

2021 대전광역시 제56회 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) 채점 진행 전 환경 셋업을 위해 다음 사항을 확인해야 합니다. <ul style="list-style-type: none"> - Bastion에 SSH로 접근 가능한지 확인합니다. - Bastion에서 curl, jq, awscli가 설치되었는지 확인합니다. - Bastion에서 IAM Role이 맵핑되어 awscli로 AWS 모든 리소스에 접근 가능한지 확인합니다. - aws sts get-caller-identity 명령을 통해 선수의 계정이 아닌 다른 계정에 접근하고 있는지 확인합니다. 만약, 다른 계정이라면 부정행위를 의심할 수 있습니다. 6) 문제지와 채점지에 있는 <> 는 변수입니다. 해당 부분을 변경해 입력합니다. 7) 채점은 문항 순서대로 진행해야 합니다. 8) 삭제된 내용은 되돌릴 수 없음으로 유의하여 채점을 진행합니다. 9) 이의신청까지 종료된 이후 선수가 생성한 클라우드 리소스를 삭제합니다. 10) 부분 점수가 있는 문항은 채점 항목에 부분 점수가 적혀져 있습니다. 11) 부분 점수가 따로 없는 문항은 전체 다 맞아야 점수로 인정 됩니다. 12) 채점 전 채점환경 구성을 위해 ~/.aws/config 에 아래 내용이 추가 되도록 합니다. <pre>aws configure ///// [default] region = ap-northeast-2 output = json /////</pre> 		

2. 채점기준표

1) 주요항목별 배점			직 종 명		클라우드컴퓨팅			
과제 번호	일련 번호	주요항목	배점	채점방법		채점시기		비고
				독립	합의	경기 진행중	경기 종료후	
제2과제	1	네트워킹	6.7		○		○	
	2	EC2	3		○		○	
	3	LB	4.5		○		○	
	4	Code Commit	4		○		○	
	5	Code Build	2.6		○		○	
	6	Code Deploy	4.4		○		○	
	7	Code Pipeline	10.3		○		○	
	8	ECR	4.5		○		○	
합 계			40					

2) 채점방법 및 기준

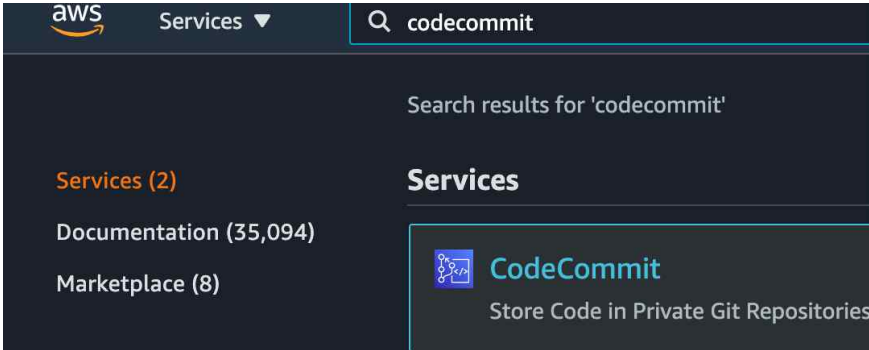
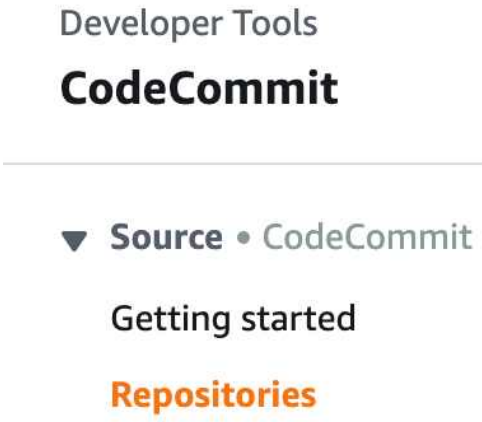
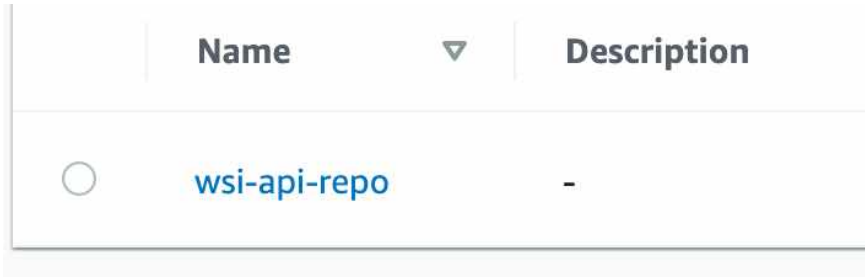
과제 번호	일련 번호	주요항목	일련 번호	세부항목(채점방법)	배점
제2과제	1	네트워킹	1	VPC	0.7
			2	서브넷	2
			3	HA 구성	2
			4	게이트웨이	2
	2	EC2	1	EC2 생성 확인	1.5
			2	IAM role 확인	1.5
	3	LB	1	LB 생성 확인	1.5
			2	healthy EC2 확인	1.5
			3	Application 호출 확인	1.5
	4	Code Commit	1	code commit 생성확인	1.2
			2	default branch 변경	1.5
			3	소스코드 업로드 확인	1.3
	5	Code Build	1	code build 생성 확인	1.1
			2	Build log 확인	1.5
	6	Code Deploy	1	code deploy 생성 확인	1.4
			2	Deploy type 확인	1.5
			3	Deploy target 확인	1.5
	7	Code Pipeline	1	Pipeline 생성확인	1.3
			2	pipeline stage 확인	1.5
			3	pipeline manual trigger	1.5
			4	new instance 배포	1.5
			5	code commit trigger	1.5
			6	new instance 동작 확인	1.5
			7	new instace LB 확인	1.5
	8	ECR	1	ECR 생성 확인	1.5
			2	Docker build 확인	1.5
			3	Docker image 확인	1.5
	총점				40

