

2022 경상남도 제57회 전국기능경기대회 채점기준

| 1. 채점상의 유의사항 | 직 종 명 | 클라우드컴퓨팅 |
|---|-------|---------|
| <p>※ 다음 사항을 유의하여 채점하십시오.</p> <ol style="list-style-type: none"> 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. | | |

2. 채점기준표

| 1) 주요항목별 배점 | | | 직 종 명 | | 클라우드컴퓨팅 | | | |
|-------------|----------|---------|-------|------|---------|-----------|-----------|----|
| 과제 번호 | 일련 번호 | 주요항목 | 배점 | 채점방법 | | 채점시기 | | 비고 |
| | | | | 독립 | 합의 | 경기 진행중 | 경기 종료후 | |
| 제1과제 | 1 | 네트워크 구성 | 2.7 | | ○ | | ○ | |
| | 2 | 어플리케이션 | 5.3 | | ○ | | ○ | |
| | 3 | 컨테이너라이징 | 3.5 | | ○ | | ○ | |
| | 4 | 쿠버네티스 | 6 | | ○ | | ○ | |
| | 5 | 배포 | 6.5 | | ○ | | ○ | |
| | 6 | 보안설정 | 5.6 | | ○ | | ○ | |
| | 7 | LB | 4.6 | | ○ | | ○ | |
| | 8 | 배치 | 3 | | ○ | | ○ | |
| | 9 | 로드테스트 | 1.5 | | ○ | | ○ | |
| | 10 | 모니터링 | 1.3 | | ○ | | ○ | |
| 합 계 | | | 40 | | | | | |

2) 채점방법 및 기준

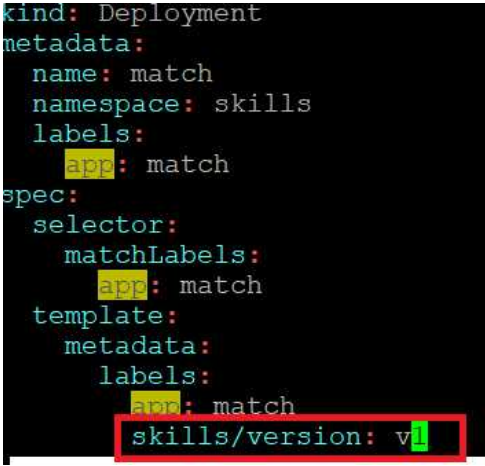
| 과제 번호 | 일련 번호 | 주요항목 | 일련 번호 | 세부항목(채점방법) | 배점 |
|----------|----------|---------|----------|------------------------|-----|
| 1과제 | 1 | 네트워크 구성 | 1 | VPC | 1.2 |
| | | | 2 | 서브넷 | 1.5 |
| | 2 | 어플리케이션 | 1 | Match API positive | 1.3 |
| | | | 2 | Match API negative | 1.6 |
| | | | 3 | Stress stress API | 1.4 |
| | | | 4 | Stress random API | 1.0 |
| | 3 | 컨테이너라이징 | 1 | ECR | 0.9 |
| | | | 2 | Docker user | 1.5 |
| | | | 3 | Docker image | 1.1 |
| | 4 | 쿠버네티스 | 1 | EKS Cluster | 1.5 |
| | | | 2 | EKS logging | 1.5 |
| | | | 3 | EKS Node Group scale | 1.5 |
| | | | 4 | EKS Node Group subnets | 1.5 |
| | 5 | 배포 | 1 | Deploy match container | 1.4 |
| | | | 2 | Deploy error test | 5.1 |
| | 6 | 보안설정 | 1 | Match ext access | 1.3 |
| | | | 2 | Match deny rule | 1.5 |
| | | | 3 | Stress ext access | 1.3 |
| | | | 4 | Stress deny rule | 1.5 |
| | 7 | LB | 1 | Match LB | 0.8 |
| | | | 2 | Match Routing | 1.5 |
| | | | 3 | Stress LB | 0.8 |
| | | | 4 | Stress routing | 1.5 |
| | 8 | 배치 | 1 | App placement | 1.5 |
| | | | 2 | Addon placement | 1.5 |
| | 9 | 로드테스트 | 1 | Stress aging | 1.5 |
| | 10 | 모니터링 | 1 | Worker dashboard | 0.6 |
| | | | 2 | match dashboard | 0.7 |
| | 총점 | | | | |

