MODUL

LKS PROVINSI JAWA TIMUR

CLOUD COMPUTING

2023

Description of project and tasks

This module is eight hours - **Day 2 – Project 2: Automate Deployment**.
The goal of this project is to automate the deployment process of two service applications to the production cluster servers to support the backend of the Course App application. Today you will create a pipeline to automate the deployment process of the backend services to more reliable and efficient by the terms is given. You should also build the best production Architecture to be more reliable, scalable, secure, and cost-efficient.

Task

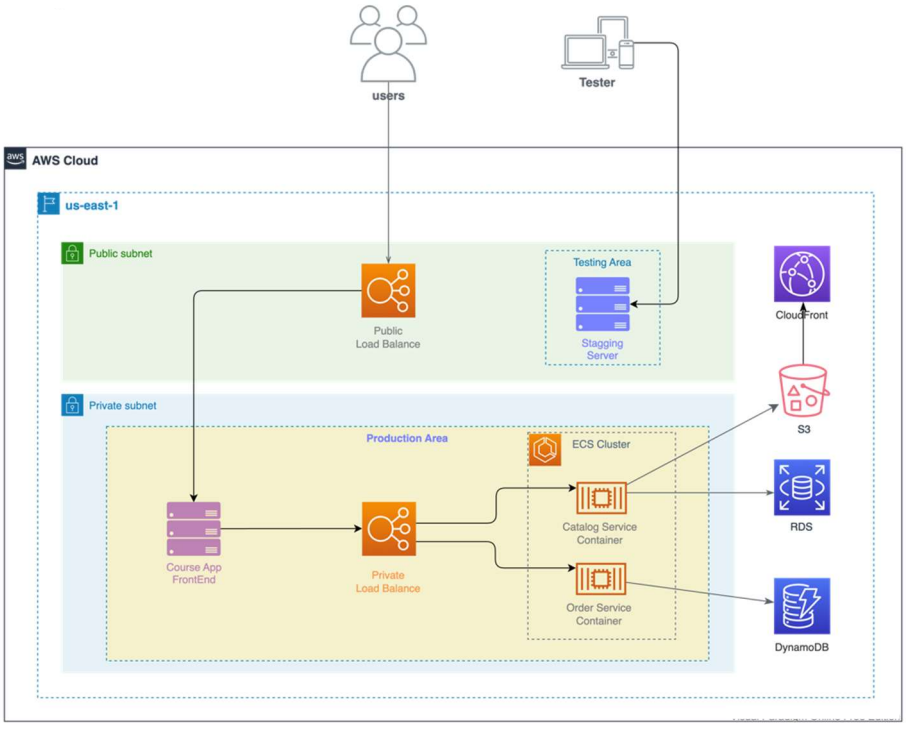
1. Share your AWS pro account on this link ***https://s.id/akunccjatim2023***.
2. Read the documentation thoroughly (Outlined below).
3. Please read and understand the application architecture in the ***Architecture section***.
4. Please carefully read the ***technical details*** section.
5. Please carefully read the ***application details***.
6. Prepare your GitHub account.
7. Log into the AWS console.
8. Set up your VPC configurations. The VPC configuration details are in the ***Network Architecture - Service Details*** section.
9. Set up a security group. You can read security additional rules in ***Security – Service Details*** section.
10. Set up initial support services like Elastic Container Service, database, object storage (S3), and CloudFront. More details about these services are in the ***Service Details*** section.
11. Prepare the IAM and SSM Parameter Store to provide access and configuration for the application.
12. Set up container orchestration with Elastic Container Service. The ECS configuration details are in the ***Elastic Container Service - Service Details*** section.
13. Create the pipeline according to the instruction given in ***Deployment Process Details*** Section and deploy the backend services.
14. Prepare and install Frontend. More details instructions are in The ***Frontend - Application*** section.
15. Configure necessary application monitoring and metrics in CloudWatch.

Technical Details

1. The goal of this project is to build automation deployment for the backend service, so please focus on that or you will lose many points.
2. The Course App has 3 source codes that you must fork from the GitHub source to your GitHub account. Please read the ***Application Section Details***.
3. The Course App has two services with one additional service that must be deployed to different server setups.
4. All instances that are used in this project must use T3.small as the instance type.
5. The staging server must use AWS Beanstalk with Docker as a platform.

