2025 전국기능경기대회 채점기준

1. 채점 상의 유의사항	직 종 명	클라우드컴퓨팅

※ 다음 사항을 유의하여 채점하시오.

1) AWS의 지역은 ap-northeast-2을 사용합니다.
2) 웹페이지 접근은 크롬이나 파이어폭스를 이용합니다.
3) 웹페이지에서 언어에 따라 문구가 다르게 보일 수 있습니다.
4) shell에서의 명령어의 출력은 버전에 따라 조금 다를 수 있습니다.
5) 문제지와 채점지에 있는 <> 는 변수입니다. 해당 부분을 변경해 입력합니다.
6) 채점은 문항 순서대로 진행해야 합니다.
7) 삭제된 채점자료는 되돌릴 수 없음으로 유의하여 진행하며, 이의신청까지 완료 이후 선수가 생성한 클라우드 리소스를 삭제합니다.
8) 부분 점수가 있는 문항은 채점 항목에 부분 점수가 적혀져 있습니다.
9) 부분 점수가 따로 없는 문항은 모두 맞아야 점수로 인정됩니다.
10) 리소스의 정보를 읽어오는 채점항목은 기본적으로 스크립트 결과를 통해 채점을 진행하며, 만약 선수가 이의가 있다면 명령어를 직접 입력하여 확인해볼 수 있습니다.
11) [ ] 기호는 채점에 영향을 주지 않습니다.
12) 채점 내용의 $ 기호는 명령어에 포함되는 것이 아니라 쉘을 의미합니다.

2. 채점기준표

1) 주요항목별 배점	직 종 명	클라우드컴퓨팅

과제 번호	일련 번호	주요항목	배점	채점방법		채점시기		비고
				독립	합의	경기 진행중	경기 종료후	
제3과제	1-1	IP 주소 부족 문제	5		○		○	
	1-2	Security Group for Pods	5		○		○	
	2	Pod Identity	5		○		○	
	3	Graceful Node Shutdown	5		○		○	
합계			20					

2) 채점방법 및 기준

과제 번호	일련 번호	주요항목	일련 번호	세부항목(채점방법)	배점
3과제	1-1	IP 주소 부족 문제	1	wsi-cluster 클러스터 생성	0.5
			2	Nodegroup 생성	0.5
			3	VPC CNI 환경 변수 확인	2
			4	200개의 pod status 확인	2
	1-2	Security Group for Pods	1	VPC CNI 환경 변수 확인	1
			2	Pod 보안그룹과 노드 보안 그룹 확인	1.5
			3	pod-sg-policy.yaml 파일 확인	1
			4	Cross-VPC 통신 테스트	1.5
	2	Pod Identity	1	EBS CSI Add-on 설치 확인	1
			2	Pod Identity IAM Role 확인	1
			3	Pod Identity Association 확인	1
			4	PVC bound 확인	1
			5	Pod status 확인	1
	3	Graceful Node Shutdown	1	Nodegroup 생성	1
			2	NTH 설치 확인	1
			3	FIS Experiment 수행 확인	1
			4	Busy-app 재스케줄링 확인	2
	총점				20

3) 채점내용

순번	사전준비
0	1) bastion 명령어 및 권한 확인 (awscli, ,kubectl, curl, helm, eksctl, yq, jq, permission, region)

순번	사전준비
1-1-(1)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어 입력 후 'wsi-cluster'가 출력되는지 확인합니다. **$ aws eks describe-cluster --name $CLUSTER_NAME --region $REGION --query 'cluster.name'** 3) 아래 명령어 입력 후 wsi-main-vpc가 출력되는 것을 확인함으로써, 클러스터가 해당 VPC 내에 생성되었는지 확인합니다. **$ VPC_ID=$(aws eks describe-cluster --name $CLUSTER_NAME --region $REGION --query "cluster.resourcesVpcConfig.vpcId" --output text)** **$ ACTUAL_VPC_NAME=$(aws ec2 describe-tags --filters "Name=resource-id,Values=$VPC_ID" "Name=key,Values=Name" --region $REGION --query "Tags[0].Value" --output text)** **$ echo $ACTUAL_VPC_NAME**
1-1-(2)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어 입력 후 wsi-main-nodegroup 이름의 노드 그룹이 존재하는지 확인합니다. **$ aws eks describe-nodegroup --cluster-name $CLUSTER_NAME --nodegroup-name wsi-main-nodegroup --region $REGION --query 'nodegroup.nodegroupName'** 3) 아래 명령어와 반복문 및 if문을 이용하여 **"[PASS] Private Subnet"**가 출력되는지 확인합니다. **$ aws eks describe-nodegroup --cluster-name $CLUSTER_NAME --nodegroup-name wsi-main-nodegroup --region $REGION --query 'nodegroup.subnets'** **$ aws ec2 describe-subnets --subnet-ids "$subnet" --query "Subnets[0].[VpcId, MapPublicIpOnLaunch]" --output text** 4) 아래 명령어 입력 후 { "minSize": 3, "maxSize": 3, "desiredSize": 3 }을 만족하는지 확인합니다. **$ aws eks describe-nodegroup --cluster-name $CLUSTER_NAME --nodegroup-name $NODEGROUP_NAME --region $REGION --query 'nodegroup.scalingConfig'**
1-1-(3)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어 입력 후 ENABLE_PREFIX_DELEGATION 값이 true인지 확인합니다. **$ kubectl describe daemonset aws-node -n kube-system | grep -I ENABLE_PREFIX_DELEGATION**