3) 채점내용

| 순번 | 사전준비 |
|----|---|
| 0 | 1) bastion 명령어 및 권한 확인(kubectl, awscli permission, jq, curl, awscli region) 2) marking 스크립트들을 /root/marking에 다운로드 합니다. |

| 순번 | 채점 항목 |
|-----|--|
| 1-1 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력합니다.</p> <pre>aws ec2 describe-vpcs --filter Name=tag:Name,Values=skills-vpc --query "Vpcs[].CidrBlock"</pre> <p>3) 10.0.0.0/16이 출력되는지 확인합니다.</p> |
| 1-2 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어 입력</p> <pre>aws ec2 describe-route-tables --filter Name=tag:Name,Values=skills-private-b-rt --query "RouteTables[].Routes[].NatGatewayId"</pre> <p>3) "nat-" 로 시작하는 문구가 출력되는지 확인합니다.</p> |
| 2-1 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어 입력하여 출력되는 Pod 이름을 기록합니다.</p> <pre>kubectl -n skills get pods grep match head -n 1</pre> <p>3) 아래 명령으로 해당 Pod에 접속합니다. (pod 이름은 다름)</p> <pre>kubectl exec -n skills -it match-xx12345688-xxx111 sh</pre> <p>4) 아래 명령으로 {"status": "OK"} 가 출력되는지 확인합니다.</p> <pre>curl http://localhost:8080/v1/match?token=cccccccc</pre> |
| 2-2 | <p>1) 2-1 의 match pod 접속 상태를 유지합니다.</p> <p>2) 아래 명령으로 {"status": "FAIL"} 가 출력되는지 확인합니다.</p> <pre>curl http://localhost:8080/v1/match?token=11131111</pre> |
| 2-3 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어 입력하여 출력되는 Pod 이름을 기록합니다.</p> <pre>kubectl -n skills get pods grep stress head -n 1</pre> <p>3) 아래 명령으로 해당 Pod에 접속합니다. (pod 이름은 다름)</p> <pre>kubectl exec -n skills -it stress-xx12345688-xxx111 sh</pre> <p>4) 아래 명령으로 {"status": "OK"} 가 출력되는지 확인합니다.</p> <pre>curl http://localhost:8080/v1/stress</pre> |
| 2-4 | <p>1) 2-3의 stress pod 접속 상태를 유지합니다.</p> <p>2) 아래 명령으로 {"status": "OK"} 가 출력되는지 확인합니다. (최대 40초까지 걸리 수 있음)</p> <pre>curl http://localhost:8080/v1/random</pre> |

| 순번 | 채점 항목 |
|-----|---|
| 3-1 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력해 match-ecr이 출력되는지 확인합니다.</p> <pre>aws ecr describe-repositories --repository-name match-ecr --query ` "repositories[].repositoryName" 명령어를 입력합니다.</pre> |
| 3-2 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어 입력하여 출력되는 Pod 이름을 기록합니다.</p> <pre>kubectl -n skills get pods grep stress head -n 1</pre> <p>3) 아래 명령으로 해당 Pod에 접속합니다. (pod 이름은 다름)</p> <pre>kubectl exec -n skills -it stress-xx12345688-xxx111 sh</pre> <p>4) 아래 명령어를 입력하여 stress라고 출력되는지 확인합니다.</p> <pre>whoami</pre> |
| 3-3 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력하여 latest라고 출력되는 것이 있는지 확인합니다.</p> <pre>aws ecr list-images --repository-name stress-ecr --query "imageIds[].imageTag"</pre> |
| 4-1 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력하여 1.22가 출력되는지 확인합니다.</p> <pre>aws eks describe-cluster --name skills-cluster --query "cluster.version"</pre> |
| 4-2 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력합니다.</p> <pre>aws eks describe-cluster --name skills-cluster --query "cluster.logging.clusterLogging"</pre> <p>3) 결과값이 아래와 같이 enabled: true이며, types에 5가지가 모두 포함되어 있는지 확인합니다.</p> <pre>"enabled": true, "types": ["api", "audit", "authenticator", "controllerManager", "scheduler"]</pre> |