6. Both backend services like *catalog service* and *order service* must be deployed automatically and triggered by GitHub changes.
7. The backend services must be deployed in ECS Cluster, please read ***ECS Service Details***.
8. The Front-End and the back-end services must be in a private subnet, the more secure the architecture more points you will earn.
9. You should be creating an additional security group following in the security Service Details section.
10. Every service that you have created must have a name starting with the word lks. For example, lks-allow-http, lks-course-pipeline, lks-rules, etc.
11. Remember to label every AWS service you have created like VPC, security group, ec2 instance, and everything else you have created except those services that were created automatically. The more attention to the little things, the more point you will earn.
12. Remember to fill in each description and tag of the services.
13. Remove the service items that are not needed in this project, such as vpc, security group, IAM user which are not needed in this project. For example, you create a security group to allow port 80 and you create a 2nd security group with port 8080, the project only requires port 80, then you must delete the security group with port 8080. Make your work clean and easy to understand or you will get a point deduction.
14. The base OS that has been chosen is **Amazon Linux** (https://aws.amazon.com/amazon-linux-ami/). This distribution was selected for its broad industry support, stability, availability of support, and excellent integration with AWS.

Architecture



The above example illustrates one possible architectural design for the Course Application. This shows all required segments needed for server requests. Please read the Application detail.

Application Details

The Course Apps was developed with JavaScript as the main language with Node JS as a runtime for the backend and NuxtJS as the framework that is used in the frontend. The Course App has 2 layers of the application that call Frontend and Backend. The Frontend is used to provide views and interaction between applications and users, this layer will be shown on your computer while you access it while the backend will work behind the scenes. The backend layer is for handling any functionality that Frontend needed via API (Application Programming Interface) and will communicate with any AWS services. Course Apps has 2 services are running as a backend namely **catalog service** and **order service**, both of those services have their respective function to support frontend needs. So, the Frontend needs to communicate with both of two backend services via API with a different path to access it. The frontend will use path ***/api/v1/course*** to access the **catalog API service** and ***/api/v1/order*** to access the **order API service**. Both of those services are continuously developed to follow the feature requirement of the user, so both of those services must be deployed continuously following the version of the services, so please read ***Deployment Life Cycle*** for more detail about it.

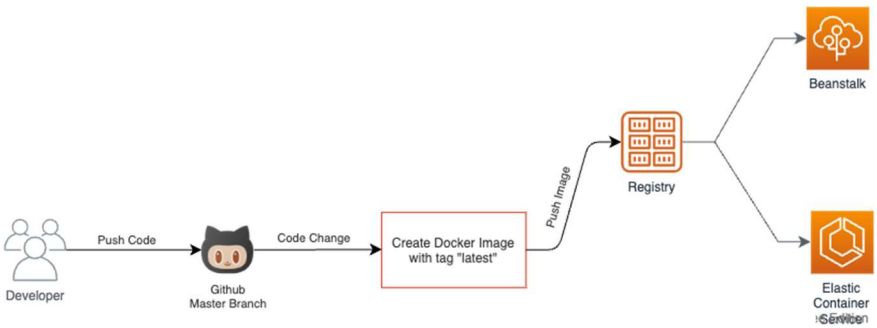
Frontend

You can deploy the frontend service in EC2 with the best suite architecture for handling any request. This frontend must be accessed from a public Load Balancer with the name lks-lb-frontend and target group name lks-tg-frontend. Please keep the frontend in a private zone for security reason. You can follow the installation instruction in the README.md of this frontend repository. The repository is *https://github.com/betuah/lks-course-frontend*.

Backend

All backend deployment must be run automatically with Code Pipeline. Please see the deployment Life Cycle for more information about it. You can test the backend service with put temporary in the public load-balance and accessing it via Postman. You can see API Endpoint in README.md of each service repository.