1-1-(4)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 이용하여Total Running Pods가 200개 인지 확인합니다. **$ kubectl get pods --all-namespaces -l app=my-pod --field-selector=status.phase=Running --no-headers | wc -l** **$ kubectl get pods --all-namespaces -l app=backup-pod --field-selector=status.phase=Running --no-headers | wc -l**
1-2-(1)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 입력 후ENABLE_POD_ENI 값이 true인지 확인합니다. **$ kubectl describe daemonset aws-node -n kube-system | grep -I ENABLE_POD_ENI**
1-2-(2)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 입력 후backup-pod-sg 이름의 보안그룹이 있는지 확인합니다. **$ aws ec2 describe-security-groups --filters Name=group-name,Values=backup-pod-sg --region $REGION** 3) 아래 명령어와 if문을 이용하여 wsi-backup-vpc 내 file-server EC2 보안그룹의 inbound rule이 80번 포트에서 backup-pod-sg를 허용하는지 확인합니다. **$ aws ec2 describe-security-groups --group-ids $FILE_SERVER_SG_ID --region $REGION --query "SecurityGroups[0].IpPermissions[?ToPort==\`80\` && IpRanges==[] && UserIdGroupPairs[?GroupId=='$BACKUP_POD_SG_ID']]" --output json**

1-2-(3)	1) SSH를 통해 Bastion 서버에 접속합니다.
	2) 아래 명령어를 이용하여 pod-sg-policy.yaml 에 spec.securityGroups.groupIds값이 추가되었는지 확인하고, 해당 값이 backup-pod-sg의 id값과 동일한지 확인합니다.
	$ yq e '.spec.securityGroups.groupIds[0]' "pod-sg-policy.yaml"
	cf) pod-sg-policy.yaml
	`apiVersion: vpcresources.k8s.aws/v1beta1`
	`kind: SecurityGroupPolicy`
	`metadata:`
	`  name: backup-pod-sg-policy`
	`spec:`
	`  podSelector:`
	`    matchLabels:`
	`      app: backup-pod`
	`  securityGroups:`
	`    groupIds:`
	`      - < pod security group id >`

1-2-(4)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 입력 후 Cross-VPC 통신을 잘 수행하는지 확인합니다. **$ BACKUP_POD_NAME=$(kubectl get pod --all-namespaces -l app=backup-pod -o jsonpath="{.items[0].metadata.name}")** **$ FILE_SERVER_IP=$(aws ec2 describe-instances \ --region "$REGION" \ --filters "Name=tag:Name,Values=backup-file-server" \ --query "Reservations[*].Instances[*].PrivateIpAddress" \ --output text)** **$ kubectl exec -it $BACKUP_POD_NAME -- curl -s --max-time 3 http://$FILE_SERVER_IP** 3) 아래 명령어를 입력하여 199개 중 하나의 pod를 랜덤으로 선택하여 접속한 후, curl 명령어를 수행하였을 때 connection error가 발생하는지 확인합니다. 명령어 입력 후 **"[FAIL] 연결 실패: $SELECTED_POD → $FILE_SERVER_IP"**가 출력되어야 합니다. **$ PODS=($(kubectl get pod -l "app=my-pod" -o jsonpath='{.items[*].metadata.name}'))** **$ SELECTED_POD=${PODS[$RANDOM % ${#PODS[@]}]}** **$ RESPONSE=$(kubectl exec -it $SELECTED_POD -- curl http://$FILE_SERVER_IP)**
2-(1)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 입력 후EBS CSI Add-on Pod 가 잘 생성되었는지 확인합니다. **$ kubectl get pods -n kube-system -l app=ebs-csi-controller --field-selector=status.phase=Running --no-headers | wc -l**
2-(2)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 입력하여 AmazonEKS-PodIdentity-EBS-CSI-DriverRole라는 이름의 IAM role이 존재하는지 확인합니다. **$ aws iam get-role --role-name AmazonEKS-PodIdentity-EBS-CSI-DriverRole --region $REGION** 3) 아래 명령어와 if문을 이용하여 AmazonEKS-PodIdentity-EBS-CSI-DriverRole 에 AmazonEBSCSIDriverPolicy 정책이 연결되어 있는지 확인합니다. **$ aws iam list-attached-role-policies --role-name "$ROLE_NAME" --query "AttachedPolicies[?PolicyName=='$POLICY_NAME']" --output json**

