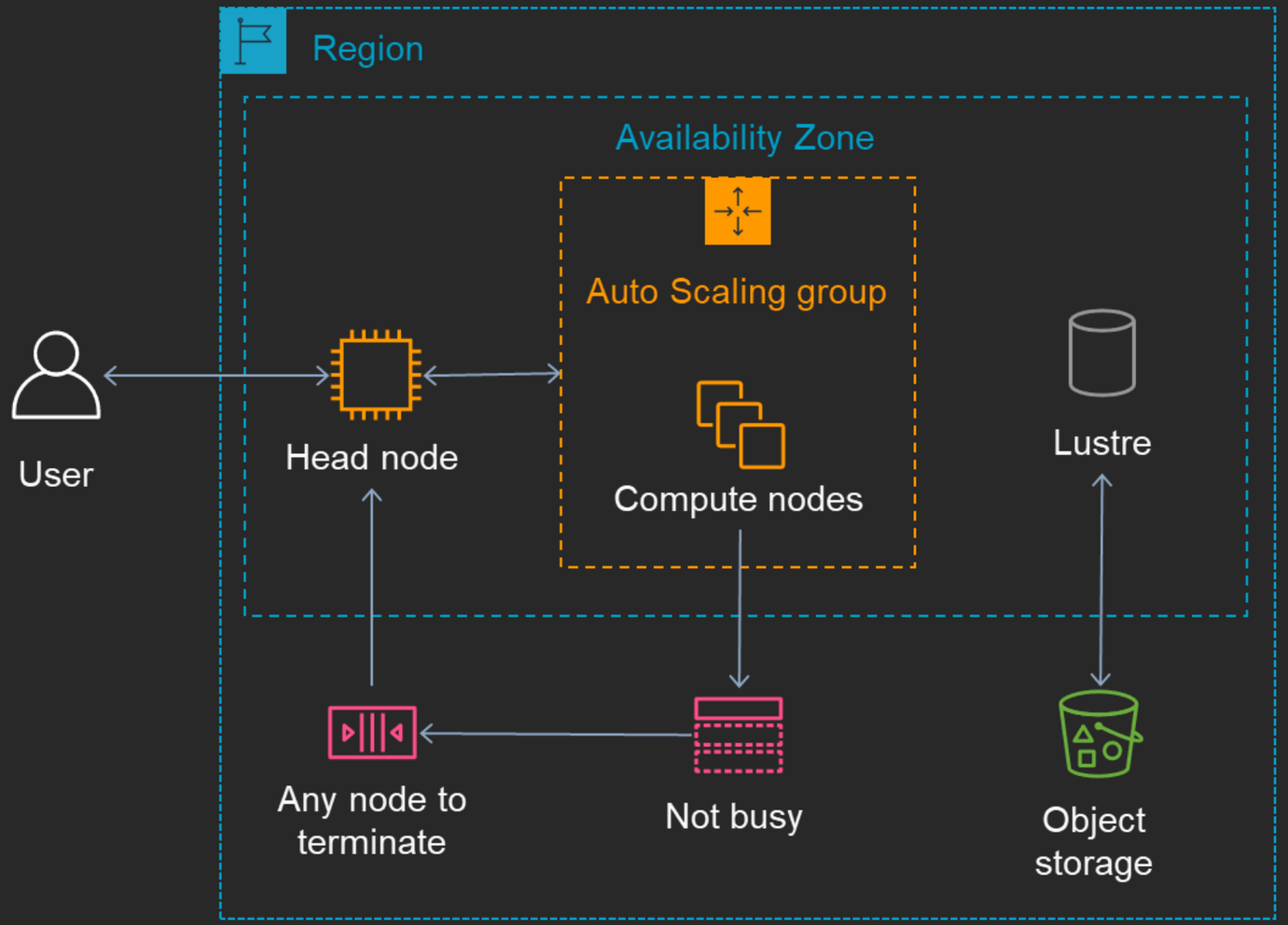
Example of configuration file

```
[aws]
aws_region_name = ${REGION}

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

[cluster default]
key_name = key-pair-name
vpc_settings = public
ebs_settings = myebs
compute_instance_type = c4.xlarge
master_instance_type = c4.xlarge
cluster_type = ondemand
placement_group = DYNAMIC
placement = compute
min_queue_size = 0
max_queue_size = 8
initial_queue_size = 0
disable_hyperthreading = true
scheduler = slurm
```

```
[vpc public]
vpc_id = ${VPC_ID}
master_subnet_id = ${SUBNET_ID}

[ebs myebs]
shared_dir = /shared
volume_type = gp2
volume_size = 20

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

[fsx myfsx]
shared_dir = /lustre
storage_capacity = 3600
import_path = s3://mybucket
```

