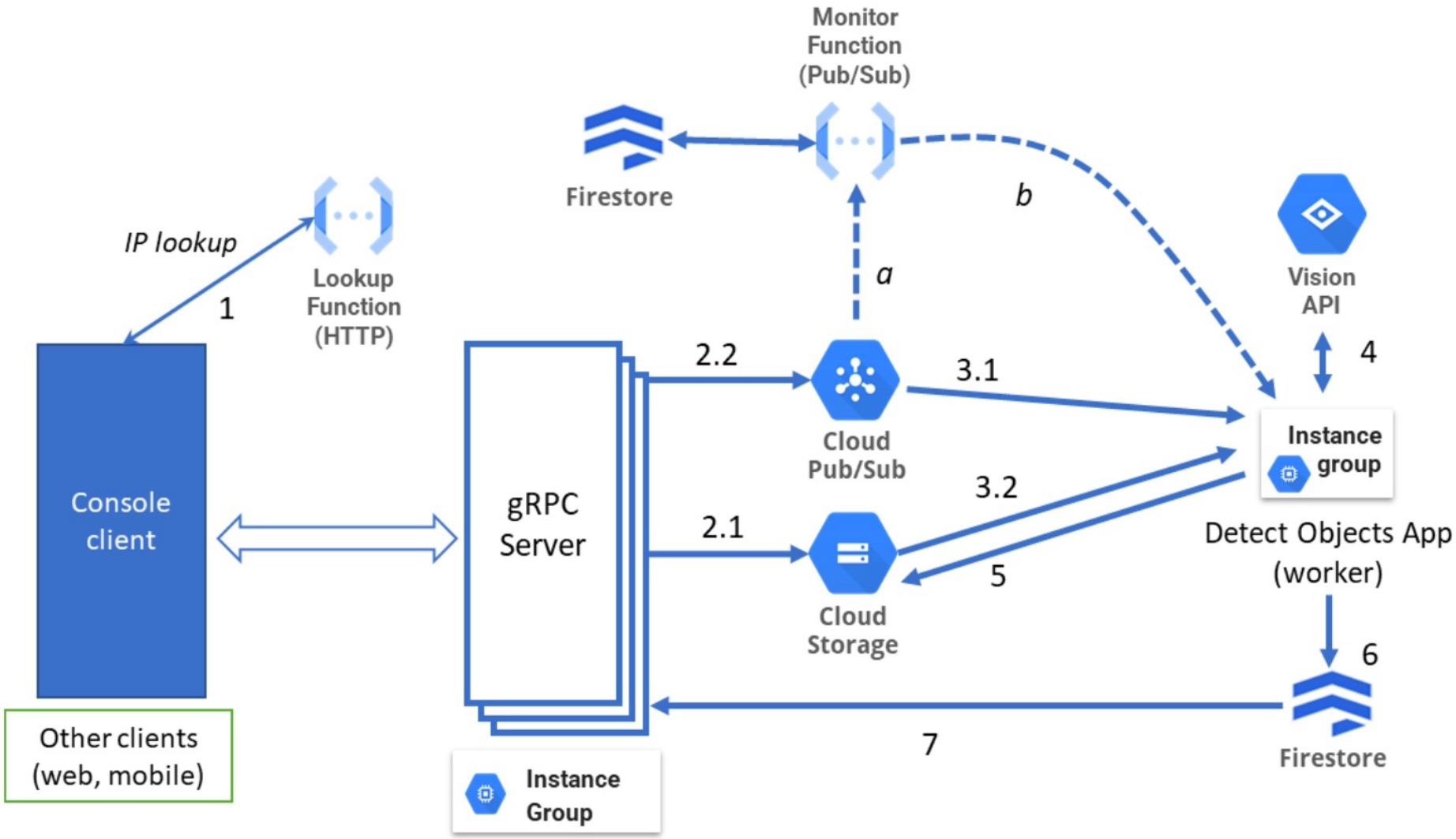
Diagrama


Monitor
Function
(Pub/Sub)
Firestore
b
a
IP lookup
1
Lookup
Function
(HTTP)
Vision
API
4
2.2
3.1
Cloud
Pub/Sub
Instance
group
Console
client
gRPC
Server
3.2
2.1
Detect Objects App
(worker)
5
Cloud
Storage
6
Other clients
(web, mobile)
7
Firestore
Instance
Group

