

# Crowd Counting Demo App Developer's Guide

Authors:

Banovic, Nemanja; Benzuijen, Robin; Kamuf, Sophie;

Makarevich, Anton; Roberto, Alessio; Pan, Jieke;

Merryman, John

February 2020 Mobiquity

## Executive Summary

The Crowd Counting Demo App Developers Guide provides developers and architects a reference to build and deploy a functional mobile application to count people. What we found exciting about this project was the opportunity to rapidly build and prove out the concept that mobile tech can be combined with a combination of cloud and machine learning platforms, to provide a working machine learning example into the hands of an ordinary mobile user. We decided to share documentation and source code for components to engage a wider community interested in this body of work.

The goal is to inspire the development community to learn, extend upon, iterate, and advance the concepts applied in this mobile application to new use cases of this technology solution. While we are very proud to present this example, it is in no way intended to be a perfect example; rather, it is a functional example that can be used and modified by the developer community.

In 2019, Mobiquity designed and engineered a crowd counting mobile application, to prove out the combined power of mobile, video streaming to Kinesis Video Streams, and the machine learning capabilities of Amazon SageMaker. The result is a working mobile application, which allows a user to scan a room and count the number of people, determined by an object detection model, which is trained to recognize 'heads and shoulders' patterns.

There are multiple use cases for which this combination of technology can be effective. The machine learning model can be updated and trained to recognize different objects, such as cars or livestock; however, we encourage development of well-curated and annotated datasets before embarking on a modified use case with SageMaker.

Here is the fully documented solution, a Developer's Guide, with private access to source code repositories for developers to experiment with the technology and build upon this simple working prototype. Please contact the Mobiquity team for access to technical details regarding source code.

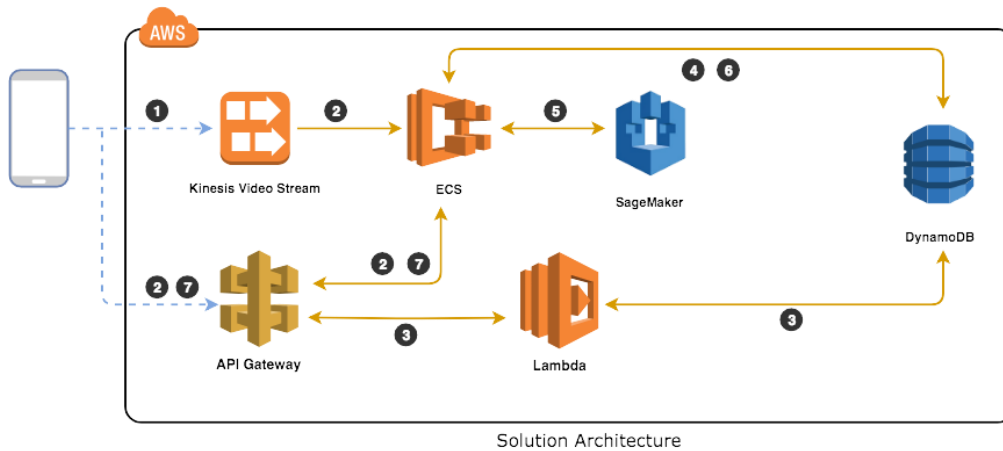
- This Developer's Guide provides developers a toolkit to rapidly build, learn, and deploy a mobile application that demonstrates the power of machine learning.
- It also provides a base for experimentation and adaptation to additional use cases, including but not limited to fixed IOT cameras, streaming applications, and non-mobile sources for video.
- The goal of this guide is to inspire innovation, harness public creativity and engineering ideas, and further the body-knowledge as additional use cases and technical approaches are identified.

|  |           |
|--|-----------|
| <b>EXECUTIVE SUMMARY</b> .....                 | <b>2</b>  |
| <b>SOLUTION ARCHITECTURE</b> .....             | <b>6</b>  |
| <b>CLUSTER CREATION</b> .....                  | <b>7</b>  |
| Task Definition and Container Creation.....    | 7         |
| Cluster Creation .....                         | 11        |
| <b>AMAZON SAGEMAKER</b> .....                  | <b>23</b> |
| Prerequisites.....                             | 23        |
| Pre-processing the Dataset.....                | 23        |
| Training the Object Detection Model .....      | 27        |
| Creating the Model and Endpoint .....          | 30        |
| <b>PANORAMA STITCHING</b> .....                | <b>33</b> |
| KVS Service.....                               | 33        |
| API .....                                      | 33        |
| Panoramic Image Stitching .....                | 33        |
| Configuration.....                             | 34        |
| Build and Release.....                         | 35        |
| Docker OpenCV Container .....                  | 35        |
| Deploy.....                                    | 35        |
| JVM Configuration .....                        | 36        |
| <b>ANDROID</b> .....                           | <b>36</b> |
| <b>SDK Improvements</b> .....                  | <b>36</b> |
| Custom StreamCallbacks .....                   | 36        |
| SDK crashes because of memory allocation ..... | 37        |
| Unsupported codecs on some devices.....        | 37        |
| <b>AWS KVS SDK Integration</b> .....           | <b>37</b> |

|  |           |
|--|-----------|
| <b>AWS KVS SDK Usage</b> .....                     | <b>38</b> |
| General Information .....                          | 38        |
| Streaming Client Class .....                       | 39        |
| Initializing New StreamingService.....             | 40        |
| Streaming Service Class .....                      | 41        |
| Creating the New KVS Client.....                   | 41        |
| Workaround for Memory Related SDK Crash.....       | 42        |
| Handling Stream Callbacks .....                    | 43        |
| Starting and Stopping Streaming.....               | 44        |
| <b>Using the Phone Camera</b> .....                | <b>46</b> |
| Request Camera Permissions .....                   | 47        |
| <b>IOS</b> .....                                   | <b>49</b> |
| <b>Architecture</b> .....                          | <b>49</b> |
| <b>SDK Improvements</b> .....                      | <b>50</b> |
| Missing functionality.....                         | 50        |
| H.264 Encoder.....                                 | 50        |
| Encode CMSampleBufferRef object .....              | 50        |
| Create Compression Session .....                   | 50        |
| Define Session Properties.....                     | 51        |
| Pass Camera Frame to Session.....                  | 51        |
| <b>Generate Kinesis Frame</b> .....                | <b>51</b> |
| <b>How to Import the AwsKvsSdk framework</b> ..... | <b>52</b> |
| How to Import the Framework.....                   | 52        |
| How to Rebuild and Import the Framework .....      | 53        |
| <b>How to Use the SDK</b> .....                    | <b>53</b> |
| Pass Camera Frame to the Stream .....              | 54        |
| Stop Streaming .....                               | 54        |
| Fragments Callback.....                            | 55        |
| <b>Phone Camera Stream</b> .....                   | <b>56</b> |

## Solution Architecture

The following solution architecture diagram illustrates the AWS pipeline for the KVS Demo App. The step process allows a user to stream video from the mobile application to Kinesis Video Streams and then to an ECS instance, which runs a panorama stitching algorithm to compose the video frames into a single file, which is ultimately processed by a SageMaker object detection model. This model is trained to recognize 'head and shoulders' humans in the stitched panorama and return an estimated count of humans in the image along with estimated probability.

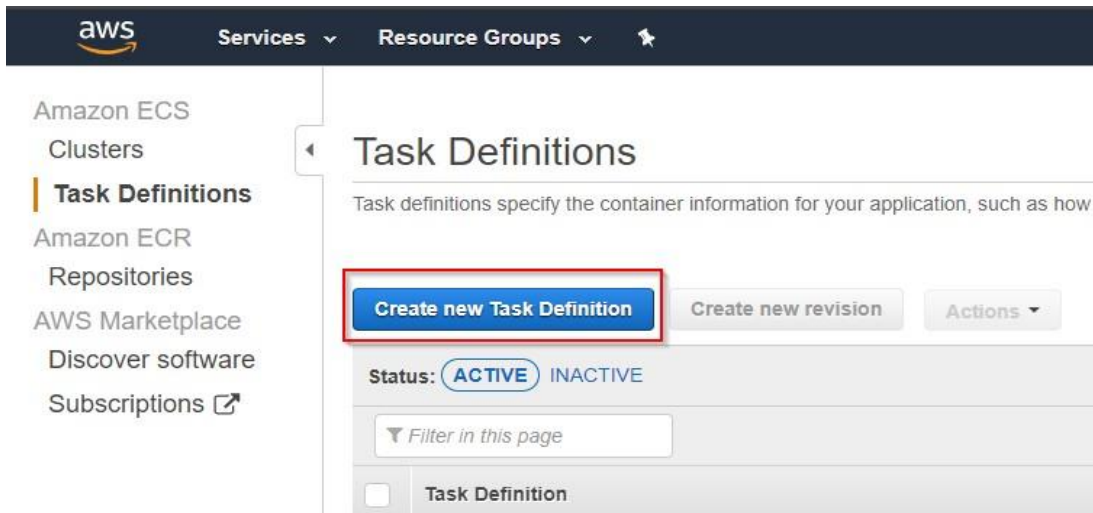


- 1 App creates video stream and streams the video
- 2 App creates video stream name, start and end epoch time stamp
- 3 The video stream name, start and end epoch time stamp is create in the DynamoDB
- 4 ECS picks up the video stream based on the entry in the DynamoDB
- 5 ECS generates the panorama picture, sends it to SageMaker
- 6 ECS gets the result from SageMaker, inserts result in DynamoDB and removes the existing stream name, start and end epoch timestamp to avoid double streaming
- 7 ECS returns the result back via API Gateway