Deployment Life Cycle



Catalog and order services need to deploy continuously following the version of those services. All services should be automatically deployed to ECS, you can use code pipeline to automate the deployment process. The pipeline will run automatically triggered by a code change in the master

branch in the repository catalog or order service. The public repository used is GitHub, so you must have a GitHub account and fork the repository catalog and order service to your account. You can see the repository address of each service in the **Pre-requirements section**. Every service, be it catalog or order, has a Dockerfile in its repository. You can create a docker image using the dockerfile provided, read the descriptions in each repository for how to install and create an image. Each image will be uploaded to the private registry with the latest tag and version with code build number, for example:

ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/catalog:**latest**
ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws. com/catalog:**v1**

Note: *You can use existing environment ${**CODEBUILD_BUILD_NUMBER**} in codebuild as a version of the image.*

You can use the Elastic container registry as a private docker registry. The images you have saved will be deployed to AWS beanstalk and the Elastic Container Service. All this process must be run in Code Pipeline.

Pre-Requirement

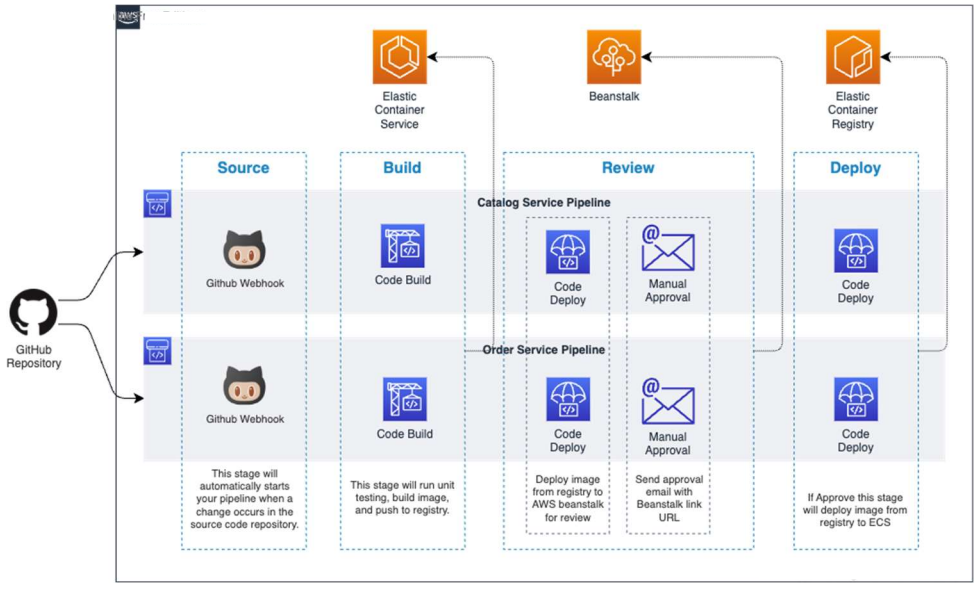
Before you start creating the pipeline you have to fork the repository of each service into your own GitHub repository, here is the repository of the catalog and order services:

Catalog Service: https://github.com/betuah/lks-course-catalog
Order Service: https://github.com/betuah/lks-course-order

You need to create an IAM user with the names lks-catalog-user and lks-order-user to create access and a secret key for each service. Please read the application installation instruction in README.md of each repository. You need to prepare a registry repository with repository names lks-catalog-image and lks-order-image then put it in the registry private repositories.

Note*: Please read the **README.md** of each service repository for the detail's installation and application requirements.*

Code Pipeline



Create pipeline names lks-catalog-pipeline and lks-order-pipeline. You must create 4 stages in the code pipeline with the following stages:

- **Source**: This stage will trigger the pipeline from a change that occurs in the catalog or order service source code repository that you have forked before. This stage will take the service (Catalog/Order) source code from your GitHub repository to be a source artifact.
- **Build**: In This stage, you must create the buildspec.yml that will run the unit test, build an image from Dockerfile that is already in the source code and push the image to ECR (Elastic Container Registry). You can see the buildspec.yml template in the Buildspec Template section.
- **Review**: This stage will deploy the image from ECR to AWS Beanstalk to review and send an approval confirmation email to your email and lkscloud@technobrainlab.com, don't forget to attach the link of AWS Beanstalk to that approval confirmation email. You can see the AWS Beanstalk detail service section for more detail about deploying AWS Beanstalk from ECR.
- **Deploy**: When the reviewer has approved the email and this stage will automatically be triggered to deploy the created image from ECR (Elastic Container Registry) with the latest tag to ECS (Elastic Container Service). You can see the ECS Details service section for more detail about Elastic Container Service.