3) 채점 내용






















순번	채점 항목
1-1	1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다. 2) <code>aws ec2 describe-vpcs --filter Name=tag:Name,Values=ws1-vpc --query "Vpcs[].CidrBlock"</code> 입력 3) 10.1.0.0/16이 출력 되는지 확인
1-2	1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다. 2) 아래 명령어를 입력 합니다. <code>aws ec2 describe-subnets --filter Name=tag:Name,Values=ws1-public-a --query "Subnets[].CidrBlock"</code> 3) 10.1.2.0/24이 출력 되는지 확인 합니다. 출력 될시 1점 4) 아래 명령어를 입력 합니다. <code>aws ec2 describe-subnets --filter Name=tag:Name,Values=ws1-private-a --query "Subnets[].CidrBlock"</code> 5) 10.1.0.0/24이 출력 되는지 확인 합니다. 출력 될시 1점
1-3	1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다. 2) 아래 명령어를 입력 합니다. <code>aws ec2 describe-subnets --filter Name=tag:Name,Values=ws1-public-a --query "Subnets[].AvailabilityZone"</code> 3) "ap-northeast-2a"이 출력 되는지 확인 합니다. 출력 될시 1점 4) <code>aws ec2 describe-subnets --filter Name=tag:Name,Values=ws1-private-b --query "Subnets[].AvailabilityZone"</code> 5) "ap-northeast-2b"이 출력 되는지 확인 합니다. 출력 될시 1점

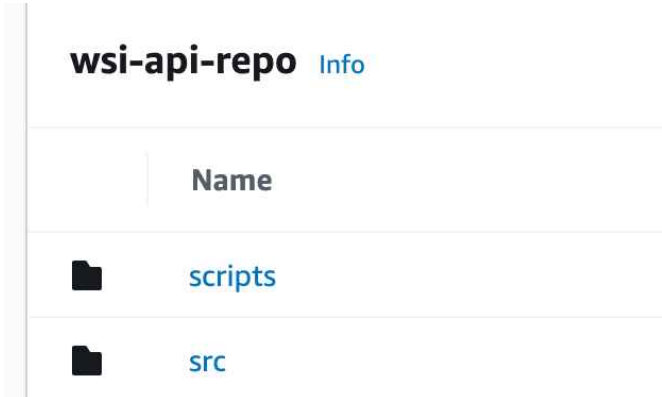
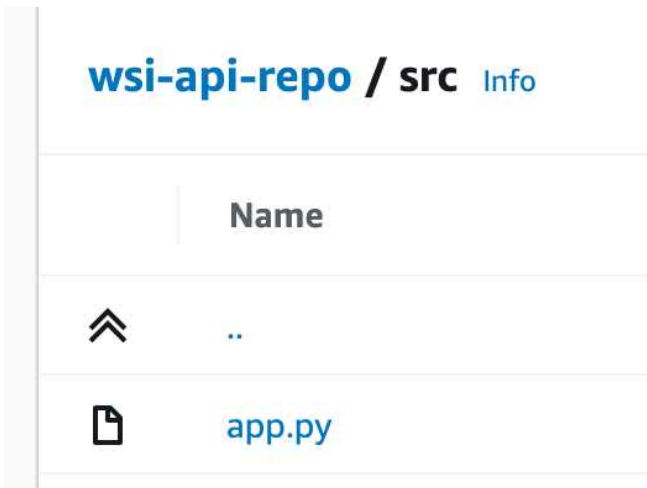
순번	채점 항목
1-4	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어 입력</p> <pre>aws ec2 describe-route-tables --filter Name=tag:Name,Values=ws-private-a-rt ₩ --query "RouteTables[].Routes[].NatGatewayId"</pre> <p>3) "nat-" 로 시작하는 문구가 출력 되는지 확인 합니다. 출력 될시 0.5점</p> <p>4) 아래 명령어 입력</p> <pre>aws ec2 describe-route-tables --filter Name=tag:Name,Values=ws-private-a-rt ₩ --query "RouteTables[].Routes[].NatGatewayId"</pre> <p>5) "nat-" 로 시작하는 문구가 출력 되는지 확인하고, 위의 채점 3)에서 출력된 것과 다른 ID를 갖는 지 확인 합니다. 0.5점</p> <p>6) 아래 명령어 입력</p> <pre>aws ec2 describe-route-tables --filter Name=tag:Name,Values=ws-public-rt ₩ --query "RouteTables[].Routes[].GatewayId"</pre> <p>7) 출력된 문구 중 "igw-" 로 시작하는 문구가 있는지 확인 합니다. 출력시 1점</p>
2-1	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력 합니다.</p> <pre>aws ec2 describe-instances --filters "Name=tag:ws:deploy:group,Values=dev-api" ₩ --query "Reservations[].Instances[].Tags[]" jq '[] select (.Key == "Name") .Value'</pre> <p>3) 아래와 동일하게 2개의 결과가 나오는지 확인 합니다.</p> <pre>"ws-api-1" "ws-api-2"</pre>
2-2	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력 합니다.</p> <pre>aws ec2 describe-instances --filters "Name=tag:Name,Values=ws-api-1" ₩ --query "Reservations[].Instances[].IamInstanceProfile[].Arn"</pre> <p>3) 아래와 같이 나오는지 확인 합니다. 중간 숫자 0000은 계정번호 임으로 다를 수 있으나 다른 문자열은 일치 해야함.</p> <pre>"arn:aws:iam::000000000000:instance-profile/ws-api"</pre>