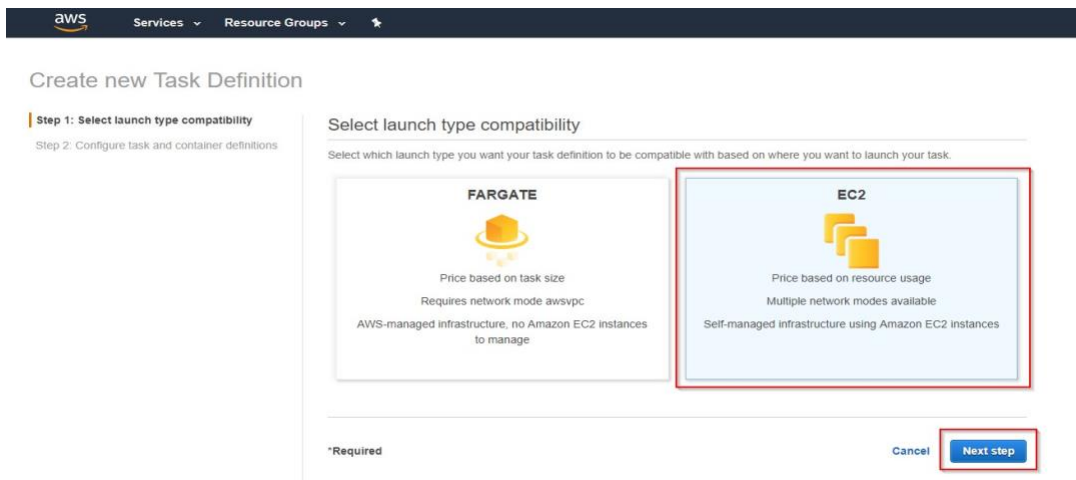
## Cluster Creation

### Task Definition and Container Creation

1. In order to create an ECS instance for the KVS cluster, we need to create a task definition, which we will use in our cluster.



2. We chose EC2 as the instance type. Also, we create the role, which the task will use for communicating with needed services. We leave the <default> network mode.



### Create new Task Definition

Step 1: Select launch type compatibility

**Step 2: Configure task and container definitions**

#### Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can also specify data volumes for your containers to use. [Learn more](#)

Task Definition Name\*

Requires Compatibilities\* EC2

Task Role

Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the IAM Console

Network Mode

If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. <default> is the only supported mode on Windows.

#### Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on your behalf. If you do not have the ecsTaskExecutionRole already, we can create one for you.

Task execution role

### Create new Task Definition

Step 1: Select launch type compatibility

**Step 2: Configure task and container definitions**

#### Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can also specify data volumes for your containers to use. [Learn more](#)

Task Definition Name\*

Requires Compatibilities\* EC2

Task Role

Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the IAM Console

Network Mode

If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. <default> is the only supported mode on Windows.

#### Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on your behalf. If you do not have the ecsTaskExecutionRole already, we can create one for you.

Task execution role

Task size

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (MiB)

16384

The amount of memory (in MiB) used by the task. It can be expressed as an integer using MiB, for example 1024, or as a string using GB, for example '1GB' or '1 gb'.

Task CPU (unit)

10240

The number of CPU units used by the task. It can be expressed as an integer using CPU units, for example 1024, or as a string using vCPUs, for example '1 vCPU' or '1 vcpu'.

Task memory maximum allocation for container memory reservation



Task CPU maximum allocation for containers



Container Definitions

Add container

| Container Name | Image | Hard/Soft memory... | CPU Units | Essential |
|----------------|-------|---------------------|-----------|-----------|
| No results     |       |                     |           |           |

- Assign 16 GB of memory and 10 CPUs per task.
- When adding a container, we specify the container options and the repository, from which the docker image for that container will be pulled. Port 8080 needs to be mapped as seen on the image below.



Add container ✕

▼ Standard

Container name\*  ⓘ

Image\*  ⓘ

Custom image format: {registry-uri}/{namespace}/{image}:{tag}

Private repository authentication\*  ⓘ

Memory Limits (MiB)\*  ⓘ

[Add Soft limit](#)

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the "memory" and "memoryReservation" parameters, respectively, in task definitions.  
EC2 recommends 300-500 MiB as a starting point for web applications.

Port mappings 

| Host port                         | Container port                    | Protocol                         |
|-----------------------------------|-----------------------------------|----------------------------------|
| <input type="text" value="8080"/> | <input type="text" value="8080"/> | <input type="text" value="tcp"/> |

 ⓘ

[Add port mapping](#)

▼ Advanced container configuration

**HEALTHCHECK**

Command  ⓘ

Interval\*  second(s) ⓘ

Timeout\*  second(s) ⓘ

Start period\*  second(s) ⓘ

Retries\*  ⓘ

**ENVIRONMENT**

CPU units  ⓘ

**ENVIRONMENT**

CPU units:

Essential:

Entry point:

Command:

Working directory:

---

**SECURITY**

Privileged:

User:

Docker security options:

---

**RESOURCE LIMITS**

Ulimits:

[Add ulimit](#)

---

**DOCKER LABELS**

Key value pairs:

\* Required Cancel **Add**

## Cluster Creation

1. After creating a container and a task, we now move to creating a cluster. We choose EC2 again.

The screenshot shows the AWS Management Console interface for creating an Amazon ECS cluster. The left-hand navigation menu is visible, with 'Amazon ECS' expanded and 'Clusters' selected, highlighted by a red rectangular box. The main content area is titled 'Clusters' and contains a description: 'An Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests.' Below this, there is a blue information box with the heading 'Opt in to the new ARN and resource ID format' and a link to 'Configure ECS ARN setting'. At the bottom of the main content area, the 'Create Cluster' button is highlighted with a red rectangular box, next to a 'Get Started' button. The 'View' section at the bottom shows 'list' and 'card' options.

## Create Cluster

**Step 1: Select cluster template**

Step 2: Configure cluster

### Select cluster template

The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

**Networking only**

Resources to be created:

- Cluster
- VPC (optional)
- Subnets (optional)

**Powered by AWS Fargate**

**EC2 Linux + Networking**

Resources to be created:

- Cluster
- VPC
- Subnets
- Auto Scaling group with Linux AMI

**EC2 Windows + Networking**

Resources to be created:

- Cluster
- VPC
- Subnets
- Auto Scaling group with Windows AMI

\*Required

Cancel

Next step



- We specify the settings of the EC2 instance and the VPC (a new one may need to be created). We then select “Create” to complete the creation of the cluster.

The screenshot shows the AWS console interface for creating an ECS cluster. The navigation bar at the top includes the AWS logo, 'Services', and 'Resource Groups'. The main heading is 'Create Cluster', with two steps: 'Step 1: Select cluster template' and 'Step 2: Configure cluster'. The 'Configure cluster' section is active and contains the following fields:

- Cluster name\***: A text input field containing 'kvs'.
- Create an empty cluster
- Instance configuration** section:
  - Provisioning Model**: Radio buttons for 'On-Demand Instance' (selected) and 'Spot'. The 'On-Demand Instance' option includes a description: 'With On-Demand Instances, you pay for compute capacity by the hour, with no long-term commitments or upfront payments.' The 'Spot' option includes a description: 'Amazon EC2 Spot Instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. [Learn more](#)'
  - EC2 instance type\***: A dropdown menu showing 'm5.4xlarge'.
  - Manually enter desired instance type
  - Number of instances\***: A text input field containing '1'.
  - EC2 Ami Id\***: A text input field containing 'amzn-ami-2018.03 i-amazon-ecs-optimized [ami-01b70aea4161476b7]'.
  - EBS storage (GiB)\***: A text input field containing '22'.
  - Key pair**: A dropdown menu showing 'None - unable to SSH'. Below it is a note: 'You will not be able to SSH into your EC2 instances without a key pair. You can create a new key pair in the [EC2 console](#)'.


## Networking

Configure the VPC for your container instances to use. A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You can choose an existing VPC, or create a new one with this wizard.

**VPC**   



Check the structure for vpc-073bd6a0fb273fa29 [in the Amazon EC2 console.](#)

**Subnets** 

subnet-09eeab8095758c202 

(10.100.0.0/24) - us-west-2b

assign ipv6 on creation: Disabled

**Security group**   

Rules for sg-043b2d635dd79fb0e [in the EC2 Console.](#)

## Container instance IAM role

The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so container instances that run the agent require the ecsInstanceRole IAM policy and role for the service to know that the agent belongs to you. If you do not have the ecsInstanceRole already, we can create one for you.

**Container instance IAM role**  

## Tags

| Key                                  | Value                                  |
|--------------------------------------|--|
| <input type="text" value="Add key"/> | <input type="text" value="Add value"/> |

\*Required

Cancel

Previous

**Create**

The screenshot shows the 'Launch status' page in the AWS console. At the top, there are navigation tabs for 'Services' and 'Resource Groups'. Below the header, the text reads: 'Your container instances are launching, and it may take a few minutes until they are in the running state and ready to access. Usage hours on your new container instances start immediately and continue to accrue until you stop or terminate them.'

There are two buttons: 'Back' and 'View Cluster', with 'View Cluster' highlighted with a red box. Below this, the status is 'ECS status - 3 of 3 complete kvs'. Three green status boxes are shown:

- ECS cluster**: ECS Cluster kvs successfully created
- ECS Instance IAM Policy**: IAM Policy for the role ecsinstanceRole successfully attached
- CloudFormation Stack**: CloudFormation stack EC2ContainerService-kvs and its resources successfully created

At the bottom, 'Cluster Resources' are listed:

- Instance type: m5.4xlarge
- Desired number of instances: 1
- Key pair: am
- ECS AMI ID: vpx
- VPC: sut
- Subnets: us-
- VPC Availability Zones: sg-
- Security group: EC
- Launch configuration: EC
- Auto Scaling group: EC

- We now create a service, which will run the task that we created previously. We again select EC2 and specify the task definition that we created.