Attention: *You can skip the review stage if you give up on it and lose some points but make sure you can deploy to ECS well or you* ***will lose a lot of points****. You will get extra points when all stages including the review stage can be run successfully*.

Note*: If you create the IAM roles for codebuild or code pipeline, please use lks-pipeline-catalog-role and lks-pipeline-order-role for the name of codepipeline roles. Use lks-build-catalog-role and lks-build-order-role for the codebuild roles.*

Docs: *https://docs.aws.amazon.com/codepipeline/latest/userguide/pipelines.html*

Buildspec Template

This is a buildspec template that you can use and adjust according to the pipeline you have created:

```
version: 0.2

env: # Create the environment
  variables:
    ECR_REPO: YOUR_ECR_REPO
    ECR_REGION: YOUR_ECR_REGION
    CONTAINER_NAME: YOUR_CONTAINER_NAME

phases:
  install: # Install runtime
    runtime-versions:
      nodejs: 16
      docker: 20
  pre_build: # You must use this phase for installing dependencies and login into ECR
    commands: # Run your pre-build script here
      - echo "Install Dependencies and login into ECR"
  build: # In This phase you must Run Unit testing and Build an image from Dockerfile
    commands:
      - echo "Run unit testing and build docker image"
  post_build: # In this phase you must push the image to ECR and create an image definition
    commands: # Run your post-build script here
      - echo "push the image to ECR.."
      - printf '[{"name":"%s","imageUri":"%s"}]' $CONTAINER_NAME ${ECR_REPO}:latest > imagedefinitions.json
```

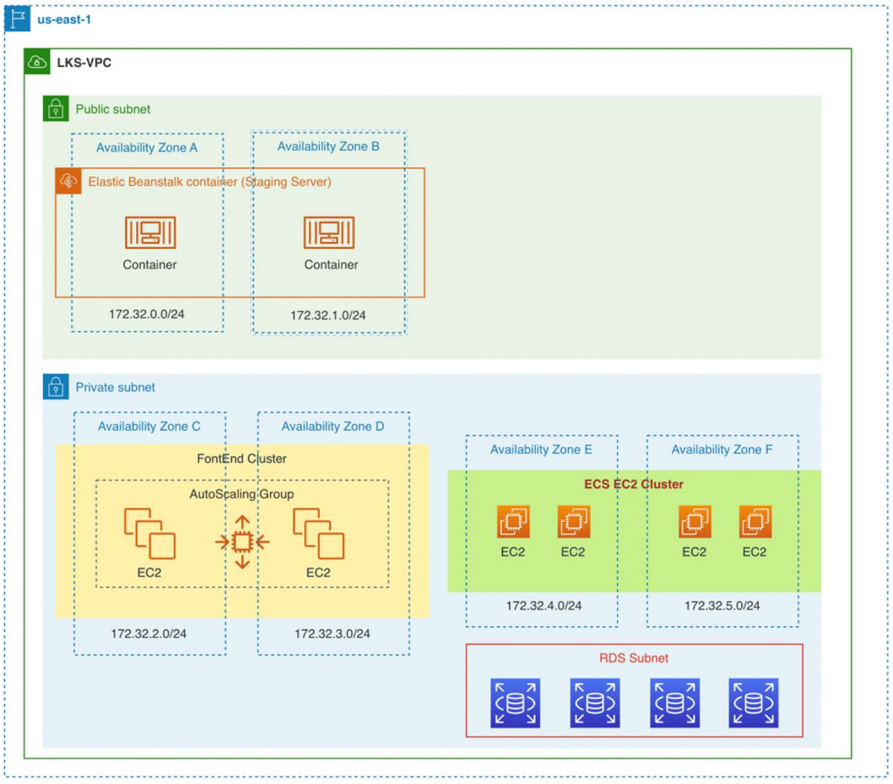
```
    - echo "build complete.."

reports: # Report file
 test-report:
  files:
   - 'report/test-result.xml'
  file-format: JUNITXML

artifacts: # Output artifact will generate build
 files:
  - imagedefinitions.json
```

Note: *This buildspec will generate an artifact with name buildSource and will be put in S3.*

Doc*: https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html*

Service Details

Networking (VPC)



You have to build a network infrastructure before you start all the tasks. You should create VPC in the us-east-1 region with names LKS-VPC and CIDR 172.32.0.0/20. As shown in the following topology, you must set up VPC with the following conditions:

- The public subnet must be containing IPv4 and IPv6 subnets with the internet-gateway name lks-inet-gw and route name lks-public-route.
- The public subnet must have 2 subnets with 2 Availability Zone with subnet names public-subnet-az-a and public-subnet-az-b.