Contrato

```
service GrpcserverCN {
  rpc submitFileForObjectDetection(stream image) returns (imageId);
  rpc getObjectsList(imageId) returns (objectsList);
  rpc getOriginalImageAnnotated(imageId) returns (stream image);
  rpc getImageNameWithQuery(filter) returns (fileList);
}
```

```
message filter {
  string startDate = 1;
  string endDate = 2;
  string objectName = 3;
  double t = 4;
}
message fileList {
  repeated string fileName = 1;
}

message image {
  string name = 1;
  bytes image_data = 2;
}
message imageId {
  string id = 1;
}
message objectsList {
  repeated string object = 1;
}
```

Cloud Function: HTTP trigger

```java
public class LookupFunction implements HttpFunction {
    RainPumpkin
    @Override
    public void service(HttpRequest httpRequest, HttpResponse httpResponse) throws Exception {
        String project = "cn2122-t2-g11";
        String zone = "europe-west1-b";
        //Get query parameter instanceGroupName
        String instanceGroupName;
        Map<String, List<String>> params = httpRequest.getQueryParameters();
        if(params.containsKey("instanceGroupName")){
            instanceGroupName = params.get("instanceGroupName").get(0);
        } else {
            httpResponse.setStatusCode( i: 400,  s: "Missing parameter instanceGroupName");
            return;
        }

        //Get list of managed instances
        InstanceGroupManagersClient managersClient = InstanceGroupManagersClient.create();
        ListManagedInstancesInstanceGroupManagersRequest request =
                ListManagedInstancesInstanceGroupManagersRequest.newBuilder()
                        .setInstanceGroupManager(instanceGroupName)
                        .setProject(project)
                        .setReturnPartialSuccess(true)
                        .setZone(zone)
                        .build();
        System.out.println("Instances of instance group: " + instanceGroupName);
        List<String> instances = new LinkedList<>();
        for (ManagedInstance instance : managersClient.listManagedInstances(request).iterateAll()) {
            String[] split = instance.getInstance().split( regex: "/");
            instances.add(split[split.length-1]);
        }
```

Cloud Function: HTTP trigger - Continuação

```
        Random rm = new Random();
        int n = rm.nextInt(instances.size());

        try (InstancesClient client = InstancesClient.create()) {
            for (Instance instance : client.list(project, zone).iterateAll()) {
                if(instances.contains(instance.getName())){
                    if(n == 0) {
                        String ip = instance.getNetworkInterfaces( index: 0).getAccessConfigs( index: 0).getNatIP();
                        httpResponse.getWriter().write(ip);
                        httpResponse.setContentType("text/plain");
                        return;
                    }
                    else n--;
                }
            }
        } catch (Exception e) {
            httpResponse.getWriter().write(e.getMessage());
            httpResponse.setStatusCode( i: 501,  s: "exception");
        }
        httpResponse.getWriter().write( str: "zone: " + zone + " | " + instanceGroupName + " | "
                + instances.size() + " |" + instances.get(n));
        httpResponse.setStatusCode( i: 500,  s: "unexpected result of function");
    }
}
```

Cliente: usage of CloudFunction ->IPlookup

```
private static void ipLookup() throws IOException {
    URL url = new URL(IP_LOOKUP_URL);
    HttpURLConnection con = (HttpURLConnection) url.openConnection();
    con.setRequestMethod("GET");
    con.setDoOutput(true);
    DataOutputStream out = new DataOutputStream(con.getOutputStream());
    out.writeBytes( s: "instanceGroupName=" + INSTANCE_NAME);
    out.flush();
    out.close();
    int status = con.getResponseCode();
    if(status != 200){
        //get the error
        svcIP=null;
        System.out.println(con.getErrorStream());
        System.out.println("Ip lookup error, shutting down");
        System.exit( status: -1);
    } else {
        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        String toRet=in.readLine();
        con.disconnect();
        svcIP = toRet;
    }
}
```

Cliente: Setup

```
private static void setup() throws IOException {
    System.out.println("Starting setup");
    ipLookup();
    System.out.println("ServerIP=" + svcIP);

    channel = ManagedChannelBuilder.forAddress(svcIP,svcPort) ManagedChannelBuilder<capture of ?>
            .usePlaintext() capture of ?
            .build();
    blockingStub = newBlockingStub(channel);
    noBlockStub = newStub(channel);
    scan = new Scanner(System.in);
    System.out.println("Setup finish");
}
```