The screenshot shows the 'Cluster : kvs' page in the AWS console. The left sidebar contains navigation options: Amazon ECS, Clusters (highlighted), Task Definitions, Amazon ECR, Repositories, AWS Marketplace, Discover software, and Subscriptions. The main content area shows 'Clusters > kvs' and 'Cluster : kvs'. Below this, it says 'Get a detailed view of the resources on your cluster.' and 'Status ACTIVE'.

Summary statistics for the cluster:

- Registered container instances: 1
- Pending tasks count: 0 Fargate, 0 EC2
- Running tasks count: 0 Fargate, 0 EC2
- Active service count: 0 Fargate, 0 EC2
- Draining service count: 0 Fargate, 0 EC2

Below the statistics are tabs for 'Services', 'Tasks', 'ECS Instances', 'Metrics', 'Scheduled Tasks', and 'Tags'. The 'Services' tab is selected. There are buttons for 'Create' (highlighted with a red box), 'Update', 'Delete', and 'Actions'. Below these are filters: 'Filter in this page', 'Launch type ALL', and 'Service type ALL'. At the bottom, a table header is visible with columns for 'Service Name', 'Status', and 'Service type'.

### Create Service

- Step 1: Configure service
- Step 2: Configure network
- Step 3: Set Auto Scaling (optional)
- Step 4: Review

#### Configure service

A service lets you specify how many copies of your task definition to run and maintain in a cluster. You can optionally use an Elastic Load Balancing load balancer to distribute incoming traffic to containers in your service. Amazon ECS maintains that number of tasks and coordinates task scheduling with the load balancer. You can also optionally use Service Auto Scaling to adjust the number of tasks in your service.

Launch type  FARGATE  EC2 ⓘ

Task Definition Family  
kvs

Revision  
10 (latest)

Cluster kvs ⓘ

Service name kvs-ecs-sagemaker ⓘ

Service type\*  REPLICA  DAEMON ⓘ

Number of tasks 1 ⓘ

Minimum healthy percent 100 ⓘ

Maximum percent 200 ⓘ

## Deployments

Choose a deployment option for the service.

- Deployment type\***
- Rolling update ⓘ
  - Blue/green deployment (powered by AWS CodeDeploy) ⓘ  
This sets AWS CodeDeploy as the deployment controller for the service. A CodeDeploy application and deployment group are created automatically with **default settings** for the service. To change to the rolling update deployment type after the service has been created, you must re-create the service and select the "rolling update" deployment type.

## Task Placement

Lets you customize how tasks are placed on instances within your cluster. Different placement strategies are available to optimize for availability and efficiency.

**Placement Templates** AZ Balanced Spread Edit

This template will spread tasks across availability zones and within the availability zone spread tasks across instances. [Learn more.](#)

**Strategy:** spread(attribute:ecs.availability-zone), spread(instanceid)

ⓘ **Tagging requires that you opt in to the new ARN and resource ID format.**  
The IAM user/role has not opted in to the new ARN format. Opt-in to the new format to use this feature. [Manage your opt-in settings.](#)

\*Required

Cancel

Next step



- Continuing with configuring the network, we chose a VPC or create a new one along with security group, where the service of this cluster will run.

**Create Service**

Step 1: Configure service  
**Step 2: Configure network**  
 Step 3: Set Auto Scaling (optional)  
 Step 4: Review

**Configure network**

VPC and security groups  
 VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC\* vpc-073bd6a0fb273fa29 (10.100.0....) ⓘ

Subnets\* subnet-09eeab6095756c202 (10.100.0.0/24) - us-west-2b assign ipv6 on creation: Disabled ⓘ

Security groups\* sg-043b2d635d79fb0e **Edit** ⓘ

Auto-assign public IP DISABLED ⓘ

---

**Configure security groups**

A security group is a set of firewall rules that control the traffic for your task. On this page, you can add rules to allow specific traffic to reach your task, or you can choose to use an existing security group. [Learn more.](#)

Assigned security groups  Create new security group  
 **Select existing security group**

Existing security groups  
 All existing security groups for the VPC of this cluster are listed below.

1 selected

| Security group ID                                       | Name    | Description                | Actions                     |
|---|---------|----------------------------|-----------------------------|
| <input checked="" type="checkbox"/> sg-043b2d635d79fb0e | ivs-ecs | Port 8080 from anywhere    | <a href="#">Copy to new</a> |
| <input type="checkbox"/> sg-0c9934bd3d3d6bc24f9         | default | default VPC security group | <a href="#">Copy to new</a> |

Inbound rules for selected security groups

| Security group ID   | Type | Protocol | Port range | Source  |
|---------------------|------|----------|------------|---------|
| sg-043b2d635d79fb0e | HTTP | TCP      | 80         | 0.0.0.0 |

Cancel **Save**

- Because a static public IP is needed, we create a network load balancer, which is attached to a service.

#### Health check grace period

If your service's tasks take a while to start and respond to ELB health checks, you can specify a health check grace period of up to 7,200 seconds during which the ECS service scheduler will ignore ELB health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up. This is only valid if your service is configured to use a load balancer.

Health check grace period  ⓘ

#### Load balancing

An Elastic Load Balancing load balancer distributes incoming traffic across the tasks running in your service. Choose an existing load balancer, or create a new one in the [Amazon EC2 console](#).

#### Load balancer type\*

- None  
Your service will not use a load balancer.
- Application Load Balancer  
Allows containers to use dynamic host port mapping (multiple tasks allowed per container instance). Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.
- Network Load Balancer  
A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection (OSI) model. After the load balancer receives a request, it selects a target from the target group for the default rule using a flow hash routing algorithm.
- Classic Load Balancer  
Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

**Service IAM role** Task definitions that use the awsvpc network mode use the AWSServiceRoleForECS service-linked role, which is created for you automatically. [Learn more.](#)

Load balancer name  ⓘ

## Container to load balance

work\_open-cv : 8080

Remove ✕

Production listener port create new 8080 ⓘ

Production listener protocol TCP

Target group name create new ecs-kvs-kvs-ecs-sager ⓘ

Target group protocol TCP ⓘ

Target type ip ⓘ

Health check protocol TCP

## Service discovery (optional)

Service discovery uses Amazon Route 53 to create a namespace for your service, which allows it to be discoverable via DNS.

Enable service discovery integration

\*Required

Cancel

Previous

Next step



## Create Service

Step 1: Configure service

Step 2: Configure network

**Step 3: Set Auto Scaling (optional)**

Step 4: Review

### Set Auto Scaling (optional)

Automatically adjust your service's desired count up and down within a specified range in response to CloudWatch alarms. You can modify your Service Auto Scaling configuration at any time to meet the needs of your application.

- Service Auto Scaling
- Do not adjust the service's desired count
  - Configure Service Auto Scaling to adjust your service's desired count

\*Required

Cancel

Previous

Next step

### Create Service

- Step 1: Configure service
- Step 2: Configure network
- Step 3: Set Auto Scaling (optional)
- Step 4: Review**

#### Review Edit

**Cluster** kvs  
**Launch type** EC2  
**Task Definition** kvs:10  
**Service name** kvs-ecs-sagemaker  
**Service type** REPLICHA  
**Number of tasks** 1  
**Minimum healthy percent** 100  
**Maximum percent** 200

#### Configure network Edit

**VPC Id** vpc-073bd6a0fb273fa29  
**Subnets** subnet-09eeab8095758c202  
**Selected security groups** sg-043b2d635dd79fb0e  
**Auto assign IP** DISABLED  
**Container Name:** work\_open-cv  
**Container Port:** 8080  
**ELB Name:** kvs  
**Target Group:** ecs-kvs-kvs-ecs-sagemaker  
**Health check protocol:** TCP  
**Listener Port:** 8080

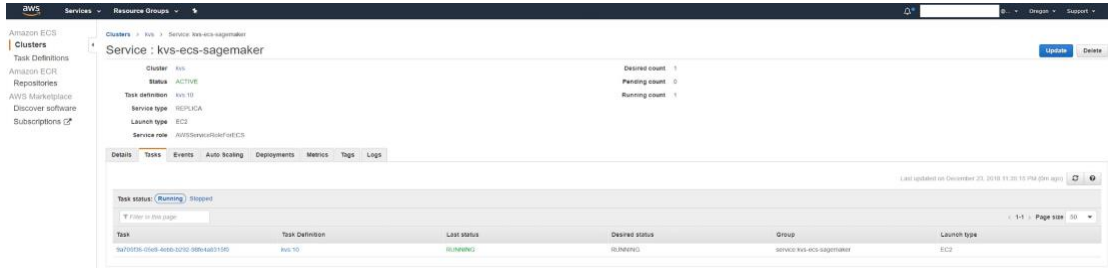
#### Set Auto Scaling (optional) Edit

not configured

Cancel Previous **Create Service**

**Launch Status**  
ECS Service status - 3 of 3 completed  
Configure Task Networking  
Create Load Balancer  
Target Group: ecs-kvs-kvs-ecs-sagemaker  
Target Group created  
Listener: 8080 - TCP  
Listener created  
Create Service  
Create service: kvs-ecs-sagemaker  
Service created  
Additional integrations you can connect to your ECS service  
Code Pipeline  
Create a pipeline of

6. After creation of the service, your cluster is fully running with the KVS recognition software.



## Amazon SageMaker

**Source:** Please contact the Mobiquity team for access to technical details.

### Prerequisites

Please follow the following instructions to mount an Amazon Elastic File System (EFS) to your SageMaker notebook:

<https://aws.amazon.com/blogs/machine-learning/mount-an-efs-file-system-to-an-amazon-sagemaker-notebook-with-lifecycle-configurations/>

The EFS will allow you to easily share the downloaded data with your team members.