- Private subnet must have 4 subnets with 4 Availability Zone with subet name is private-subnet-az-c, private-subnet-az-d, private-subnet-az-e, private-subnet-az-f.
- The IPv6 public subnet prefix is /64 for each AZ.

Note: *Backend service needs to communicate with other **AWS services** that run outside your VPC like S3, SSM Parameter Store, DynamoDB, and ECS, and **don't need internet access** to run it. Please give us a better solution for this condition rather than using NAT Gateway and we will give you more points.*

Security

Security is an important thing that should not be pass when you are building infrastructure. You have to include the security planning for the course app. The course app must be secure and run in a private zone with security rules. You can use a security group or IAM to protect any course app services. You can follow the security rules below to secure the course app.

- Frontend just can be accessible from ELB public zone in port 3000.
- Catalog service and order service just can be accessible from Frontend.
- SSM Parameter store just can be accessible from Catalog Service.
- PostgreSQL database in RDS just can be accessible from ECS Cluster in port 5432.
- DynamoDB just can be accessible from Order Service.
- S3 bucket can be write and read from Catalog Service.

Note: *Catalog and order services must have different AWS credentials. Every rule you have created in IAM rules, policies, or Security groups must start with the word lks. Please remove the security group or IAM user that is not used.*

SSM Parameter Store

Catalog Service is using several AWS services that exist in AWS such as S3, RDS, and CloudFront with all service configurations stored in the SSM Parameter Store. The Parameter Store Name uses a hierarchy path like **/{{APP_NAME}}/{{ENVIRONMENT}}/{{KEY}}**, you can change **{{APP_NAME}}** with **course_catalog** and **{{ENVIRONMENT}}** with **production** for production use or **testing** for testing use. The **{{KEY}}** must be strict with the key name in the table below. For example, /course-catalog/production/AWS_BUCKET_NAME.

Note: *Be careful to write the key. The wrong key and case sensitivity will have an effect and can cause the value not to be read by the application. You must create the parameter store one by one until all keys have been generated.*

AWS Beanstalk

This project uses AWS beanstalk as a staging server to review the application before it launches into the production environment. There are a few things you should pay attention to in preparing the beanstalk application, you can follow these instructions to prepare it:

1. Prepare your beanstalk application with names lks-catalog-app and lks-order-app.
2. Use beanstalk application domain with format lks-{{service_name}}-{{your-school-name}} for example, lks-catalog-smk6jember.us-east-1.elasticbeanstalk.
3. Setting up beanstalk environment using docker as a platform with an image from ECR.
4. Don't upload the image directly, use code pipeline to deploy the latest image from ECR.

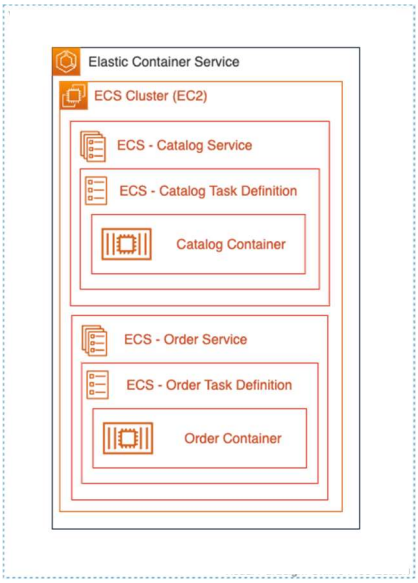
You need to create a Duckerrun.aws.json file and put it into catalog or order GitHub repository. Dockerrun.aws.json file is to get an image from ECR private repository, see the aws documentation on

how to upload an image using a single container from the ECR private repository or you can read in README.md for each service GitHub repository, how to deploy the apps to aws beanstalk.