2-(3)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어와 if문을 이용하여 Pod Identity Association이 존재하는지 확인합니다. **$ ASSOCIATIONS=$(aws eks list-pod-identity-associations --cluster-name "$CLUSTER_NAME" --region "$REGION" --query "associations" --output json)** 3) 아래 명령어와 if문을 이용하여 Pod Identity associations의 Service account가 ebs-csi-controller-sa인지 확인합니다. **$ MATCHED_SA=$(echo "$ASSOCIATIONS" | jq -r ".[] | select(.serviceAccount==\"$EXPECTED_SA\")")**
2-(4)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 입력하여 ebs-claim 이름의 pvc가 Bound 상태인지 확인합니다. **$ kubectl get pvc ebs-claim -o jsonpath="{.status.phase}"**
2-(5)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 입력하여 app이름의 pod가 Running 상태인지 확인합니다. **$ kubectl get pod app -o jsonpath='{.status.phase}'**
3-(1)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어를 이용하여 **wsi-self-managed-ng** 라는 label을 갖는 자체 관리형 노드 그룹이 존재하는지 확인합니다. **$ SELF_MANAGED_NG="wsi-self-managed-ng"** **$ kubectl get nodes -l nodegroup=$SELF_MANAGED_NG --no-headers \| wc -l** 3) 아래 명령어와 반복문 및 if문을 이용하여 해당 label을 갖는 모든 노드를 배열로 가져오고 각 노드의 subnet이 public subnet인지 확인합니다. **$ kubectl get nodes -l nodegroup=$SELF_MANAGED_NG -o jsonpath='{.items[*].metadata.name}'** **$ kubectl get node $NODE -o jsonpath='{.spec.providerID}' \| cut -d'/' -f5** **$ aws ec2 describe-instances ** **--instance-ids $INSTANCE_ID ** **--region $REGION \\** **--query "Reservations[0].Instances[0].SubnetId" \\** **--output text** **$ aws ec2 describe-subnets --subnet-ids "$SUBNET_ID" **

	--region $REGION ** **--query "Subnets[0].MapPublicIpOnLaunch" ** **--output text
3-(2)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) **kubectl get daemonset aws-node-termination-handler -n kube-system** 명령어를 입력하여 Node Termination Handler가 설치되었는지 확인합니다. cf) kubectl get daemonset aws-node-termination-handler -n kube-system 입력 시 출력 결과 NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR aws-node-termination-handler 5 5 5 5 5 kubernetes.io/os=linux
3-(3)	1) SSH를 통해 Bastion 서버에 접속합니다. 2) 아래 명령어와 if문을 이용하여 completed 상태인 FIS Experiment가 존재하는지 확인합니다. **$ aws fis list-experiments ** **--region "$REGION" ** **--query "experiments[?state.status=='completed'] \| [-1].id"** 3) 아래 명령어를 이용하여 experiment 구성이 문제와 같이 action 이 send-spot-instance-interruptions인지 확인합니다. **aws fis get-experiment --region "$REGION" --id "$EXPERIMENT_ID" --output json**
3-(4)	1) SSH를 통해 Bastion 서버에 접속합니다 2) 아래 명령어를 이용하여busy-app이 running 상태인지 확인하고 Event 정보를 가져옵니다. **BUSY_POD=$(kubectl get pods -l app=busy-app -o jsonpath='{.items[0].metadata.name}')** **$kubectl get pod $BUSY_POD** **$kubectl describe pod $BUSY_POD** 3) 출력 내용 중FailedAttachVolume된 event와, SuccessfulAttachVolume 된 event의 Age 필드를 확인합니다. 4) event의 Age 차이가 2분 미만이면 NTH에 의해 정상적으로 rescheduling된 pod로 간주합니다.