In addition, you will need basic knowledge of SageMaker, such as how to start a notebook instance and upload files.

1. Login to SageMaker and create a new notebook instance. Dock the previously created EFS to your instance.
2. Within the notebook instance, upload the two provided Jupyter notebooks: Preprocess-COCO-dataset.ipynb and SSD-person-detection.ipynb.

### Pre-processing the Dataset

The details below explain the pre-processing steps found in the code base.

1. Start the SageMaker session and create an S3 bucket, where you will store the train and validations sets.

```
%%time
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role() print(role)
sess = sagemaker.Session()

bucket = 'crowd-
counting' prefix =
'COCO-data'
```

Download the publicly available COCO dataset from 2017. (You can explore the dataset here: <http://cocodataset.org/>)

```
import os
import urllib.request

#path of efs
path = "/home/ec2-user/efs/"

def download(url):

    filename = path + url.split("/")[-1]
    if not os.path.exists(filename):
        urllib.request.urlretrieve(url, filename)

# MSCOCO image and annotation files
download('http://images.cocodataset.org/zips/train2017.zip')
download('http://images.cocodataset.org/zips/val2017.zip')
download('http://images.cocodataset.org/annotations/annotations_
trainval2017.zip')
```

2. Unzip the downloaded dataset and then remove the zip files.

```
%%bash
unzip -d ~/efs/ ~/efs/train2017.zip
unzip -d ~/efs/ ~/efs/val2017.zip
unzip -d ~/efs/ ~/efs/annotations_trainval2017.zip

rm ~/efs/train2017.zip ~/efs/val2017.zip
~/efs/annotations_trainval2017.zip
```

3. Create folders in the EFS to store the data and annotation files.

```
mkdir ~/efs/train_generated ~/efs/val_generated ~/efs/train
~/efs/train_annotation ~/efs/validation
~/efs/validation_annotation ~/efs/test ~/efs/test_annotation
```

- The COCO dataset contains 80 object categories. Since we are looking to train the model to count people, we will filter the dataset for images, which include at least one object in the category “person.” We then clean up the annotation files by removing annotations for the other object categories.

```
import json import logging
def create_dataset(file_name, json_destination): with
open(file_name) as f:
    js = json.load(f) images = js['images']
    categories = js['categories'] annotations =
    js['annotations'] for i in images:
        jsonFile = i['file_name']
        jsonFile = jsonFile.split('.')[0]+'.json'

        line = {}
        line['file'] = i['file_name'] line['image_size'] =
        [{
            'width':int(i['width']),
            'height':int(i['height']), 'depth':3
        }]
        line['annotations'] = [] line['categories'] = []
        for j in annotations:
            if j['image_id'] == i['id'] and j['category_id']
            == 1 and len(j['bbox']) > 0:
                line['annotations'].append({
                    'class_id':int(0),
                    'left':int(j['bbox'][0]),
                    'top':int(j['bbox'][1]),
                    'width':int(j['bbox'][2]),
                    'height':int(j['bbox'][3])
                })
                line['categories'].append({ 'class_id':int(0),
                    'name':'person'
                })
        if line['annotations']:
            with open(os.path.join(json_destination, jsonFile),'w') as p:
                json.dump(line,p)
```

- create\_dataset('/home/ec2- user/efs/annotations/instances\_val2017.json', '/home/ec2- user/efs/val\_generated')
- create\_dataset('/home/ec2- user/efs/annotations/instances\_train2017.json', '/home/ec2- user/efs/train\_generated')



- The dataset is then split into a train and validation set.

```
import os
import json
jsons_val = os.listdir('/home/ec2-user/efs/val_generated')
jsons_train = os.listdir('/home/ec2-user/efs/train_generated')

print ('There are {} validation images with annotation files and
class person'.format(len(jsons_val)))
print ('There are {} train images with annotation files and
class person'.format(len(jsons_train)))
```

- In order to reduce the training time, select the first 20,000 images with class “person” for the train set and 2,000 images for the validation set. Then move the training and validation files to their respective folders on the EFS.

```
import shutil

train_jsons = jsons_train[:20000]
val_jsons = jsons_val

#Moving training files to the training folders
for i in train_jsons:
    image_file = '/home/ec2-user/efs/train2017/'+i.split('.')[0]+'.jpg'
    shutil.move(image_file, '/home/ec2-user/efs/train/')
    shutil.move('/home/ec2-user/efs/train_generated/'+i,
'/home/ec2-user/efs/train_annotation/')

#Moving certain training files to the validation folders
for i in val_jsons:
    image_file = '/home/ec2-user/efs/train2017/'+i.split('.')[0]+'.jpg'
    shutil.move(image_file, '/home/ec2-user/efs/validation/')
    shutil.move('/home/ec2-user/efs/train_generated/'+i,
'/home/ec2-user/efs/validation_annotation/')
```

9. Upload the train and validation sets to S3.

```
%%time

train_channel = prefix + '/train'
validation_channel = prefix + '/validation'
train_annotation_channel = prefix + '/train_annotation'
validation_annotation_channel = prefix +
'/validation_annotation'

sess.upload_data(path='/home/ec2-user/efs/train', bucket=bucket,
key_prefix=train_channel)
sess.upload_data(path='/home/ec2-user/efs/validation',
bucket=bucket, key_prefix=validation_channel)
sess.upload_data(path='/home/ec2-user/efs/train_annotation',
bucket=bucket, key_prefix=train_annotation_channel)
sess.upload_data(path='/home/ec2-
user/efs/validation_annotation', bucket=bucket,
key_prefix=validation_annotation_channel)
```

10. When you are done with this project, you can also choose to remove the folders from EFS.

```
%%bash
rm -rf ~/efs/train ~/efs/train_annotation ~/efs/validation
~/efs/validation_annotation ~/efs/test ~/efs/test_annotation
```

## Training the Object Detection Model

1. Settings. Start a SageMaker session and specify the bucket and prefix, where the train and validation data are located in S3.

```

%%time
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)
sess = sagemaker.Session()

bucket = 'crowd-counting'
prefix = 'COCO-data'

from sagemaker.amazon.amazon_estimator import get_image_uri

```

2. Data Location. Specify the output location and the location of the following 4 channels:

- Train data
- Train annotation
- Validation data
- Validation annotation

```

train_channel = prefix + '/train'
validation_channel = prefix + '/validation'
train_annotation_channel = prefix + '/train_annotation'
validation_annotation_channel = prefix +
'/validation_annotation'

s3_train_data = 's3://{}/{}'.format(bucket, train_channel)
s3_validation_data = 's3://{}/{}'.format(bucket,
validation_channel)

```

```

s3_train_annotation = 's3://{}/{}'.format(bucket,
train_annotation_channel)
s3_validation_annotation = 's3://{}/{}'.format(bucket,
validation_annotation_channel)

s3_output_location = 's3://{}/{}-model-output'.format(bucket,
prefix)

```

3. Training. The SageMaker Object Detection algorithm uses a Single Shot multibox Detector (SSD) framework. Select `train_instance_type='ml.p3.8xlarge'`, which will train the model in approximately 6hrs. You can choose a smaller or larger instance type depending on your needs. The larger the instance, the quicker the model will be trained. Please note that larger instance types also come with more costs. See the pricing page for more details.

```
od_model = sagemaker.estimator.Estimator(
    training_image,
    role,
    train_instance_count=1,
    train_instance_type='ml.p3.8xlarge',
    train_volume_size = 50,
    train_max_run = 360000,
    input_mode = 'File',
    output_path=s3_output_location,
    sagemaker_session=sess)
```

4. Hyperparameters. We changed the following default hyperparameters:
  - `num_classes=1` → we are training on 1 category (person) and not on 80 object categories
  - `num_training_samples=20000` → we are training on 20,000 images

If you are interested in tuning the remaining hyperparameters, please refer to this guide:

<https://docs.aws.amazon.com/sagemaker/latest/dg/object-detection-api-config.html>

```
od_model.set_hyperparameters(base_network='resnet-50',
                             use_pretrained_model=1,
                             num_classes=1,
                             mini_batch_size=64,
                             epochs=140,
                             learning_rate=0.002,
                             lr_scheduler_step='80',
                             lr_scheduler_factor=0.1,
                             optimizer='sgd',
                             momentum=0.9,
                             weight_decay=0.0005,
                             overlap_threshold=0.5,
                             nms_threshold=0.45,
                             image_shape=512,
                             label_width=600,
                             num_training_samples=20000)
```

5. Specify the data to be used for training.

```

train_data = sagemaker.session.s3_input(s3_train_data,
distribution='FullyReplicated', content_type='image/jpeg',
s3_data_type='S3Prefix')

validation_data = sagemaker.session.s3_input(s3_validation_data,
distribution='FullyReplicated', content_type='image/jpeg',
s3_data_type='S3Prefix')

train_annotation =
sagemaker.session.s3_input(s3_train_annotation,
distribution='FullyReplicated', content_type='image/jpeg',
s3_data_type='S3Prefix')

validation_annotation =
sagemaker.session.s3_input(s3_validation_annotation,
distribution='FullyReplicated', content_type='image/jpeg',
s3_data_type='S3Prefix')