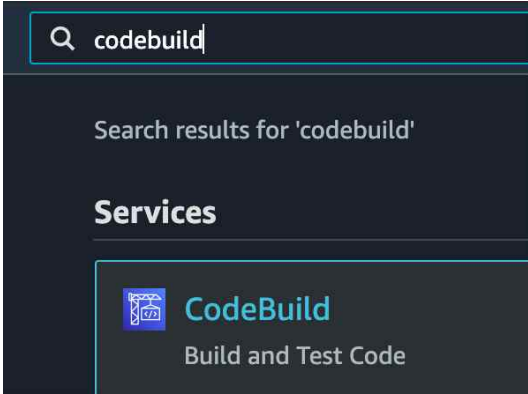
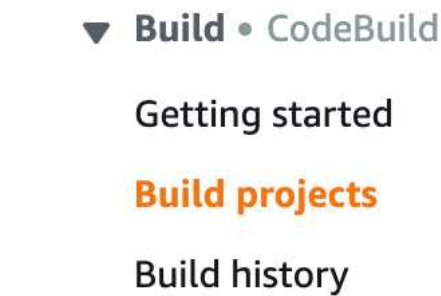

순번	채점 항목
3-1	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력 합니다.</p> <pre>aws elbv2 describe-load-balancers --names <선수가 생성한 LB Name> --query "LoadBalancers[].Type"</pre> <p>3) application 이라고 출력 되는지 확인 합니다.</p>
3-2	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력해 나오는 타겟그룹의 arn을 기록 합니다.</p> <pre>aws elbv2 describe-target-groups --names <선수가 생성한 TG Name> --query "TargetGroups[].TargetGroupArn"</pre> <p>3) 아래 명령어를 입력 합니다.</p> <pre>aws elbv2 describe-target-health --query "TargetHealthDescriptions[].TargetHealth.State" \\ --target-group-arn < 2)번에서 나온 arn ></pre> <p>4) healthy라고 표기되는 EC2 2대가 있는지 확인 합니다.</p>
3-3	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 통해 출력 되는 LB의 DNS를 기록 합니다.</p> <pre>aws elbv2 describe-load-balancers --names "wsi-api-alb" --query "LoadBalancers[].DNSName"</pre> <p>3) 아래 명령어를 입력해 {"status":"ok"} 가 나오는지 확인 합니다.</p> <pre>curl http://< 2)번에서 출력된 DNS>/health</pre>

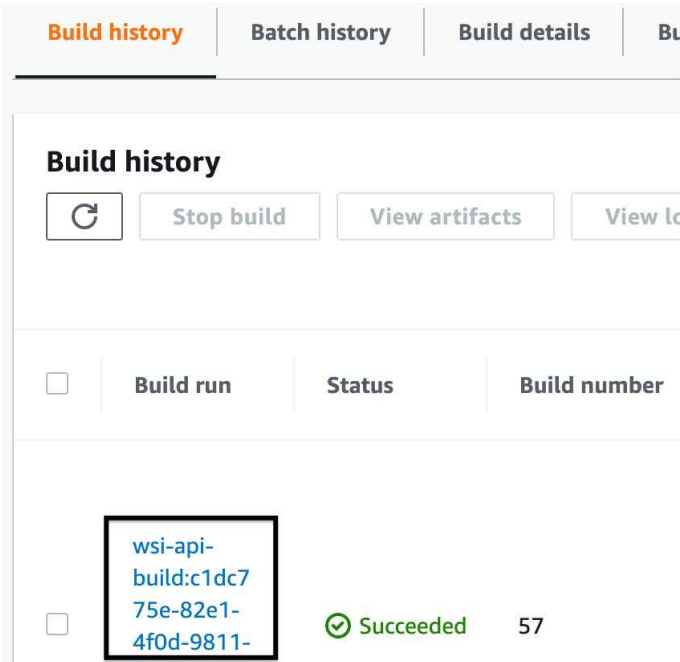
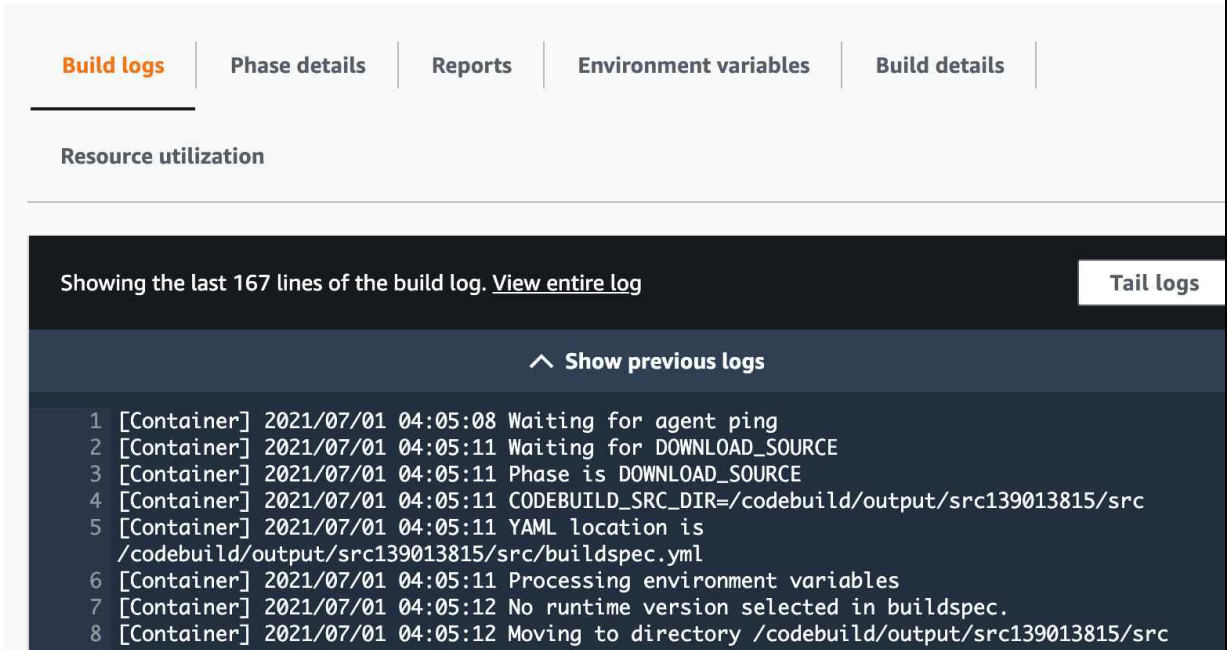
순번	채점 항목
4-1	<p>1) 웹브라우저로 AWS 페이지에 접근 합니다.</p> <p>2) 페이지 상단에서 "codecommit" 을 검색해서 해당 페이지로 이동 합니다.</p>  <p>3) 왼쪽 Source 아래의 Repositories를 클릭 합니다.</p>  <p>4) 아래 처럼 wsi-api-repo가 생성 되어 있는지 확인 합니다.</p> 