Cliente: chamar a função do contrato (case 3)

```
private static void submitFileForObjectDetection() throws IOException, InterruptedException {
    String path = scan.nextLine();
    if (path.length()<2) path = defaultImagePath;
    Path filePath = Paths.get(path);
    try (InputStream input = Files.newInputStream(filePath)) {
        byte[] buffer = new byte[1024];
        int limit;
        ImageIdObserver obs = new ImageIdObserver();
        StreamObserver<image> stream = noBlockStub.submitFileForObjectDetection(obs);
        while ((limit = input.read(buffer)) >= 0) {
            if (limit < 1024){
                System.arraycopy(buffer, srcPos: 0, buffer, destPos: 0, limit);
            }
            image img = image.newBuilder().setImageData(ByteString.copyFrom(buffer))
                    .setName(filePath.getFileName().toString()).build();
            stream.onNext(img);
        }
        stream.onCompleted();
        while (!obs.isCompleted()){
            Thread.sleep( millis: 2000);
        }
        if (obs.isSuccess()){
            lastId=obs.getId();
            lastUploadPath=path;
        }
    }
}
```

ImageIdObserver usado no slide anterior

```java
public class ImageIdObserver implements StreamObserver<imageId> {
    private imageId res;
    private boolean isCompleted = false;
    private boolean isSuccess = false;
    public boolean isSuccess() { return isSuccess; }
    public boolean isCompleted() { return isCompleted; }
    public String getId() { return res.getId(); }

    @Override
    public void onNext(imageId imageID) { res = imageID; }
    @Override
    public void onError(Throwable throwable) {
        System.out.println("Error on upload: " + throwable.getMessage());
        isCompleted = true;
    }
    @Override
    public void onCompleted() {
        System.out.println("Id: " + res.getId());
        isSuccess = true;
        isCompleted = true;
    }
}
```

Cliente: chamar a função do contrato (case 1)

```
private static void getObjectsList(){
    String idString = scan.nextLine();
    if (idString.length()<2) idString = lastId;
    imageId id = imageId.newBuilder().setId(idString).build();
    List<String> objectsList = blockingStub.getObjectsList(id).getObjectList();
    System.out.println("Objects:");
    for (String s : objectsList) {
        System.out.println("-" + s);
    }
}
```

Cliente: chamar a função do contrato (case 2)

```
private static void getAnnotatedImage() throws IOException {
    System.out.println("Write the original image path: (Enter for last uploaded)");
    String pathName = scan.nextLine();
    if (pathName.length()<2) pathName = lastUploadPath;
    String[] split = pathName.split( regex: "\\.");
    StringBuilder newPath = new StringBuilder(".").append(split[0]);
    for (int i = 1; i <= split.length-2; i++) {
        newPath.append(split[i]);
    }
    newPath.append("-annotated.").append(split[split.length-1]);
    pathName = newPath.toString();

    System.out.println("Write the imageId provided after upload: (Enter for last upload imageID)");
    String idString = scan.nextLine();
    if (idString.length()<2) idString=lastId;
    imageId id = imageId.newBuilder().setId(idString).build();
    Iterator<image> images = blockingStub.getOriginalImageAnnotated(id);
    ByteArrayOutputStream bytesOut = new ByteArrayOutputStream();
    while (images.hasNext()){
        bytesOut.write((images.next()).getImageData().toByteArray());
    }
    Path path = Paths.get(pathName);
    Files.write(path, bytesOut.toByteArray());
    System.out.println("Annotated image at: " + path.toString());
}
```