```

6. You can now go ahead and train the model. This will take around 6 hours with the provided settings.

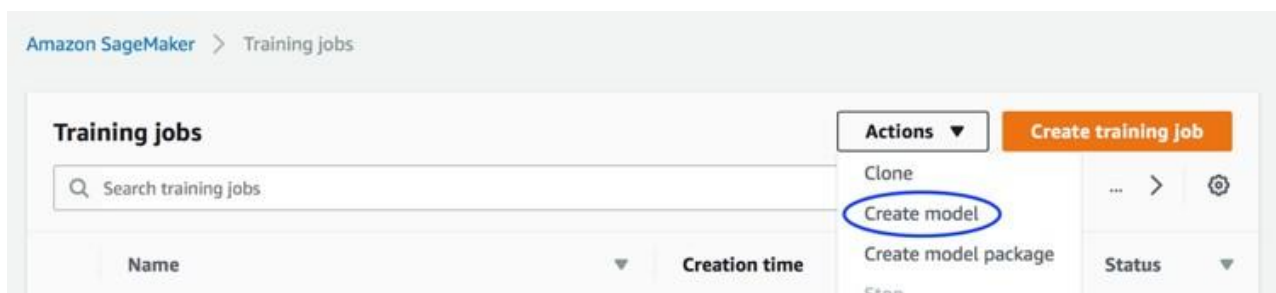
```

od_model.fit(inputs=data_channels, logs=True, job_name='od-coco-
20000-05112018-v2')

```

### Creating the Model and Endpoint

1. You can find the trained model under Training → Training jobs.
2. Select the training job, then Actions → “Create model”



3. Provide a model name but leave all other settings on default. Then press “Create model” at the bottom.

## Create model

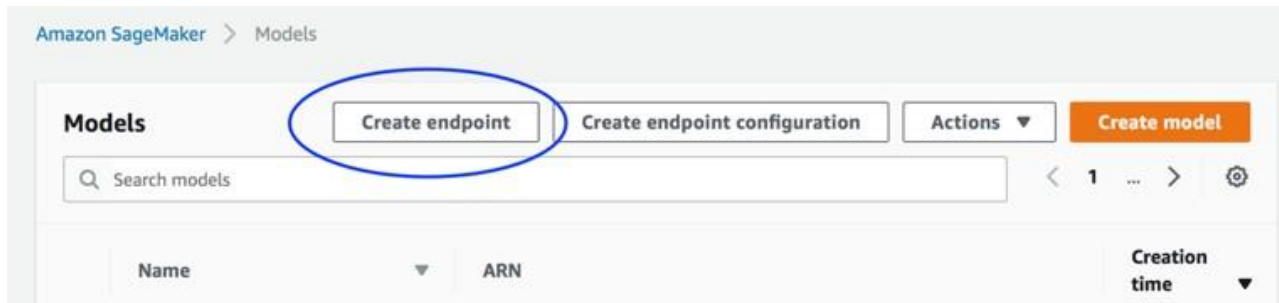
To deploy a model to Amazon SageMaker, first create the model by providing the location of the model artifacts and inference code. See [Deploying a Model on Amazon SageMaker Hosting Services](#) [Learn more about the API](#)

### Model settings

Model name

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

4. You can find the created model under Inference → Models. Select the model and press “Create endpoint.”



5. Provide a name for the endpoint and select “Create a new endpoint configuration.”

### Endpoint

**Endpoint name**  
Your application uses this name to access this endpoint.

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

### Attach endpoint configuration

**Use an existing endpoint configuration**  
Use an existing endpoint configuration or clone an endpoint configuration.

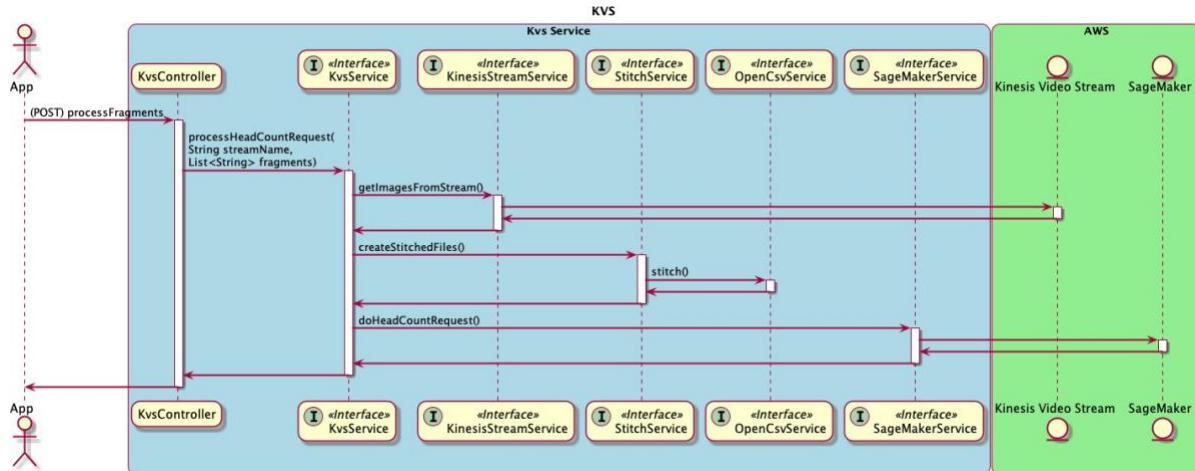
**Create a new endpoint configuration**  
Add models and configure the instance and initial weight for each model.

6. At the bottom, you can then press “Create endpoint configuration.”
7. You can find the created endpoint under Inference → Endpoints.

## Panorama Stitching

### KVS Service

The KVS Service is an OpenCV docker instance containing a Java SpringBoot application, which returns a SageMaker headcount prediction from a panoramic image by processing a Kinesis Video Stream. The video stream is created by the user (via the mobile app). The details of the video stream (name and fragments) are sent to the KVS Service via Rest API.



### API

<http://server.com:8080/info> (GET)

Health check endpoint.

<http://server.com:8080/processFragments> (POST)

```
{
  "streamName": "myStream",
  "fragments": [
    "001",
    "002",
    "003",
    "004"
  ]
}
```

### Panoramic Image Stitching

The panoramic image is created by stitching frames using regular intervals from the video stream. The stitching is done by OpenCV.



A stream can have many fragments and each fragment can contain many images. For example, in case the service needs to process 20 fragments and each fragment contains 25 frames the total frames to be processed would be 500 (20\*25). In order to speed up the process an equal interval is used assuming the user has a steady velocity when recording to the stream (when moving the device). The interval is calculated by determining the number of fragments and the number of images to use. Both numbers can be configured because higher numbers will impact performance.

## Configuration

The AWS role is provided by the deployment solution, by using the DefaultAWSCredentialsProviderChain. The region is provided in the configuration.

The role used by the container must have access to the following resources (refer to configuration):

- SageMaker
- Kinesis-Video-Stream

```
# Aws Region
aws.region=us-west-2

# Accuracy settings (max amount calculated by
interval). aws.maxFragmentsToProcess=10
aws.maxFramesToStitch=10

# SageMaker aws.sageMakerContentType=application/x-image
aws.sageMakerEndPoint=SSD-20k-endpoint2018-11-
07-16-02-54 aws.predictionThreshold=0.2

# Threads to do parallel stitching.
aws.stitchThreads=50
aws.fragmentThreads=2
aws.sageMakerThreads=50

# Maximum image size (SageMaker max image size). The
service will resize the panoramic image to avoid
sending larger images. aws.maxImageSize=6144000

# Build version (to show when using the info endpoint).
aws.version=unset
```

All application specific configurations can be overridden by using Java startup parameters. For example:

```
-Daws.version=1.0.1 -Daws.predictionThreshold=0.195
```

## Build and Release

Building the application requires Java (8 or higher) and Maven (3). Navigate to the root of the project and execute:

```
mvn clean install
```

## Docker OpenCV Container

OpenCV needs to be available on the docker instance to be able to create a panoramic image. The docker instance needs to be built and makes sure that the required installation is done. To build the docker instance, navigate to the work directory from the root, and then execute the init.sh script.

```
cd work
./init.sh
docker-compose build
docker-compose up open-cv
```

**Note:** If you still have containers running execute: `docker-compose stop/ps/rm`

## Deploy

When a docker container is built it can be uploaded to the AWS registry to be used. To deploy the container into the registry you need to login first. The command below only works when setting correct credentials and having the AWS cli installed (Please do not set AWS key and secret key in the profile or as environmental values).

```
aws ecr get-login
```

This command will produce a login to push the docker instance which needs to be copied and executed again. If you get a failure when executing, please remove the following part from the provided command:

```
-e none
```

The login is valid for 12 hours.

Now the docker instance can be deployed. First it needs to be tagged and then it can be uploaded to the registry.

```
docker tag work_open-cv 473293451041.dkr.ecr.us-west-2.amazonaws.com/kvs-service-repo:latest
```

```
docker push 473293451041.dkr.ecr.us-west-2.amazonaws.com/kvs-service-repo:latest
```

## JVM Configuration

Please note that OpenCV and reading streams are CPU and memory intensive. The docker instance needs to be configured according the specifications of the server. Java memory can be increased by specifying a JAVA\_OPTS environmental variable, i.e:

```
JAVA_OPTS="$JAVA_OPTS -Xmx6144m -Xms4096m -Dfile.encoding=UTF-8 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled"
```

## Android

**Source:** Please contact the Mobiquity team for access to technical details.

### SDK Improvements

#### Missing Functionality

The original Android producer SDK code base included in the AWS Android SDK is missing the ability to set a custom implementation of `StreamCallbacks`. It is therefore not possible to get notifications about stream fragments, which have been uploaded to KVS.