순번	채점 항목
4-2	<p>1) 4-1 채점항목을 참고해 생성된 wsi-api-repo로 이동 합니다.</p> <p>2) wsi-api-repo 페이지로 이동 뒤 왼쪽 Branches를 클릭 합니다.</p> <div data-bbox="229 383 692 987"> <p>Developer Tools</p> <p>CodeCommit</p> <hr/> <p>▼ Source • CodeCommit</p> <p>Getting started</p> <p>Repositories</p> <p>Code</p> <p>Pull requests</p> <p>Commits</p> <p>Branches</p> </div> <p>3) wsi-api-repo의 branch 페이지에서 main이 "Default branch"로 되어 있는지 확인 합니다.</p> <div data-bbox="213 1106 766 1883"> <p>wsi-api-repo</p> <div> <p>Branches Info</p> <div> <p>Delete branch</p> <p>View bran</p> </div> <p>Create branch</p> <div> <p>Q</p> </div> <div> <p>Branch name</p> </div> <div> <p><input type="radio"/> main Default branch</p> <p><input type="radio"/> release</p> </div> </div> </div>

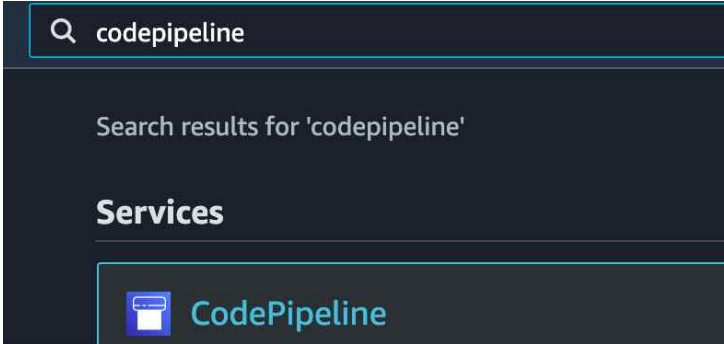
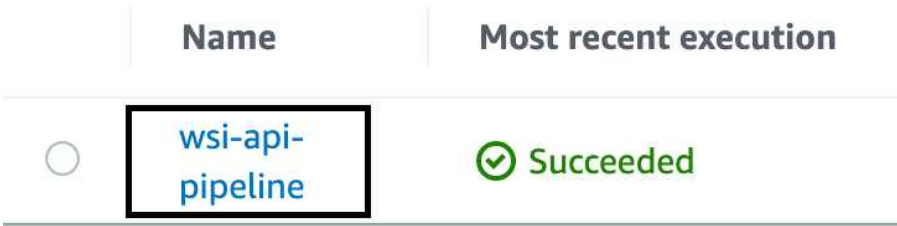
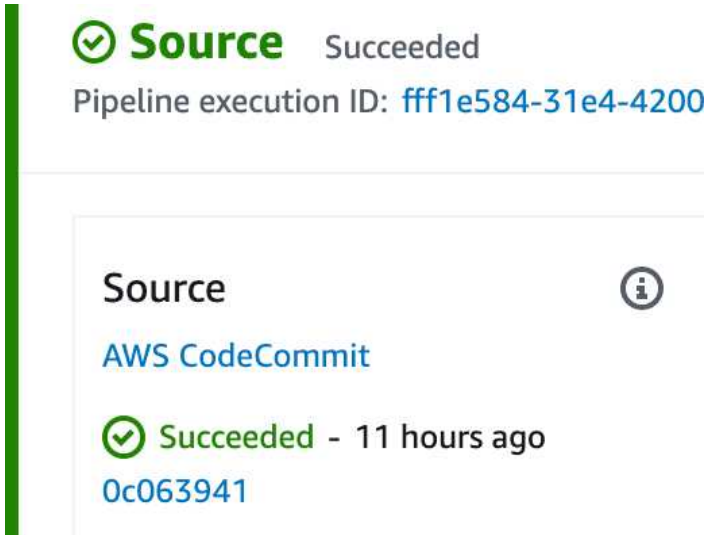
순번	채점 항목																
4-3	<p>1) 4-1 채점항목을 참고해 생성된 wsi-api-repo로 이동합니다.</p> <p>2) 왼편의 Code로 이동 합니다.</p> <div> <p>Developer Tools</p> <p>CodeCommit</p> <hr/> <p>▼ Source • CodeCommit</p> <p>Getting started</p> <p>Repositories</p> <p>Code</p> </div> <p>3) 네모 박스 처럼 appspec.yml 파일이 있는지 확인합니다.</p> <div> <p>wsi-api-repo Info</p> <hr/> <table> <thead> <tr> <th></th><th>Name</th></tr> </thead> <tbody> <tr> <td></td><td>scripts</td></tr> <tr> <td></td><td>src</td></tr> <tr> <td></td><td>appspec.yml</td></tr> <tr> <td></td><td>buildspec-rel.yaml</td></tr> <tr> <td></td><td>buildspec-rel.yaml</td></tr> <tr> <td></td><td>buildspec.yaml</td></tr> <tr> <td></td><td>index.html</td></tr> </tbody> </table> </div> <p>뒷장에 4-3 채점이 계속 됩니다.</p>		Name		scripts		src		appspec.yml		buildspec-rel.yaml		buildspec-rel.yaml		buildspec.yaml		index.html
	Name																
	scripts																
	src																
	appspec.yml																
	buildspec-rel.yaml																
	buildspec-rel.yaml																
	buildspec.yaml																
	index.html																