Note: *Don't forget to include Environment variable in the container.*

Doc: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/single-container-docker-configuration.html

Elastic Container Service



Elastic container service is used for production backend service. Catalogs and service orders should be deployed to ECS via the code pipeline automatically. You have to set up an ECS Cluster with name lks-course-cluster and EC2 as node instances and a VPC with a private subnet (See VPC Service for networking details). You should also create ECS service and task definitions for catalogs and service orders. You will need the information below to create an ECS cluster, Load balancer, and task definition:

- Set up catalog containers using port 8080 for host and port 8000 containers.
- Set up order containers using port 8081 for host and port 8000 containers.
- Catalog and service orders only use 256 Mb of RAM and 1 CPU core to run the application.
- Using the base image that has been uploaded in ECR.
- Use lks-td-catalog and lks-td-order for task definition names.
- Add a container in the task definition with the names lks-catalog-container for catalog service and lks-order-container for order service.
- Create ECS service with the name lks-catalog-service and lks-order-service with service type daemon.
- Use lks-lb-backend for the load balancer with the HTTP protocol.
- Every service use path / for health check.
- Setup container target group with names lks-tg-catalog and lks-tg-order.
- Mapping path /api/v1/course to catalog container in LB.
- Mapping path /api/v1/order to order container in LB.

AWS Documentation:

- *https://docs.aws.amazon.com/AmazonECR/latest/userguide/ECR_on_ECS.html*
- *https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/capacity.html*

Relational Database

The Catalog backend service will use PostgreSQL as a database engine and must be set up for testing and production environment. Please use a suitable RDS template to provide testing and production. Production database must be run in Multi-AZ Cluster with read and write on separate servers. Once complete, you will need to set the database environment for testing and production in the SSM Parameter store. See **SSM Parameter store details** to put the database settings for the parameter key.

Note: *Don't forget to put the initial database name while creating the database. RDS identifier and database name must start with lks, for example, lks-your-database.*

DynamoDB

Order backend service just works on non-relational databases and is optimized to use DynamoDB as the database engine. You need to be prepared and set up DynamoDB for testing and production environments, so you need to create a different table for testing and production with partition key order_id with type string. Use table name lks-order-production for the production table and lks-order-testing for the testing table. Order backend service is not like Catalog service that uses a Parameter store to save the configuration, order backend service just uses an environment variable set up in the container area. This is the environment variable that you can use to save the DynamoDB configuration for the order backend:

```
AWS_REGION=”REGION_DYNAMODB_SETUP”
AWS_DYNAMODB_TABLE_PROD=”TABLE_FOR_PRODUCTION”
AWS_DYNAMODB_TABLE_TEST=”TABLE_FOR_TESTING”
```

Object Storage (S3)

The Course App needed object storage to store assets data such as course catalog images. The backend service that has the functionality to put the assets data into the Object Storage is the catalog service. You must create an S3 Bucket in the **us-east-1** region with bucket name format ***lks-your_name-your_school-name***. The catalog images will be stored in an object with the name "coverImages". The bucket must Block all public access. For security reasons you should only grant access to list get, put, and delete objects only from catalog services and read objects only from CloudFront, please read **CloudFront Service Details** for more information about CloudFront will work with S3. You can use the SSM Parameter store to save the configuration of the S3 bucket and S3 Region for Catalog service, please read **SSM Parameter Service Detail** to see what parameter key is suited for S3 configuration.

CloudFront

The catalog service uses S3 to store "imageCover" data for each course created. The image will be distributed to every client in other countries. For minimizing latency catalog service uses CloudFront as a CDN for all edge locations to deliver images to clients. You need to set up CloudFront to support the catalog service with domain format lks-yourName-school-name and optimize for caching. You need to access the S3 Bucket from CloudFront, you can use a CloudFront origin access identity (OAI) to access the S3 bucket. However, the cover image course cannot be used by the public due to copyright reasons, therefore images distributed with CloudFront must be restricted access from viewers, you can use the Trusted authorization type with key groups in CloudFront. Catalog service

will use a Signed URL from CloudFront using public and private keys. You can create private and public keys with OpenSSL and store the public key to CloudFront key management and put the public key ID and the private key to the SSM Parameter store, see the SSM Parameter store for more detail about storing the CloudFront key.