It also has several known restrictions and bugs, such as:

- The available memory calculation produces wrong result on some devices and causes crashes during heap allocation
- H.264 encoders used on some Android models seem to not be supported

#### Custom StreamCallbacks

To make changes to the Android SDK we forked the repo and made the following changes:

- Put additional argument for StreamCallback in `createKinesisVideoClient` method of `KinesisVideoAndroidClientFactory` as shown on the following changelog:

```

public static KinesisVideoClient createKinesisVideoClient(final @NonNull Context context,
-     final @NonNull AWSCredentialsProvider credentialsProvider)
+     final @NonNull AWSCredentialsProvider credentialsProvider, final StreamCallbacks streamCallbacks)
    throws KinesisVideoException {
-     return createKinesisVideoClient(context, Regions.DEFAULT_REGION, credentialsProvider);
+     return createKinesisVideoClient(context, Regions.DEFAULT_REGION, credentialsProvider, streamCallbacks);
}

```

- Pass streamCallbacks as optional parameter to the `AndroidKinesisVideoClient` constructor:

```

        @NonNull final Context context,
        @NonNull final KinesisVideoClientConfiguration configuration,
        @NonNull final KinesisVideoServiceClient serviceClient,
-     @NonNull final ScheduledExecutorService executor) {
+     @NonNull final ScheduledExecutorService executor,
+     final StreamCallbacks streamCallbacks) {
    super(log,
        configuration,
        serviceClient,
-     executor);
+     executor,
+     streamCallbacks);
}

```

- In base `NativeKinesisVideoClient` we checked the passed streamCallbacks parameter. If it equals null, then the default implementation is used

### SDK crashes because of memory allocation

- We did not fix this issue on the SDK side, instead we used alternative overload of
- `createKinesisVideoClient` passing custom DeviceInfo with fixed memory size. This is covered in the SDK integration section of this documentation.

### Unsupported codecs on some devices

Even though some of our test devices were affected by this issue, resolving it was not in the scope of this project.

## **AWS KVS SDK Integration**

In order to add an AWS SDK component to an Android project, a corresponding line in the module's build.gradle file needs to be included. For example, to add AWS KVS SDK, the gradle command should look like this:

```
implementation ("com.amazonaws:aws-android-sdk-
kinesisvideo:$aws_version")
```

But since we utilized our own modified SDK version, we could not use this process. Instead we followed the following steps to add our own compiled aar file to the project:

1. Copy `aws-android-sdk-kinesisvideo-debug.aar` to the `app/libs/` folder of the Android project;
2. Add libs folder to the path, where the system is going to look for dependencies by including the following in the module's build.gradle file:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

3. Add a reference to the dependencies section of the aar file itself:

```
implementation (name:'aws-android-sdk-kinesisvideo-debug',
ext:'aar')
```

4. The KVS SDK still has dependencies related to the core AWS SDK, but we did not modify it and still used the default version:

```
implementation ("com.amazonaws:aws-android-sdk-
auth-core:$aws_version")
```

## AWS KVS SDK Usage

### General Information

The app has been developed with the Model-View-ViewModel design pattern in mind and used Dagger for injecting all the required dependency implementations in the ViewModels.

All the functionality to access KVS and stream video is wrapped in the StreamingService class. We create a new instance of it for every streaming session, meaning we also create a new instance of all the underlying KVS from the SDK.

StreamingService is hosted by the StreamingClient. StreamingClient is a singleton class responsible for asynchronous creation and stopping of new StreamingService instances.

## Streaming Client Class

StreamingClient class is very simple and used to only hold and manage StreamingService instances:

```
class StreamingClient(private val context: Context)

    { var service: StreamingService? = null

    fun free() {
        if
            (service!=null
            ){
                service?.free(
                ) service =
                null
            }
        }

    fun initialize() {
        service = StreamingService(context)
    }

    var hasCameraAccess: Boolean = false
}
```

## Initializing New StreamingService

StreamingService could be initialized from within the StreamingClient class. In our case we pass StreamingClient as a dependency to the DemoOneViewModel (the home view model for the demo) and use the following function:

```
fun initKvsClient() = launch
    {
        changeTryItState(false)

        val checkJob = async(Dispatchers.IO) {
            connectionChecker.isInternetAvailable()
        }

        if (!checkJob.await()) {
            navigator.showDialog(
                stringsProvider.getString(R.string.alert_noconnection_title),
                stringsProvider.getString(R.string.alert_noconnection_message)
            )

            return@launch
        }

        val job = async(Dispatchers.IO)
        { streamingClient.free()
        }
        job.await()
        streamingClient.initialize()
        changeTryItState(true)
    }
}
```

The method contains two asynchronous jobs: first it checks for internet connection and if a connection is available, it initializes a new instance of StreamingService. During initialization we first call the streamingClient.free() method to stop the current instance and free resources. Both the FreeStream() method and the new native KVS client created in the KVS SDK can take some time and block the running thread, so we run everything asynchronously using Kotlin coroutines to not block the UI. In the meantime, we show a “One moment...” text and disable the “Try it button” to let the user know that something is happening in the background.

**Note:** Checking for an internet connection is very important because the KVS SDK will crash when it tries to access the KVS API without an internet connection.

### Streaming Service Class

The `StreamingService` class wraps all the functionality of the KVS SDK, allowing to create a new native `KvsClient`, open new stream, start and stop streaming from the device.

### Creating the New KVS Client

The native KVS Client is the key component of the `StreaminService` as it manages streams and handles all the underlying communication with the KVS SDK. We use the `'KinesisVideoAndroidClientFactory'` class of the SDK to create a new instance of the `kvsClient`.

Every instance of `StreamingService` wraps its own instance of `kvsClient` and creates it during initialization:

```
private val kvsClient =
KinesisVideoAndroidClientFactory.createKinesisVideoClient(context,
    configuration,
    defaultDeviceInfo(),
    log,
    executor,
    kvsStreamCallbacks)
```

Where `configuration` – is an instance of `'KinesisVideoClientConfiguration'` and contains information about user credentials and the Amazon environment:

```
private val configuration =
KinesisVideoClientConfiguration.builder()
    .withRegion(region.getName())

.withCredentialsProvider(kinesisVideoCredentialsProvider)
    .withLogChannel(outputChannel)
    .withStorageCallbacks(DefaultStorageCallbacks())
    .build()
```



Where `kinesisVideoCredentialsProvider` is an instance of `AwsCredentialsProvider`. We use the simplest version in the app accepting access and secret keys:

```
private val credentialsProvider =
    AwsBasicCredentialsProvider(BasicAWSCredentials("<your-access-
key>", "<your-secret-key>"))
    private val region = Regions.US_WEST_2
```

### Workaround for Memory Related SDK Crash

As mentioned earlier, in order to avoid SDK crashes related to wrong memory calculation, we have to use overload of `KinesisVideoAndroidClientFactory.createKinesisVideoClient()` accepting device info parameter where we can provide our own value for memory allocation:

```
private fun defaultDeviceInfo(): DeviceInfo
    { return DeviceInfo(
        DEVICE_VERSION,
        N,
        DEVICE_NAME,
        defaultStorageInfo(
        ), STREAMS_COUNT,
        defaultDeviceTags())
    }

private fun defaultStorageInfo(): StorageInfo
    { return StorageInfo(0,

StorageInfo.DeviceStorageType.DEVICE_STORAGE_TYPE_IN_ME
        M, defaultMemorySize(),
        SPILL_RATIO_90_PERCENT,
        STORAGE_PATH)
    }

private fun defaultMemorySize(): Long
    { return MAX_STORAGE_SIZE_MEGS
    }
```

## Handling Stream Callbacks

To handle stream callbacks and collect information such as fragments ids and timestamps we created a custom implementation of the StreamCallbacks interface:

```
class KvsStreamCallbacks: StreamCallbacks {
    private var fragments: MutableList<KinesisVideoFragmentAck>
= mutableListOf()
    private val tag = "KvsStreamCallbacks"
    private var isRecording = false
    private var startTimestamp = Long.MAX_VALUE

    fun startFragmentsRecording(currentTimestamp: Long) {
        startTimestamp = currentTimestamp
        fragments = mutableListOf()
        isRecording = true
    }

    fun stopRecording(): List<String>{
        isRecording = false
        fragments.sortBy { f -> f.timestamp }
        return fragments
            .map { f -> f.sequenceNumber }.distinct()
    }
}
```

```
    override fun fragmentAckReceived(fragmentInfo:
KinesisVideoFragmentAck) {
        Log.i(tag, "receive fragment type:
${fragmentInfo.ackType}, fragment:
${fragmentInfo.sequenceNumber}")
        if (isRecording &&
            fragmentInfo.timestamp > startTimestamp &&
            fragmentInfo.ackType.intType ==
FragmentAckType.FRAGMENT_ACK_TYPE_PERSISTED) {
            fragments.add(fragmentInfo)
        }
    }
...
}
```

Every time we receive AckReceived event we add fragments information to the corresponding collection and return it on stop streaming request.