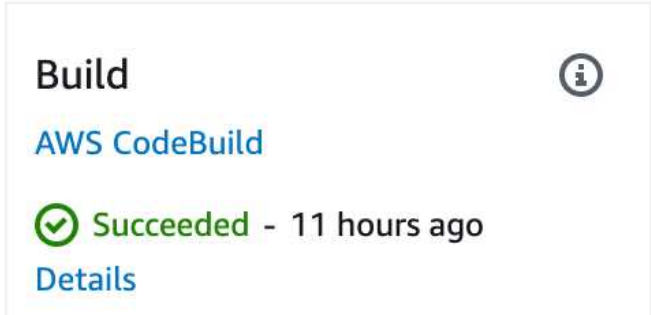

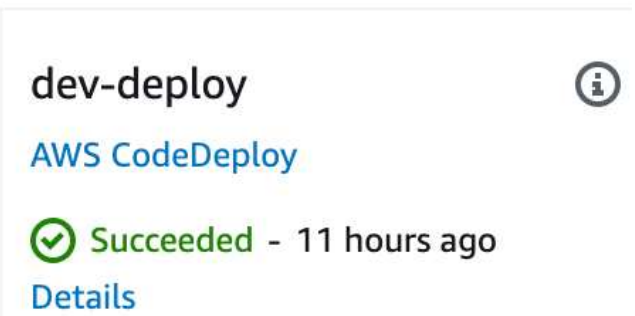
순번	채점 항목
	<p>4) src 디렉토리로 이동합니다.</p> 
4-3	<p>5) src 디렉토리 아래에 app.py 파일이 있는지 확인합니다.</p> 

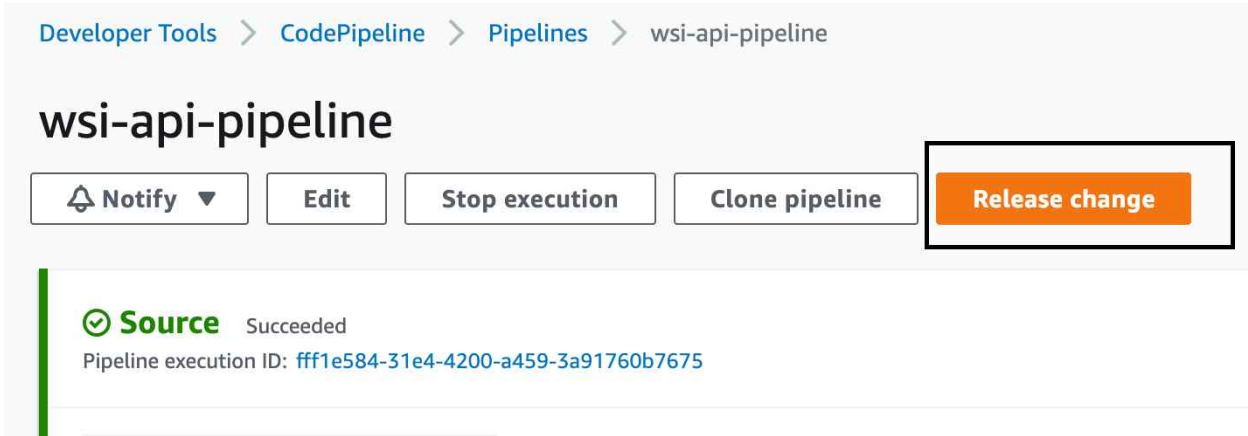
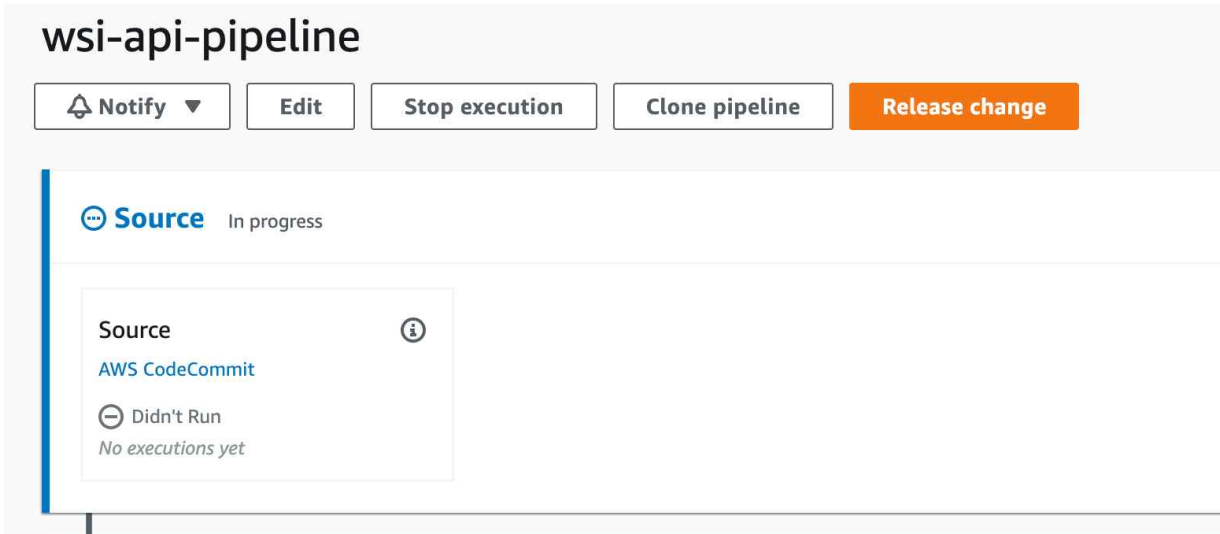
순번	채점 항목
5-1	<p>1) 웹브라우저로 AWS 페이지를 접속 합니다.</p> <p>2) 페이지 상단에서 "codebuild"를 검색해 해당 페이지로 이동 합니다.</p>  <p>3) 왼쪽의 Build projects를 클릭 합니다.</p>  <p>4) 아래 처러 wsi-api-build가 생성 되어 있는지 확인 합니다.</p> 