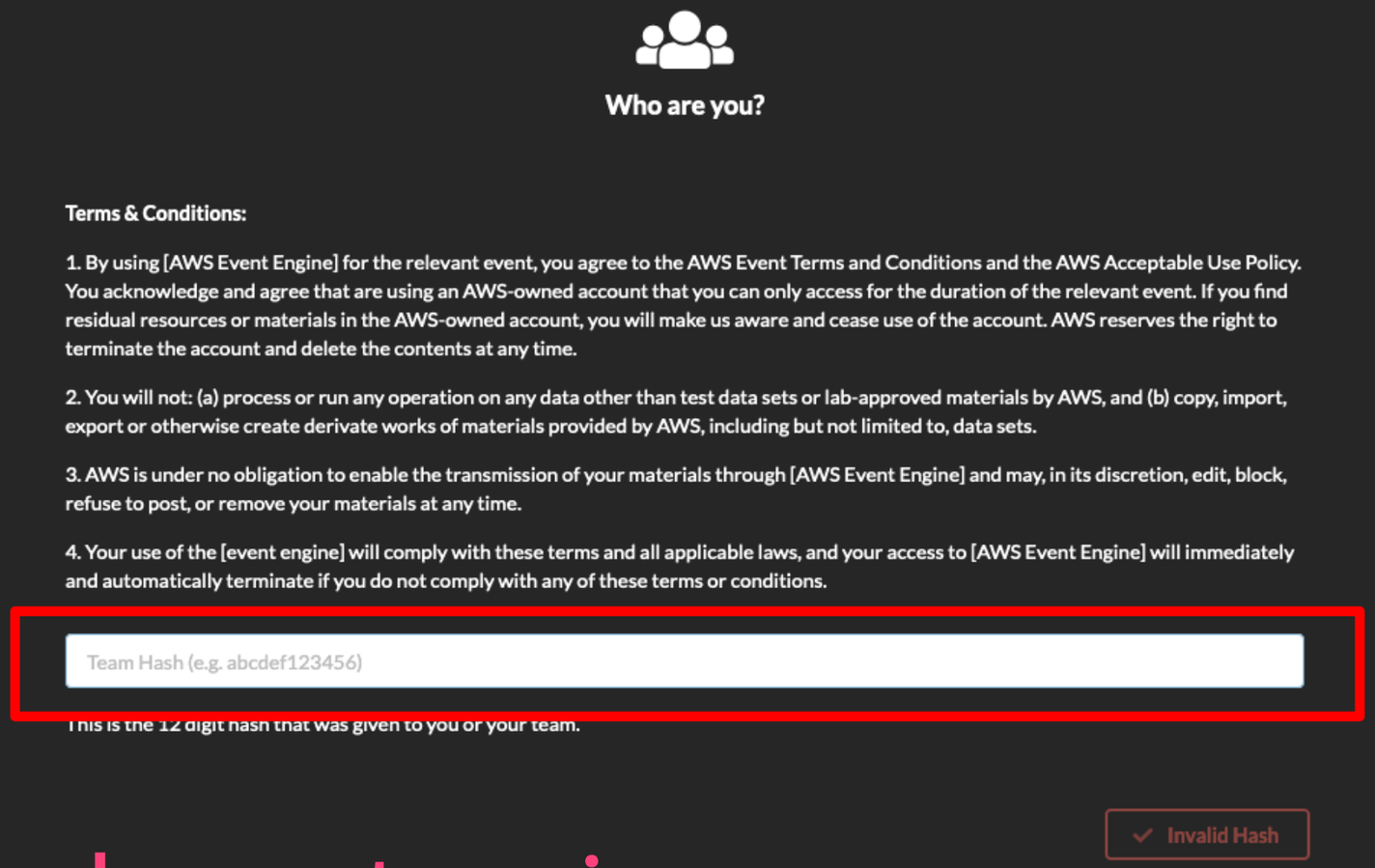
Hands-on

Review for the lab



What you will build today

Use the 12-character hash to log in to your account



https://dashboard.eventengine.run

What you will build today

Objectives for this lab

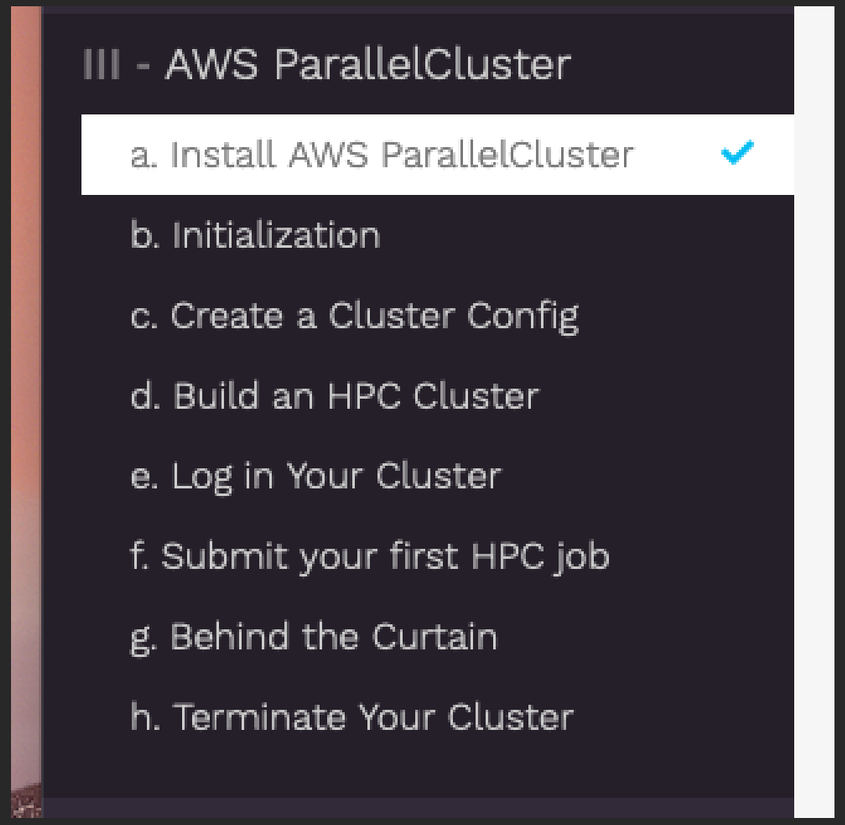
1. Install AWS ParallelCluster and configure it
2. Create a cluster and connect to it
3. Run an application
4. Tear down the cluster

Login: your 12-character hash

https://dashboard.eventengine.run

Lab: Section III

http://bit.ly/aws-hpc



Western Digital Corporation

Hiroshi Kobayashi
Sr. Solutions Architect – Global Engineering Services- WDC

AWS re:Invent

aws

Why cloud?

Storage everywhere

- Smartphone
- Mobility
- Radio telescopes on the top of the mountains

→ Various environment conditions

Silicon-to-system engineering

- Run millions of simulations in:
  - Material level
  - Device level
  - System level

Cloud’s scalability enables us to explore this huge design space

The world’s first image of a black hole
Image credit: Event Horizon Telescope Collaboration

Western Digital’s unique ability to design, tune, and optimize across the entire portfolio technology stack