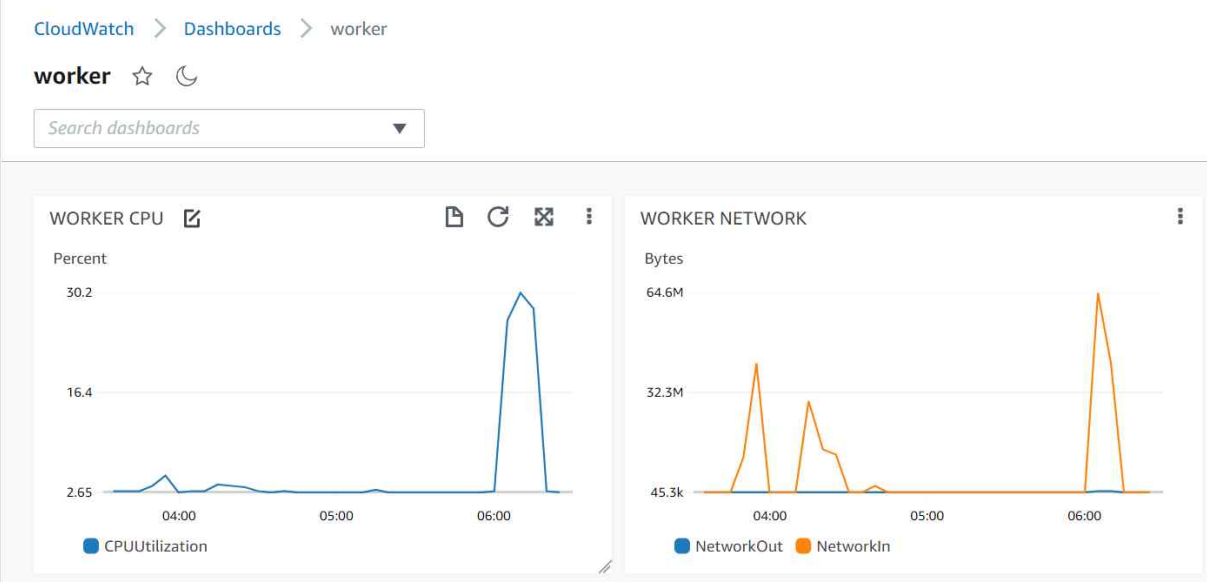
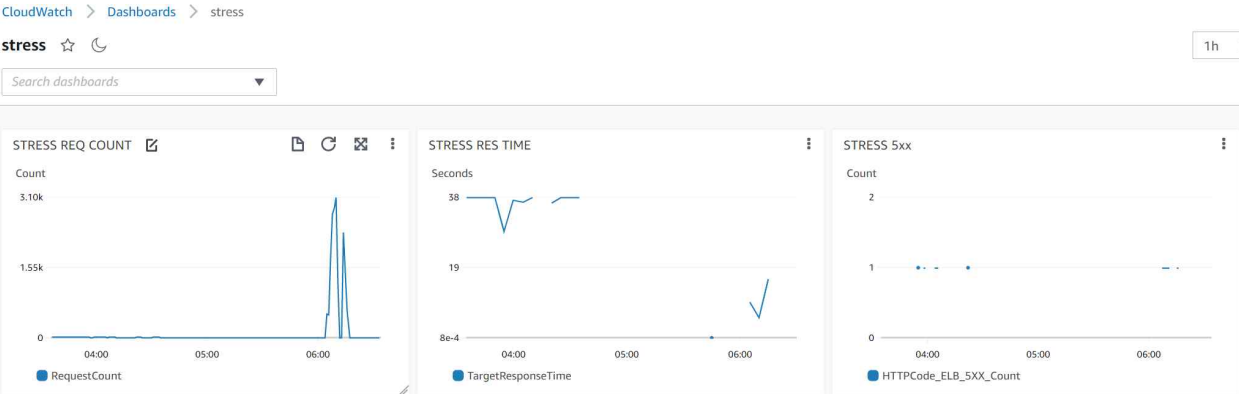
| 순번 | 채점 항목 |
|-----|--|
| 4-3 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력하여 "c5.large"가 출력되는지 확인합니다.</p> <pre>aws eks describe-nodegroup --cluster-name skills-cluster --nodegroup-name skills-app --query "nodegroup.instanceTypes"</pre> |
| 4-4 | <p>1) SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력하여 subnet- 으로 시작하는 리소스 3개가 출력되는지 확인합니다.</p> <pre>aws eks describe-nodegroup --cluster-name skills-cluster --nodegroup-name skills-addon --query "nodegroup.subnets"</pre> |
| 5-1 | <p>1) SSH를 통해 Bastion 서버에 접근한 후 아래 명령어로 디렉토리를 이동합니다.</p> <p>2) deployment.yaml 파일을 열어 아래처럼 skills/version: v1이 되어 있는 부분을 찾습니다.</p>  <pre>kind: Deployment metadata: name: match namespace: skills labels: app: match spec: selector: matchLabels: app: match template: metadata: labels: app: match skills/version: v1</pre> <p>3) 해당 부분을 skills/version: v101로 변경 후 저장합니다.</p> <p>4) 변경된 yaml 파일을 적용합니다.</p> <pre>kubectl apply -n skills -f deployment.yaml</pre> <p>5) 아래 명령어를 입력하여 새로 생성된 ReplicaSet ID를 기록합니다.</p> <pre>kubectl describe -n skills deployment match grep NewReplicaSet grep match</pre> <p>6) 생성된 ReplicaSet(위에서 검색한 ID)을 바탕으로 생성된 Pod를 검색합니다.</p> <pre>kubectl get pods -n skills grep match-xxxxyyyyzz</pre> <p>7) Pod의 상태가 아래와 같이 Running인지 확인합니다.</p> <pre>kubectl get pods grep match-xxxxyyyyzz</pre> <pre>match-xxxxyyyyzz-pppzz 1/1 Running 0 3m8s</pre> <p>8) 아래 명령어로 Pod를 상세보기 하여 skills/version: v101이 출력되는지 확인합니다.</p> <pre>kubectl get pods match-xxxxyyyyzz-pppzz -o yaml grep v101</pre> |