순번	채점 항목
5-2	<p>1) 5-1을 참고하여 wsi-api-build code build 페이지로 이동 합니다.</p> <p>2) Build history에서 가장 최근에 실행된 build 하나를 클릭 합니다.</p>  <p>2) build 페이지 중간에 Build log를 클릭 하고 아래 그림처럼 로그가 출력 되는지 확인 합니다. 로그 내용은 선수마다 다를 수 있으므로 로그가 활성화 되어 볼 수 있는지만 확인 하면 됩니다.</p> 
6-1	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력해 "wsi-api"가 출력 되는지 확인 합니다.</p> <pre>aws deploy get-application --application-name wsi-api --query "application.applicationName"</pre>



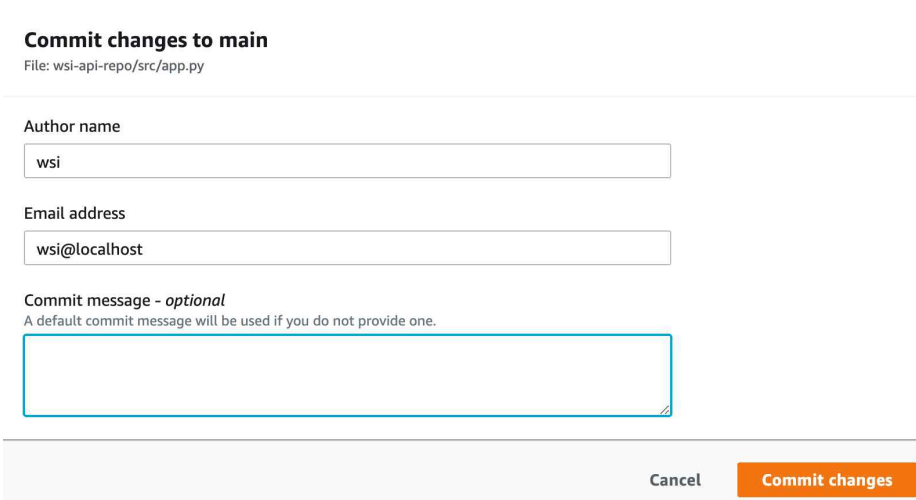
순번	채점 항목
6-2	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력해 "IN_PLACE" 가 출력 되는지 확인 합니다.</p> <pre>aws deploy get-deployment-group --application-name wsi-api --deployment-group-name ₩ dev-api --query "deploymentGroupInfo.deploymentStyle.deploymentType"</pre>
6-3	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력 합니다.</p> <pre>aws deploy get-deployment-group --application-name wsi-api ₩ --deployment-group-name dev-api --query "deploymentGroupInfo.ec2TagSet.ec2TagSetList[]"</pre> <p>3) 출력 결과에 아래 두개가 포함 되어 있는지 확인 합니다.</p> <pre>"Key": "wsi:deploy:group", "Value": "dev-api",</pre>
7-1	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) 아래 명령어를 입력해 출력된 결과 중 "wsi-api-pipeline" 이 있는지 확인 합니다.</p> <pre>aws codepipeline list-pipelines --query "pipelines[].name"</pre>

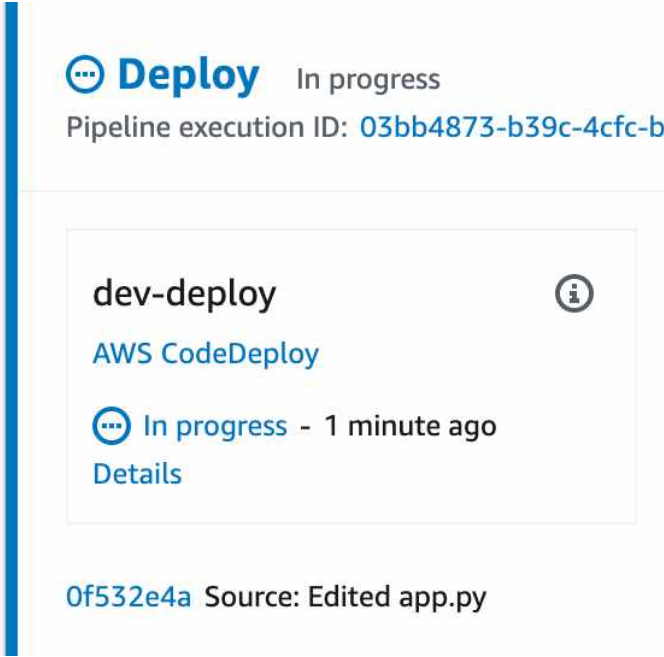
순번	채점 항목
7-2	<p>1) 웹브라우저로 AWS 페이지에 접근 합니다.</p> <p>2) 페이지 상단에 codepipeline을 검색 하여 해당 페이지로 이동 합니다.</p>  <p>3) wsi-api-pipeline 파이프라인을 클릭 합니다.</p>  <p>4) 파이프라인 처음 시작이 Source 인지 확인 합니다.</p> <p>해당 Source는 아래 그림 처럼 AWS CodeCommit을 Source로 가져야 합니다.</p>  <p>뒷장에 7-2 채점이 계속 됩니다.</p>