Cliente: chamar a função do contrato (case 1)

```
private static void getImageNameWithQuery(){
    System.out.println("Write start date: (format:dd/mm/yyyy) (enter for default 1/06/2022)");
    String sDate = scan.nextLine();
    if (sDate.length()<2) sDate = DEFAULT_START_DATE;

    System.out.println("Write end date: (format:dd/mm/yyyy) (enter for default 10/07/2022)");
    String eDate = scan.nextLine();
    if (eDate.length()<2) eDate = DEFAULT_SEND_DATE;

    System.out.println("Write object name: (enter for default \"wheel\")");
    String obj = scan.nextLine();
    if (obj.length()<2) obj = DEFAULT_OBJECT;

    System.out.println("Write confidence: (write anything above 1 for default 0,5)");
    double confidence = scan.nextDouble();
    if(confidence>1 || confidence<0) confidence = DEFAULT_CONFIDENCE;
    scan.nextLine();

    filter request = filter.newBuilder().setStartDate(sDate).setEndDate(eDate)
            .setObjectName(obj).setT(confidence).build();
    List<String> res = blockingStub.getImageNameWithQuery(request).getFileNameList();
    if (res.size()==0) System.out.println("No files found.");
    else {
        System.out.println("Files:");
        for (String re : res) {
            System.out.println("-" + re);
        }
    }
}
```

Server Implementation

```
public class GRPCServer extends GrpcserverCNGrpc.GrpcserverCNImplBase {
    private static int svcPort = 7001;
    static Storage storage = null;
    static Firestore db = null;
    static String projID;
    private static String bucketName = "cn2122t2g11images";
    private static String currentCollection = "Annotations";


    public static void main(String[] args) {...}

    private static void storageSetup() {...}

    public static void initFirestore() throws IOException {...}

    public StreamObserver<image> submitFileForObjectDetection(StreamObserver<imageId> responseObserver) {...}

    @Override
    public void getObjectsList(imageId request, StreamObserver<objectsList> responseObserver) {...}

    @Override
    public void getOriginalImageAnnotated(imageId request, StreamObserver<image> responseObserver) {...}

    @Override
    public void getImageNameWithQuery(filter request, StreamObserver<fileList> responseObserver) {...}
}
```

```java
public static void main(String[] args) {
    try {
        storageSetup();
        initFirestore();
        io.grpc.Server svc = ServerBuilder.forPort(svcPort).addService(new GRPCServer()).build();
        svc.start();
        svc.awaitTermination();
        svc.shutdown();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

```java
private static void storageSetup() {
    StorageOptions storageOptions = StorageOptions.getDefaultInstance();
    storage = storageOptions.getService();
    projID = storageOptions.getProjectId();
    if (projID != null) System.out.println("Current Project ID:" + projID);
    else {
        System.out.println("The environment variable GOOGLE_APPLICATION_CREDENTIALS isn't well defined!!");
        System.exit( status: -1);
    }
}
```

```java
public static void initFirestore() throws IOException {
    // use GOOGLE_APPLICATION_CREDENTIALS environment variable
    GoogleCredentials credentials = GoogleCredentials.getApplicationDefault();
    FirestoreOptions options = FirestoreOptions.newBuilder().setCredentials(credentials).build();
    db = options.getService();
}
```

```java
@Override
public StreamObserver<image> submitFileForObjectDetection(StreamObserver<imageId> responseObserver) {
    return new ImageObserver(storage, responseObserver, bucketName);
}

@Override
public void getObjectsList(imageId request, StreamObserver<objectsList> responseObserver) {
    String[] split = request.getId().split( regex: "\\|");
    String imageId = split[1];
    DocumentReference docRef = db.collection(currentCollection).document(imageId);

    LinkedList<String> list = new LinkedList<>();
    try {
        ApiFuture<DocumentSnapshot> future = docRef.get();
        DocumentSnapshot doc = future.get();

        AnnotationDetail annotationDetail = doc.toObject(AnnotationDetail.class);
        var annotationsArray : ArrayList<Annotation> = annotationDetail.getAnnotation();
        for (Annotation annotation : annotationsArray) {
            if (!list.contains(annotation.annotatedName)) {
                list.add(annotation.annotatedName);
            }
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
    var protoList : objectsList = objectsList.newBuilder().addAllObject(list).build();

    responseObserver.onNext(protoList);
    responseObserver.onCompleted();
}
```

```java
@Override
public void getOriginalImageAnnotated(imageId request, StreamObserver<image> responseObserver) {
    String[] split = request.getId().split( regex: "\\|");
    BlobId blobId = BlobId.of(split[0],  name: "annotated-" + split[1]);
    Blob blob = storage.get(blobId);
    // When Blob size is big or unknown use the blob's channel reader.
    try (ReadChannel reader = blob.reader()) {
        ByteBuffer bytes = ByteBuffer.allocate( capacity: 64 * 1024);
        while (reader.read(bytes) > 0) {
            bytes.flip();
            responseObserver.onNext(image.newBuilder().setImageData(ByteString.copyFrom(bytes)).build());
            bytes.clear();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    responseObserver.onCompleted();
}
```