| 순번 | 채점 항목 |
|-----|--|
| 5-2 | <p>1) SSH를 통해 Bastion에 접근합니다.</p> <p>2) 아래 명령어로 /script로 이동 후 deploy_test.sh를 실행합니다.</p> <pre>cd ~/marking ./deploy_test.sh</pre> <p>3) 아래와 같이 기존 컨테이너가 모두 없어지고, 새로 생성된 컨테이너가 Pending, Terminating 등 상태에서 Running 상태가 될 때까지 기다립니다. 4번째 칸 85s가 생성 후 지난 시간을 의미합니다.</p> <pre>stress-xxxxxxxx-aaaaa 1/1 Running 0 85s stress-xxxxxxxx-bbbbb 1/1 Running 0 84s</pre> <p>4) 배포가 완료되면 실행 중인 스크립트를 control + c를 눌러 취소하고 생성된 stress_result_<시간>.txt 파일을 읽어 예러 개수를 확인합니다.</p> <pre>cat stress_result_053231.txt grep -v 200</pre> <p>5) 위 txt 파일에서 502, 503, 400 등 내용이 출력되면 감점입니다. 5점에서 2개 출력 당 감점 1.02이며, 출력된 내용이 하나도 없으면 5.1점입니다.</p> <p>예) 0 개 출력 시 (5.10)</p> <p>1~2 개 출력 시 (4.08)</p> <p>3~4 개 출력 시 (3.06)</p> <p>5~6 개 출력 시 (2.04)</p> <p>7~8 개 출력 시 (1.02)</p> <p>9~ 개 출력 시 (0)</p> |