## Starting and Stopping Streaming

StreamingService has corresponding public methods to start and stop streaming and start recording fragments, which we can call from the StreamingViewModel:

```

fun startStreaming(previewTexture: SurfaceTexture) = try {

    cameraMediaSource!!.setPreviewSurfaces(Surface(previewTexture)) cameraMediaSource!!.start()

    } catch (e: KinesisVideoException) {
        Log.e("KvsService.Start",
            "unable to start
streaming
")
            throw RuntimeException("unable to start streaming",
e)
        }

    fun startFragmentsRecording(timestamp: Long) {
        kvsStreamCallbacks.startFragmentsRecording(
            timestamp)
    }

    fun stopStreamingAndGetFragments():
        Capture? { try {
            cameraMediaSource!!.stop()
            kvsClient.stopAllMediaSources() return
        } catch (e: Exception) {
            Log.e("KvsService.Stop", "unable to stop
streaming",
e)
            return null
        }
    }
}

```

From the ViewModel we call start streaming as soon as StreamingView is loaded (it is the only way to have camera preview working with current the SDK implementation) and call start recording fragments when the user presses the button and starts actual streaming. We use a reactive based timer to stop streaming after 10 seconds:

```

fun startTimer(){
    isTipsVisible = false

    streamingClient.service?.startFragmentsRecording(System
m.currentTimeMillis()-activationTime)
    timeValue.set("00 :
    $RECORD_TIME")
    disposeBag.add(observabl
    e
        .interval(1,1,TimeUnit.SECONDS)
        .take(RECORD_TIME.toLong())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe {
            timeValue.set("00 : 0${RECORD_TIME -
(it+1)}
")
            if (it == (RECORD_TIME-
1).toLong()) {
                stopStreaming()
            }
        })
    })
}

private fun
stopStreaming(
){ val capture
=
streamingClient.service?.stopStreamingAndGetFragme
nts()
captureService.setRecentCapture(capture)
navigator.startFragment(AppFragments.Posts
can, false,
true)
}

```

## Using the Phone Camera

We use the default implementation of `AndroidCameraSource` from the SDK passing all the required camera parameters to `CameraMediaSourceConfiguration`:

```

val cameraConfig = getCameras(kvsClient)[0] val
alltypes = getSupportedMimeTypes()

val mimeType = alltypes[0]

mediaSourceConfiguration =
AndroidCameraMediaSourceConfiguration(
    AndroidCameraMediaSourceConfiguration.builder()
        .withCameraId(cameraConfig.cameraId)
        .withEncodingMimeType(mimeType.mimeType)
        )

.withHorizontalResolution(videoResolution.width)

.withVerticalResolution(videoResolution.height)

.withCameraFacing(cameraConfig.cameraFacing)

.withIsEncoderHardwareAccelerated(cameraConfig.isEncoderHardwareAccelerated)
        .withFrameRate(videoFramerate)

.withRetentionPeriodInHours(retentionPeriod)
        .withEncodingBitRate(videoBitrate)

.withCameraOrientation(cameraConfig.cameraOrientation)

.withNalAdaptationFlags(StreamInfo.NalAdaptationFlags.
NAL_ADAPTATION_ANNEXB_CPD_AND_FRAME_NALS)
        .withIsAbsoluteTimecode(false))
cameraMediaSource = kvsClient
        .createMediaSource(streamName,

```

```
mediaSourceConfiguration)
```

### Request Camera Permissions

Starting with Android 6.0 we have to explicitly request camera permission from the user to allow its usage by the application. We do this in the home demo.

```
activity:private fun requestPermissions(){
    val permissions =
        mutableListOf<String>() if
        (ContextCompat.checkSelfPermission
            (this,
            Manifest.permission.CAMERA) !=
            PackageManager.PERMISSION_GRANTED) {
            permissions.add(Manifest.permission.CAMERA)
        } else {
            viewModel.setCameraPermission(true)
        }

        if
            (permissions.isNotEmpty()) {
                ActivityCompat.requestPermissions(this,
                    permissions.toArray(),
                    PERMISSIONS)
            }
        }

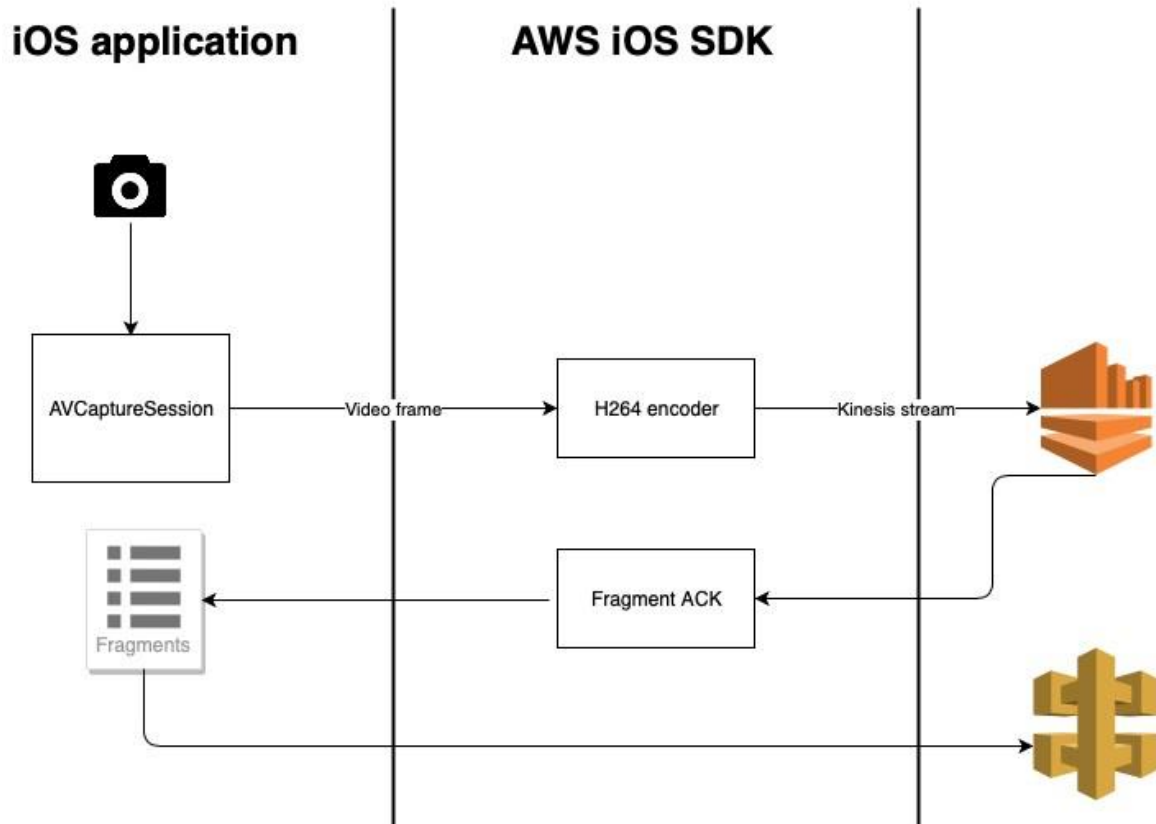
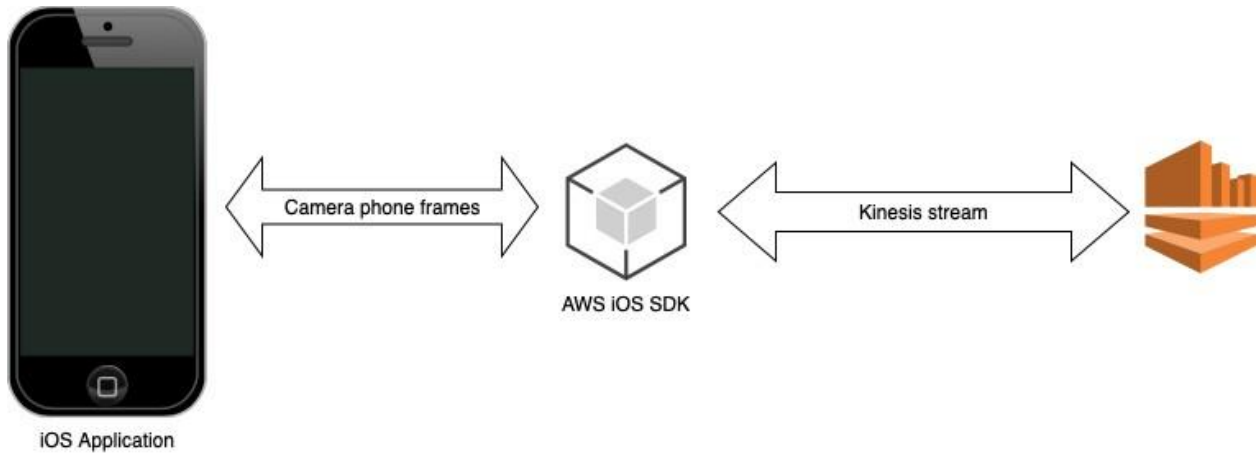
        override fun
        onRequestPermissionsResult(requestCode: Int,
        permissions: Array<String>, grantResults:
        IntArray) {
            if (requestCode ==
                PERMISSIONS) {
                for ((index,
                    permission) in
                    permissions.withIndex()) {
                    when (permission) {
                        Manifest.permission.CAMERA -> {
                            if
                                (grantResults[index] !=
                                PackageManager.PERMISSION_GRANTED) {
                                    viewModel.setCameraPermission(false)
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

**Note:** The `viewModel.setCameraPermission(true)` parameters allows ViewModel to be aware of the user's choice and to not allow KVS initialization if permission is denied.

## iOS

Source: Please contact the Mobiquity team for access to technical details.

### Architecture





## SDK Improvements

### Missing functionality

The original C++ open source code provided by AWS does not include:

1. iOS native H.264 encoder
2. Native interface (an application written in Swift or Objective-C can't call C++ functions directly)

### H.264 Encoder

Kinesis Video Stream supports only streams, which are H.264 encoded. Since it is not possible to have a multi-platform encoder, it is necessary to create a native encoder for each platform.

The new SDK project is located in the folder `AwsKvsSdk` inside the application folder. All the encode logic is defined inside the `KVSSStream.mm` file.