```java
@Override
public void getImageNameWithQuery(filter request, StreamObserver<fileList> responseObserver) {
    try {
        SimpleDateFormat dtf = new SimpleDateFormat( pattern: "dd/MM/yyyy");
        Date startDate = dtf.parse(request.getStartDate());
        Date endDate = dtf.parse(request.getEndDate());
        String objectName = request.getObjectName();
        double confidence = request.getT();

        CollectionReference cRef = db.collection(currentCollection);
        Iterable<DocumentReference> allDocs = cRef.listDocuments();
        fileList.Builder toRet = fileList.newBuilder();
        for (DocumentReference docRef : allDocs) {

            ApiFuture<DocumentSnapshot> docFuture = docRef.get();
            DocumentSnapshot doc = docFuture.get();

            AnnotationDetail annotationDetail = doc.toObject(AnnotationDetail.class);
            //check for date
            Date docDate = dtf.parse(annotationDetail.date);
            if (startDate.before(docDate) && endDate.after(docDate)) {
                //docDate is  between date parameters
                //search fields for object and confidence
                for (Annotation a : annotationDetail.getAnnotation()) {
                    if (a.annotatedName.equals(objectName) && a.confidence >= confidence) {
                        toRet.addFileName(docRef.getId());
                        break;
                    }
                }
            }
        }
        responseObserver.onNext(toRet.build());
        responseObserver.onCompleted();
    } catch (ParseException | ExecutionException | InterruptedException e) {...}
```

```java
public class ImageObserver implements StreamObserver<image> {
    private final Storage storage;
    private final ByteArrayOutputStream bytesOut;
    private String blobName = null;
    private String bucketName;
    private final String PROJECT = "cn2122-t2-g11";
    private final String TOPIC = "CN2122TFImage";
    private final StreamObserver<imageId> responseObserver;
    ImageObserver(Storage storage, StreamObserver<imageId> responseObserver, String bucketName) {
        this.storage = storage;
        this.responseObserver = responseObserver;
        this.bucketName = bucketName;
        bytesOut = new ByteArrayOutputStream();
    }
    @Override
    public void onNext(image image) {...}
    @Override
    public void onError(Throwable throwable) {
        throwable.printStackTrace();
    }
    @Override
    public void onCompleted() {...}
}
```

```java
@Override
public void onError(Throwable throwable) {
    throwable.printStackTrace();
}
```

```java
@Override
public void onNext(image image) {
    if (blobName == null) blobName = image.getName();
    //use better structure later
    byte[] bytes = image.getImageData().toByteArray();
    try {
        bytesOut.write(bytes);
    } catch (IOException e) {
        e.printStackTrace();
    }
    //for (byte aByte : bytes) imageBytes.add(aByte);
}
```

```java
@Override
public void onCompleted() {
    byte[] bytes = bytesOut.toByteArray();
    BlobId blobId = BlobId.of(bucketName, blobName);
    BlobInfo blobInfo = BlobInfo.newBuilder(blobId).setContentType("image").build();
    try (WriteChannel writer = storage.writer(blobInfo)) {
        byte[] buffer = new byte[1024];
        try (InputStream input = new ByteArrayInputStream(bytes)) {
            int limit;
            while ((limit = input.read(buffer)) >= 0) {
                try {
                    writer.write(ByteBuffer.wrap(buffer,  offset: 0, limit));
                } catch (Exception ex) {...}
            }
        }
    } catch (IOException e) {...}
    String res = bucketName + "|" + blobId.getName();
    responseObserver.onNext(imageId.newBuilder().setId(res).build());
    responseObserver.onCompleted();
    TopicName topic = TopicName.ofProjectTopicName(PROJECT, TOPIC);
    Publisher publisher;
    try {
        publisher = Publisher.newBuilder(topic).build();
    } catch (IOException e) {...}
    ByteString msgData = ByteString.copyFromUtf8(res);
    PubsubMessage pubsubMessage = PubsubMessage.newBuilder()
            .setData(msgData)
            .build();
    publisher.publish(pubsubMessage);
    publisher.shutdown();
}
```

Cloud Function: Pub/Sub Trigger