| 순번 | 채점 항목 |
|-----|--|
| 6-1 | <p>1) SSH를 통해 Bastion에 접근합니다.</p> <p>2) 아래 명령어를 통하여 match 컨테이너 ID 하나를 기록합니다.</p> <pre>kubectl -n skills get pods grep match</pre> <p>3) 해당 컨테이너에 접근합니다.</p> <pre>kubectl -n skills exec -it match-xxxxxxx-aaaaa sh</pre> <p>5) 아래 명령어를 입력해 302 Found 라는 문구가 포함된 응답을 출력하는지 확인합니다.</p> <pre>curl http://www.naver.com</pre> |
| 6-2 | <p>1) 6-1 컨테이너 접근을 유지합니다.</p> <p>2) 아래 명령어로 IPv4 아이피 하나가 출력되는지 확인합니다.</p> <pre>echo \$STRESS_SERVICE_HOST</pre> <p>3) 아래 명령어를 입력하여 출력되는 응답이 없거나 timeout 에러 등으로 접근이 불가능한지 확인합니다. (10초 이내 응답이 없으면 성공으로 간주합니다.)</p> <pre>curl http://\${STRESS_SERVICE_HOST}:\${STRESS_SERVICE_PORT}/v1/stress</pre> <p>or</p> <pre>curl --connect-timeout 10 http://stress.skills.svc.cluster.local/v1/stress</pre> |
| 6-3 | <p>1) SSH를 통해 Bastion에 접근합니다.</p> <p>2) 아래 명령어를 통하여 stress 컨테이너 ID 하나를 기록합니다.</p> <pre>kubectl -n skills get pods grep stress</pre> <p>3) 해당 컨테이너에 접근합니다.</p> <pre>kubectl -n skills exec -it stress-xxxxxxx-aaaaa sh</pre> <p>4) 아래 명령어를 입력해 302 Found 라는 문구가 포함된 응답을 출력하는지 확인합니다.</p> <pre>curl http://www.naver.com</pre> |
| 6-4 | <p>1) 6-3 컨테이너 접근을 유지합니다.</p> <p>2) 아래 명령어로 IPv4 아이피 하나가 출력되는지 확인합니다.</p> <pre>echo \$MATCH_SERVICE_HOST</pre> <p>3) 아래 명령어를 입력하여 출력되는 응답이 없거나 timeout 에러 등으로 접근이 불가능한지 확인합니다. (10초 이내 응답이 없으면 성공으로 간주합니다.)</p> <pre>curl http://\${MATCH_SERVICE_HOST}:\${MATCH_SERVICE_PORT}/v1/match?token=aa</pre> <p>or</p> <pre>curl --connect-timeout 10 http://match.skills.svc.cluster.local/v1/match</pre> |

| 순번 | 채점 항목 | | | | | | | | | | | | |
|--------|--|-------|--------------------------------------|-------|---------|-------|-----|--------|--------|---|--------------------------------------|----|-------|
| 7-1 | <div>1) SSH를 통해 Bastion에 접근합니다.</div> <div>2) 아래 명령어를 입력하여 match의 ingress가 amazonaws.com이 포함된 ADDRESS가 존재하는지 확인합니다.</div> <div>kubectl -n skills get ingress grep -v stress</div> <table><tr><td>NAME</td><td>CLASS</td><td>HOSTS</td><td>ADDRESS</td><td>PORTS</td><td>AGE</td></tr><tr><td>match</td><td><none></td><td>*</td><td>xxx.ap-northeast-2.elb.amazonaws.com</td><td>80</td><td>7d13h</td></tr></table> | NAME | CLASS | HOSTS | ADDRESS | PORTS | AGE | match | <none> | * | xxx.ap-northeast-2.elb.amazonaws.com | 80 | 7d13h |
| NAME | CLASS | HOSTS | ADDRESS | PORTS | AGE | | | | | | | | |
| match | <none> | * | xxx.ap-northeast-2.elb.amazonaws.com | 80 | 7d13h | | | | | | | | |
| 7-2 | <div>1) SSH를 통해 Bastion에 접근합니다.</div> <div>2) 7-1에서 출력된 ADDRESS를 복사합니다.</div> <div>3) 아래 명령을 입력하여 403이 출력되는지 확인합니다.</div> <div>curl --silent -o /dev/null -w %{http_code} http://<ADDRESS>/health</div> <div>4) 아래 명령을 입력하여 200이 출력되는지 확인합니다.</div> <div>curl --silent -o /dev/null -w %{http_code} http://<ADDRESS>/v1/match?token=aa</div> | | | | | | | | | | | | |
| 7-3 | <div>1) SSH를 통해 Bastion에 접근합니다.</div> <div>2) 아래 명령어를 입력하여 stress의 ingress가 amazonaws.com이 포함된 ADDRESS가 존재하는지 확인합니다.</div> <div>kubectl -n skills get ingress grep -v match</div> <table><tr><td>NAME</td><td>CLASS</td><td>HOSTS</td><td>ADDRESS</td><td>PORTS</td><td>AGE</td></tr><tr><td>stress</td><td><none></td><td>*</td><td>xxx.ap-northeast-2.elb.amazonaws.com</td><td>80</td><td>7d13h</td></tr></table> | NAME | CLASS | HOSTS | ADDRESS | PORTS | AGE | stress | <none> | * | xxx.ap-northeast-2.elb.amazonaws.com | 80 | 7d13h |
| NAME | CLASS | HOSTS | ADDRESS | PORTS | AGE | | | | | | | | |
| stress | <none> | * | xxx.ap-northeast-2.elb.amazonaws.com | 80 | 7d13h | | | | | | | | |
| 7-4 | <div>1) SSH를 통해 Bastion에 접근합니다.</div> <div>2) 7-3에서 출력된 ADDRESS를 복사합니다.</div> <div>3) 아래 명령을 입력하여 403이 출력되는지 확인합니다.</div> <div>curl --silent -o /dev/null -w %{http_code} http://<ADDRESS>/health</div> <div>4) 아래 명령을 입력하여 200이 출력되는지 확인합니다.</div> <div>curl --silent -o /dev/null -w %{http_code} http://<ADDRESS>/v1/stress</div> | | | | | | | | | | | | |