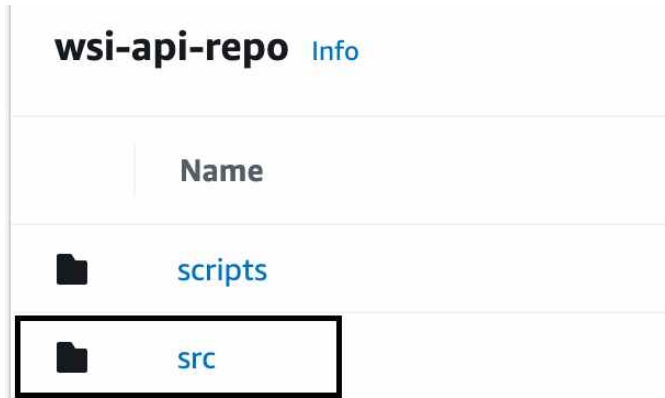
순번	채점 항목
7-2	<p>5) Source 다음 단계로 Build를 갖는지 확인 합니다.</p> <p>Build는 아래그림처럼 AWS CodeBuild를 build로 가져야 합니다.</p> <div data-bbox="268 398 944 510">  </div> <div data-bbox="268 600 922 913">  </div> <p>6) Build 다음 단계로 Deploy를 갖는지 확인 합니다.</p> <p>Deploy는 아래 그림처럼 AWS CodeDeploy를 Deploy로 가져야 합니다.</p> <div data-bbox="268 1104 906 1216">  </div> <div data-bbox="268 1305 906 1619">  </div>

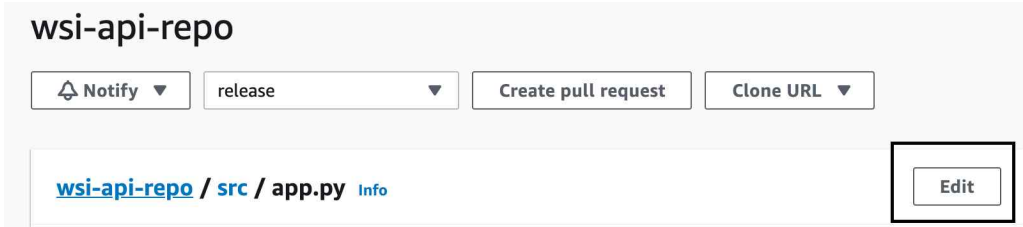

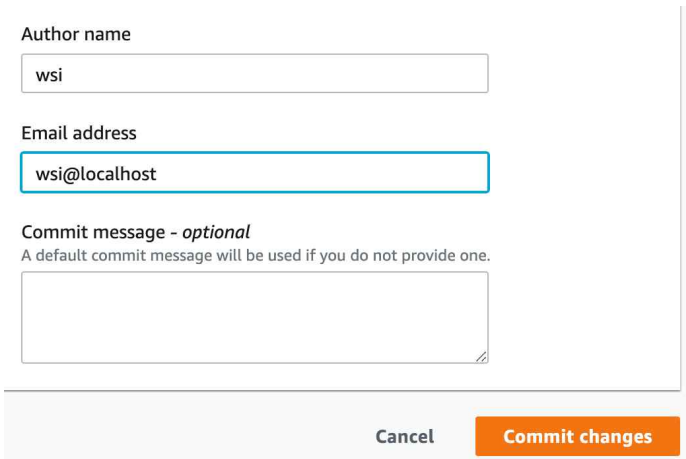
순번	채점 항목
7-3	<p>1) 7-2 채점 항목을 참고해 wsi-api-pipeline으로 이동 합니다.</p> <p>2) wsi-api-pipeline으로 이동뒤 release change를 클릭 합니다.</p> <div data-bbox="212 383 1466 817">  <p>Developer Tools > CodePipeline > Pipelines > wsi-api-pipeline</p> <h2>wsi-api-pipeline</h2> <p>Notify Edit Stop execution Clone pipeline Release change</p> <p>Source Succeeded Pipeline execution ID: fff1e584-31e4-4200-a459-3a91760b7675</p> </div> <p>3) 아래 처럼 Source나 Build가 파란색에 In progress로 보인다면 release가 실행된것 입니다.</p> <div data-bbox="248 931 1474 1464">  <h2>wsi-api-pipeline</h2> <p>Notify Edit Stop execution Clone pipeline Release change</p> <p>Source In progress</p> <div> <p>Source</p> <p>AWS CodeCommit</p> <p>Didn't Run</p> <p>No executions yet</p> </div> </div> <p>4) Source > Build > Deploy 가 순차 실행되며 마지막 단계에선 Deploy가 파란색 In progress로 보 입니다. 파란색으로 보인다면 진행 중 임으로 기다리고, 빨간색이면 에러 임으로 틀린 것 입니다. 최종적으로 모두 정상 완료되어 Source, Build, Deploy가 성공으로 끝나 모두 초록색으로 보이는지 확인 합니다. 배포는 최대 15분 정도 소요될 수 있습니다.</p>

순번	채점 항목
7-4	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근 합니다.</p> <p>2) /opt/ec2_launch.sh wsi-api-9847 명령어를 입력 합니다.</p> <p>3) 아래 명령어를 입력 하여 wsi-api-9847 EC2가 생성 되었는지 확인 합니다.</p> <pre>aws ec2 describe-instances --filters "Name=tag:Name,Values=wsi-api-9847" \ --query "Reservations[].Instances[].Tags[]"</pre> <p>4) 출력된 결과에 아래와 같은 문구가 포함 되어 있는지 확인 합니다. 생성 되는데 일정 시간이 소요 됨으로 생성 명령어 수행 이후 최대 2분 정도까지 3)번 확인 명령어를 재시도 해볼 수 있습니다.</p> <pre>{ "Key": "wsi:deploy:group", "Value": "dev-api" }</pre>

순번	채점 항목
7-5	<p>1) 4-1 채점항목을 참고해 생성된 wsi-api-repo로 이동합니다.</p> <p>2) src/app.py 파일을 클릭합니다.</p> <p>3) 파일을 클릭 한뒤 아래와 같은 페이지가 보일시 Edit 버튼을 눌러 편집합니다.</p> <div data-bbox="213 439 1279 645">  </div> <p>4) 네모 박스와 같이 ret 값을 변경합니다.</p> <p>ret = {'hash': 'wsi-ffad-9642', 'code': 200} 으로 변경합니다.</p> <div data-bbox="213 788 1279 1375">  </div> <p>5) 스크롤을 내려 아래와 같이 수정자 정보를 입력하고 commit changes 를 클릭합니다.</p> <div data-bbox="213 1491 1133 1989">  </div> <p>뒷장에 7-5 채점이 계속 됩니다.</p>