```java
public class MonitorFunction implements BackgroundFunction<MonitorFunction.PubSubMessage> {
    private static final Logger logger = Logger.getLogger(MonitorFunction.class.getName());
    private static final String COLLECTION_NAME = "DetectObjectsMonitor";
    private static final String INSTANCE_GROUP = "detect-object-group";
    private static final String ZONE = "europe-west1-b";
    private static final String PROJECT = "cn2122-t2-g11";
    private static Firestore db = initFirestore();
    public static class PubSubMessage {

        String data;
        Map<String, String> attributes;
        String messageId;
        String publishTime;
    }
    private static Firestore initFirestore(){...}
    @Override
    public void accept(PubSubMessage message, Context context) throws Exception {...}
    private static void postRequestFirestore(String message, String eventID){...}
    private static double getRequestsLastMinute() throws ExecutionException, InterruptedException {...}
    private static void resizeInstances(int newNumber) throws IOException {...}
    private static int getCurrentVMCount() throws IOException {...}
    private static void decisionMaker(double reqs, int currVMs) throws IOException {...}
```

```java
@Override
public void accept(PubSubMessage message, Context context) throws Exception {
    String msg = new String(Base64.getDecoder().decode(message.data));
    postRequestFirestore(msg, context.eventId());
    double reqs = getRequestsLastMinute();
    int VMs = getCurrentVMCount();
    decisionMaker( reqs: reqs/60, VMs);
}
private static void postRequestFirestore(String message, String eventID){
    Date curr = new Date(System.currentTimeMillis());
    DocumentReference docRef = db.collection(COLLECTION_NAME).document();
    HashMap<String, Object> map = new HashMap<>() {
        {
            put("message", message);
            put("eventID", eventID);
            put("date", Timestamp.of(curr));
        }
    };
    docRef.create(map);
}
```

```
private static double getRequestsLastMinute() throws ExecutionException, InterruptedException {
    Query query = db.collection(COLLECTION_NAME) CollectionReference
            .whereGreaterThanOrEqualTo( field: "date",
                    Timestamp.of(new Date(System.currentTimeMillis()-60000))) Query
            .whereLessThanOrEqualTo( field: "date",
                    Timestamp.of(new Date(System.currentTimeMillis())));
    ApiFuture<QuerySnapshot> querySnapshot = query.get();
    int toRet = querySnapshot.get().getDocuments().size();
    logger.info( msg: "left getRequests with value = " + toRet);
    return toRet;
}
private static void resizeInstances(int newNumber) throws IOException {
    InstanceGroupManagersClient managersClient = InstanceGroupManagersClient.create();
    managersClient.resizeAsync(PROJECT, ZONE, INSTANCE_GROUP, newNumber);
}
```

```java
public class Logger {
    private static final String PROJECT_ID = "cn2122-t2-g11";
    private static final String SUBSCRIPTION_NAME = "annotationSub";

    public static void main(String[] args) {
        try {
            DetectedObject.initFirestore( pathFileKeyJson: null);
            Subscriber subs = subscribeMessages();
            while (subs.isRunning());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Subscriber subscribeMessages() {
        ProjectSubscriptionName projSubscriptionName = ProjectSubscriptionName.of(
                PROJECT_ID, SUBSCRIPTION_NAME);
        Subscriber subscriber =
                Subscriber.newBuilder(projSubscriptionName, new MessageReceiverHandler())
                        .build();
        subscriber.startAsync().awaitRunning();
        return subscriber;
    }
}
```

```java
public class MessageReceiverHandler implements MessageReceiver {
    @Override
    public void receiveMessage(PubsubMessage pubsubMessage, AckReplyConsumer ackReplyConsumer) {
        try {
            String str = pubsubMessage.getData().toStringUtf8();
            System.out.println("Data:" + str);

            String[] dataArray = str.split( regex: "\\|");
            if (dataArray.length == 2) {
                String bucketName = dataArray[0];
                String blobName = dataArray[1];
                System.out.println("bucketName:" + bucketName + " blobName" + blobName);

                DetectedObject.getAnnotatedImage(bucketName, blobName);//TODO:

                ackReplyConsumer.ack();
            } else {
                throw new Exception("invalid pubsub message");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```