```
#import <Foundation/Foundation.h>
#import <AVFoundation/AVFoundation.h>

@interface KVSSStream : NSObject

- (void)encodeSampleBuffer:(CMSampleBufferRef) sampleBuffer;
- (void)stop;
```

We imported the AVFoundation framework in order to convert the frames from the camera phone. These phone camera frames are defined in the AVFoundation to have the type `CMSampleBufferRef`.

We imported another iOS native framework through the implementation of the `VideoToolbox` class, because the native functionality to encode a video to H.264 can be found in this framework.

### Encode CMSampleBufferRef object

While the encode process is long and complex, the comments in the code base will guide you, so here we only describe the main steps.

Inside the `encodeSampleBuffer` function we define the property of the H.264 compression session and then pass the frame received from the camera phone to the compress logic.

### Create Compression Session

```
size_t width = CVPixelBufferGetWidth(imageBuffer);
size_t height = CVPixelBufferGetHeight(imageBuffer);
```

```
VTCompressionSessionCreate(NULL, (int)width, (int)height,
    kCMVideoCodecType_H264, NULL, NULL, NULL, OutputCallback,
    NULL, &session);
```

### Define Session Properties

```
VTSessionSetProperty(session,
    kVTCompressionPropertyKey_RealTime, kCFBooleanTrue);
VTSessionSetProperty(session,
    kVTCompressionPropertyKey_AllowFrameReordering,
    kCFBooleanFalse);
VTSessionSetProperty(session,
    kVTCompressionPropertyKey_ProfileLevel,
    kVTProfileLevel_H264_High_AutoLevel);
VTSessionSetProperty(session,
    kVTCompressionPropertyKey_H264EntropyMode, kCFBooleanTrue);
```

### Pass Camera Frame to Session

```
VTCompressionSessionEncodeFrame(session, imageBuffer,
    presentationTimestamp, kCMTimeInvalid, NULL, NULL, NULL);
VTCompressionSessionEndPass(session, 0, NULL);
```

### **Generate Kinesis Frame**

VideoToolbox calls a call back function when a new H.264 frame is compressed. This property is passed to the VideoToolbox when you create a new compression session.

```
void OutputCallback(void *outputCallbackRefCon,
    void *sourceFrameRefCon,
    OSStatus status,
    VTEncodeInfoFlags infoFlags,
    CMSampleBufferRef sampleBuffer)
```

The complex logic inside this function analyses the H.264 frame and extracts all the information necessary to create the proper Frame, which will be sent to the Kinesis stream.

```
UINT64 timestamp = 0;
Frame frame;

timestamp =
std::chrono::duration_cast<std::chrono::nanoseconds>(
    std::chrono::system_clock::now().time_since_epoch()).count() /
DEFAULT_TIME_UNIT_IN_NANOS;
```

```

frame.decodingTs = timestamp;
frame.presentationTs = timestamp;
frame.duration = 50000;
frame.index = frameIndex;
frame.flags = (isKeyFrame) ? FRAME_FLAG_KEY_FRAME :
FRAME_FLAG_NONE;
frame.size =size;
frame.frameData = bytes;

if (nativeKvsStream->putFrame(frame) == false ) {
    LOG_ERROR("putFrame failed")
} else {
    frameIndex++;
}

```

## How to Import the AwsKvsSdk framework

At the moment, the process of building and adding this framework to a Xcode project is quite uncommon. This is due to its dependency on five external libraries, which are mandatory for the SDK. These libraries are compiled only for the ARM64 platform right now, not for x86, and they are also not conforming to bitcode.

This results in two limitations:

1. We need to use the Archive process to create a final framework, which conforms to bitcode.
2. We can run an application that uses the AwsKvsSdk only on a physical device.

## How to Import the Framework

Steps to use this library inside a new project:

1. Create a folder called Frameworks inside your project and copy the AwsKvsSdk.framework into it.
2. Open Xcode and add the folder you just created to the project.
3. Check that the AwsKvsSdk.framework is also in the Linked Frameworks, Libraries and Embedded Binaries inside the General section of your project.
4. In your target's Build phases, add a New Copy files phase.
5. In Destination select Frameworks.

6. Click on the + sign and select `AwsKvsSdk.framework`.

### How to Rebuild and Import the Framework

If you need to improve the SDK codebase, these are the steps to follow to build a new `AwsKvsSdk.framework`:

1. Select the `AwsKvsSdk` schema and a Generic iOS device
2. From Product menu run Archive
3. Copy `AwsKvsSdk.framework` from the folder

`UsersyourusernameLibraryDeveloperXcodeDerivedDataAwsKvsSdk-BuildIntermediates.noindexArchiveIntermediatesAwsKvsSdkIntermediateBuildFilesPathUniversalProductsiphoneos*` and follow the steps of the previous section.

### How to Use the SDK

Four Features of the SDK:

1. Create or re-open a Kinesis video stream
2. Get frames from the phone camera
3. Stop the stream
4. Get the information of the fragments streamed

In our demo we created a specific service class as the unique point of interaction with the SDK, `AwsKvsService.swift`. This is the interface:

```
protocol AwsKvsServiceProtocol {
    var streamCreated: Observable<Bool> { get }

    func createStream()
    func get(stream: CMSampleBuffer)
    func stopKVSStream()
```

#### Create a Stream

1. Create a `KVSCredentials` instance with your AWS access and private keys, as well as a `KVSVideoClientConfiguration` object
2. Create a `KVSVideoClient` object

### 3. Create a stream with a specific name and retention hours

```
// 1
let kvsCredentials = KVSCredentials("access_key", and:
"secret_key")
let kvsClientConfiguration =
KSVVideoClientConfiguration(kvsCredentials)
// 2
self.kvsVideoClient = KSVVideoClient(kvsClientConfiguration)
// 3
self.kvsStream = self.kvsVideoClient?.createStream(with:
self.streamName, retentionHours: 24)
```

If the stream was created beforehand, the `createStream` function re-opens the exiting one. Because these operations are synchronous, it's better to encapsulate them inside a background thread.

#### Pass Camera Frame to the Stream

After the Kinesis stream is created, the application can start to pass the frames to the SDK:

```
func get(stream: CMSampleBuffer) {
    kvsStream?.encode(stream)
}
```

The `CMSampleBuffer` comes from the application service `AVFoundationService.swift` every time `AVCaptureVideoDataOutputSampleBufferDelegate` is called:

```
extension AVFoundationService:
AVCaptureVideoDataOutputSampleBufferDelegate {
    func captureOutput(_ output: AVCaptureOutput,
                      didOutput sampleBuffer: CMSampleBuffer,
                      from connection: AVCaptureConnection) {
        onSampleComplete(sampleBuffer)
    }
}
```

#### Stop Streaming

When the user ends the stream, or the application enters into the background, the application must stop the Kinesis stream.

1. Call the stop stream SDK function
2. Call the free buffer SDK function

```
func stopKVSStream() {
    // 1
    kvsStream?.stop()
    // 2
    kvsVideoClient?.freeStream()

    ...
}
```

The call to `freeStream()` is important since it prevents the SDK from sending all the frames from its buffer after the stop stream function is called. This would cause the application UI to slow down or become blocked. The drawback is that the frames in the SDK buffer are lost.

### Fragments Callback

The application needs to store the sent fragments in order to understand where the last user record inside the Kinesis stream is located.

Fragment information is shared from the SDK through a callback broadcast using the `NotificationCenter`. The name of the notification is `fragmentAckReceived`

```
static let fragmentAck =
    NotificationCenter.Name("fragmentAckReceived")
```

Every time this notification is sent, the application needs to store the information inside it. Everything is managed by `AwsKvsService.swift`:

```

notificationCenter.addObserver(self,
                                selector:
#selector(self.fragmentAck(notification:)),
                                name: .fragmentAck,
                                object: nil)

@objc func fragmentAck(notification: Notification) {
    guard let userInfo = notification.userInfo,
          let number = userInfo[KVSAAppConstants.fragmentNumber]
as? String,
          let timestampMillisec =
userInfo[KVSAAppConstants.fragmentTimestamp] as? Int else {
        return
    }

    fragments.append(Fragment(number: number, timestamp:
timestampMillisec))
}

struct Fragment: Codable {
    let number: String
    let timestamp: Int
}

```

## Phone Camera Stream

All the logic to retrieve the frames from the phone camera is inside the service `AVFoundationService.swift` and is based on the native standard framework `AVCaptureSession`

Main function:

```
func setupService(with view: UIView,
                 configuration: DeviceConfiguration?,
                 onSampleComplete: @escaping
SampleBufferComplete) throws
```

- View is the view where we want to show the images from the phone camera
- Configuration is a simple structure used to define the camera frames per second (fps)
- onSampleComplete is the callback, which is called whenever a frame is ready

Setup of the camera:

1. Define the type of output to be used with the AVCaptureSession. In our case, we use AVCaptureVideoDataOutput, because we need a video format output.
2. Create the AVCaptureVideoPreviewLayer to show the camera output to the user.
3. Define the input we want to use with the AVCaptureSession.



```

func setupService(with view: UIView,
                  configuration: DeviceConfiguration? = nil,
                  onSampleComplete: @escaping
SampleBufferComplete) throws {

    self.onSampleComplete = onSampleComplete

    notificationCenter.addObserver(self,
                                    selector:
#selector(self.applicationDidEnterBackground),
                                    name:
.applicationDidEnterBackground,
                                    object: nil)

    if let conf = configuration {
        self.configuration = conf
    }

    // 1
    videoDataOutput = createCaptureVideoDataOutput()
    /* To receive samples in their device native format, set
this property to nil */
    videoDataOutput?.videoSettings = nil
    // 2
    previewLayer = createPreviewLayer(to: view)
    // 3
    try setupDevice()
}

```