순번	채점 항목
7-5	<p>6) 7-2 채점항목을 참고하여 wsi-api-pipeline codepipeline 페이지로 이동합니다.</p> <p>7) Deploy가 파란색으로 되어 진행중인지 확인합니다. Deploy가 성공해 초록색 Success로 변경될 때까지 기다립니다. 최대 15분정도 소요 될 수 있습니다.</p> <div data-bbox="220 456 887 1111">  </div> <p>8) 완료되면 Bastion에 SSH로 접근합니다.</p> <p>9) 아래 명령어를 통해 출력 되는 LB의 DNS를 기록합니다.</p> <pre>aws elbv2 describe-load-balancers --names "wsi-api-alb" --query "LoadBalancers[].DNSName"</pre> <p>10) 아래 명령어를 입력해 {'hash': 'wsi-ffad-9642', 'code': 200} 가 나오는지 확인합니다.</p> <pre>curl http://< 2)번에서 출력된 DNS>/health</pre>
7-6	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력해 새로 생성된 인스턴스의 IP주소를 얻습니다.</p> <pre>aws ec2 describe-instances --filters "Name=tag:Name,Values=wsi-api-9847" \ --query "Reservations[].Instances[].NetworkInterfaces[].PrivateIpAddress"</pre> <p>3) ssh ec2-user@< 2)번 출력된 IP>를 통해 새로운 인스턴스로 접근합니다. 필요에 따라 -i 옵션으로 키를 통해 접근 가능합니다.</p> <p>4) 아래 명령어를 입력해 {"code":200,"hash":"wsi-ffad-9642"} 라고 출력 되는지 확인합니다.</p> <pre>curl http://localhost:80/health</pre>

순번	채점 항목
7-7	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력해 새로 생성된 인스턴스 ID를 얻습니다.</p> <pre>aws ec2 describe-instances --filters "Name=tag:Name,Values=ws-api-9847" \ --query "Reservations[].Instances[].InstanceId"</pre> <p>2) 아래 명령어를 입력해 나오는 타겟그룹의 arn을 기록합니다.</p> <pre>aws elbv2 describe-target-groups --names ws-api-tg --query "TargetGroups[].TargetGroupArn"</pre> <p>3) 아래 명령어를 입력합니다.</p> <pre>aws elbv2 describe-target-health --target-group-arn < 3)번에서 나온 arn > \ --query "TargetHealthDescriptions[].TargetId"</pre> <p>4) 출력된 결과 중에 2)의 결과인 새로운 인스턴스 ID가 포함 되어 있는지 확인합니다.</p>
8-1	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령어를 입력해 ws-api-ecr을 출력 하는지 확인합니다.</p> <pre>aws ecr describe-repositories --repository-name ws-api-ecr --query "repositories[].repositoryName"</pre>
8-2	<p>1) 웹브라우저로 AWS 페이지에 접근합니다.</p> <p>2) 4-1 채점항목을 참고하여 ws-api-repo code commit 페이지로 이동합니다.</p> <p>3) 아래 그림처럼 release branch를 선택합니다.</p>  <p>4) src 디렉토리로 이동 합니다.</p>  <p>뒷장에서 8-2 채점이 계속 됩니다.</p>

순번	채점 항목
8-2	<p>5) Edit 버튼을 눌러 app.py 파일 편집모드로 들어갑니다.</p> 
	<p>6) app.py 파일 내용중 네모 박스 부분을 그림과 같이 변경합니다.</p> <pre>ret = {'hash': 'wsi-release-ffaa', 'code': 200}</pre>  <p>7)스크롤을 내려 author와 email을 임의 값으로 채운 뒤 commit 을 누릅니다.</p>  <p>뒷장에 8-2 채점이 계속 됩니다.</p>

순번	채점 항목
8-2	<p>8) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근합니다.</p> <p>9) <code>aws ecr list-images --repository-name wsi-api-ecr --query "imagelds[].imageTag"</code> 명령어를 입력합니다.</p> <p>10) 출력되는 결과 중 현재 시간으로 된 이미지가 있는지 확인합니다. 년도 월 일 시 분 초 형식 입니다. (만약, 21년 07월 01일 14시 12분 9초라면 210701141209로 표기됨)</p> <p>빌드와 업로드에 최대 5분정도 소요 될 수 있습니다. 5분간 9)번 명령어를 여러 번 입력해도 무방합니다. 현재 날짜의 이미지를 만들어야 함으로 현재 시간과 5분 이상 차이나면 안 됩니다.</p>
8-3	<p>1) putty등의 프로그램으로 SSH를 통해 Bastion 서버에 접근합니다.</p> <p>2) 아래 명령으로 ECR repo의 URI를 기록합니다.</p> <pre>aws ecr describe-repositories --repository-name wsi-api-ecr --query "repositories[].repositoryUri"</pre> <p>3) 아래 명령으로 가장 최신 image tag를 기록합니다.</p> <pre>aws ecr list-images --repository-name wsi-api-ecr --query "imagelds[].imageTag"</pre> <p>4) 아래 명령어로 ECR에 로그인 합니다.</p> <pre>aws ecr get-login --no-include-email sh -x</pre> <p>5) 아래 명령으로 도커이미지를 다운로드 합니다.</p> <pre>docker pull <2번의 URI>:<3번의 image tag></pre> <p>예) <code>docker pull 0000000000.dkr.ecr.ap-northeast-2.amazonaws.com/wsi-api-ecr:210701141209</code></p> <p>6) 아래 명령으로 도커를 실행합니다.</p> <pre>docker run -d -p 28888:80/tcp < 2)번의 URI>:< 3)번의 image tag></pre> <p>7) 아래 명령을 호출해 <code>{"code":200,"hash":"wsi-release-ffaa"}</code> 라고 응답이 오는지 확인합니다.</p> <pre>curl http://localhost:28888/health</pre>