| 순번 | 채점 항목 |
|-----|---|
| 8-1 | <p>1) SSH를 통해 Bastion에 접근합니다.</p> <p>2) 아래 명령어를 입력하여 출력되는 node 하나를 복사합니다.</p> <pre>kubectl get pods -n skills -o wide</pre> <p>3) 복사된 노드를 이용해 아래 명령어로 노드를 상세 출력합니다.</p> <pre>kubectl get nodes ip-10-x-x-x.ap-northeast-2.compute.internal -o json jq ".metadata.labels"</pre> <p>4) 출력된 내용 중 아래 두 라인이 포함되어 있는지 확인합니다.</p> <pre>"eks.amazonaws.com/nodegroup": "skills-app", "skills/dedicated": "app",</pre> |
| 8-2 | <p>1) SSH를 통해 Bastion에 접근합니다.</p> <p>2) 아래 명령어를 입력하여 출력되는 node를 복사합니다.</p> <pre>kubectl get pods -n kube-system -o wide grep cluster-autoscaler</pre> <p>3) 복사된 노드를 이용해 아래 명령어로 노드를 상세 출력합니다.</p> <pre>kubectl get nodes ip-10-x-x-x.ap-northeast-2.compute.internal -o json jq ".metadata.labels"</pre> <p>4) 출력된 내용 중 아래 두 라인이 포함되어 있는지 확인합니다.</p> <pre>"eks.amazonaws.com/nodegroup": "skills-addon", "skills/dedicated": "addon",</pre> |
| 9-1 | <p>1) SSH를 통해 Bastion에 접근합니다.</p> <p>2) 아래 명령어를 입력해 현재 몇 대의 nodes가 Ready 상태인지 확인합니다.</p> <pre>kubectl get nodes</pre> <p>3) 아래 명령을 통하여 부하를 주입합니다.</p> <pre>/marking/load.sh</pre> <p>4) 시간이 지남에 따라 새로운 노드가 추가되고 추가된 노드의 STATUS가 Ready로 변경되는지 확인합니다. (최대 5분까지 기다릴 수 있습니다.)</p> |

| 순번 | 채점 항목 |
|------|---|
| 10-1 | <p>1) AWS Web console에 로그인 합니다.</p> <p>2) CloudWatch page로 이동합니다.</p> <p>3) 왼쪽 패널에 Dashboards를 클릭해 이동합니다.</p> <p>4) worker dashboard를 클릭합니다.</p>  <p>5) 그림처럼 WORKER CPU, WORKER NETWORK 라는 그래프 두 개가 존재하는지 확인합니다.</p> <p>6) 9-1에서 부하를 주입하였기 때문에 CPU 사용률과 네트워크 사용률 모두 그림처럼 10분 이내에 튀는 구간이 존재하는지 확인합니다.</p> |
| 10-2 | <p>1) 10-1처럼 CloudWatch dashboard로 이동합니다.</p> <p>2) stress dashboard를 선택합니다.</p>  <p>3) 그림처럼 STRESS REQ COUNT, STRESS RES TIME, STRESS 5xx 그래프 3개가 존재하는지 확인합니다.</p> <p>4) 9-1에서 부하를 주입하였기 때문에 STRESS REQ COUNT가 그림처럼 10분 이내에 튀는 구간이 존재해야 합니다. 1000(1K) 이상인 구간 하나 이상이 존재해야 합니다.</p> |