Pipeline

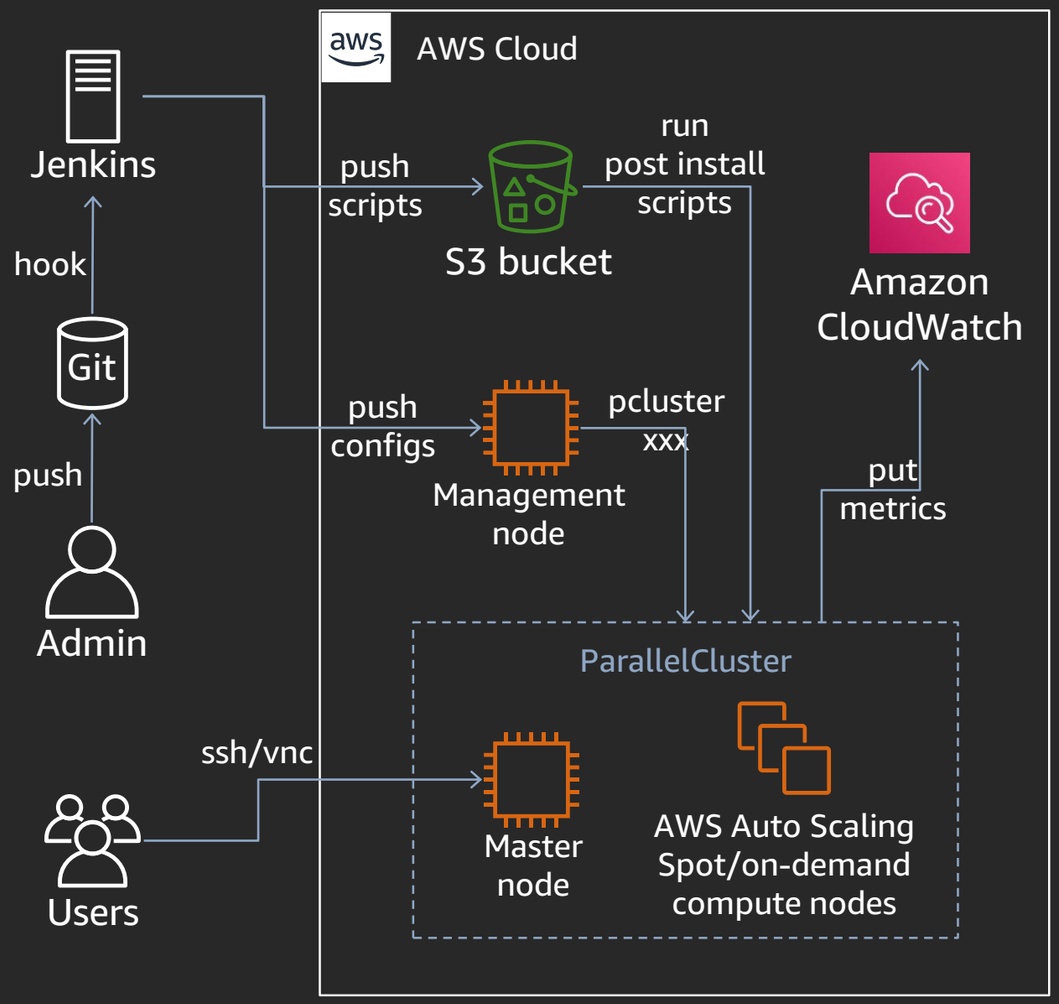
A pipeline for quick cluster optimization and its records

Git base operation

- All changes are recorded in Git
- Cluster config files → management node
- Custom bootstrap scripts → Amazon S3

Python virtual environments

- AWS ParallelCluster development is very active
- Mixed version of clusters



Custom bootstrap

- AWS ParallelCluster can execute arbitrary code either before (pre-install) or after (post-install) the main bootstrap action
- Main post-install script calls actual setup scripts

  `post_install = s3://<bucket-name>/projects/mycluster/scripts/00-cluster-init.sh`
- Differentiate between master and compute nodes execution by sourcing **`/etc/parallelcluster/cfnconfig`** file and evaluating **`cfn_node_type`** env var

```
# import parameters
CLSTINI_DIR="/shared/cluster-init"

# call scripts
mkdir -p ${CLSTINI_DIR}/run
aws s3 sync s3://<bucket-name>/projects/mycluster/scripts/${CLSTINI_DIR}/run
${CLSTINI_DIR}/run/01-disable-ht.sh
${CLSTINI_DIR}/run/02-createuser-user.sh
${CLSTINI_DIR}/run/03-install-intel-compiler.sh
```

```
# import parameters
. /etc/parallelcluster/cfnconfig
CLSTINI_DIR="/shared/cluster-init"
APPNM="parallel_studio_xe_2019_update4_cluster_edition"

#install intel compiler
if [[ ${cfn_node_type} =~ "MasterServer" ]]; then
  aws s3 sync s3://<bucket-name>/installers/intel ${CLSTINI_DIR}/icomp_install
  tar -C ${CLSTINI_DIR}/icomp_install -xf ${CLSTINI_DIR}/icomp_install/${APPNM}.tgz
  pushd ${CLSTINI_DIR}/icomp_install/${APPNM}
  ./install.sh --silent intel_silent_2019.cfg
  popd
fi
```

Monitoring & backup

Monitoring

- Compute node GPU usage
- Shared storage usage
- SGE active/dead node
- Installed and configured in post-install script

Amazon CloudWatch

Backup

- Fully utilizing AWS Backup
- Tag base backup
- Tags were added in post-install script

AWS Backup

Thank you!

Anh Tran
trnh@

Pierre-Yves Aquilanti, Ph.D.
pierreya@

AWS re:Invent

aws

Please complete the session survey in the mobile app.