Cameras Ticketing System

Daniel Sabba, Stav Yosef

Introduction:

In this designing system, we need to focus on a couple of things:

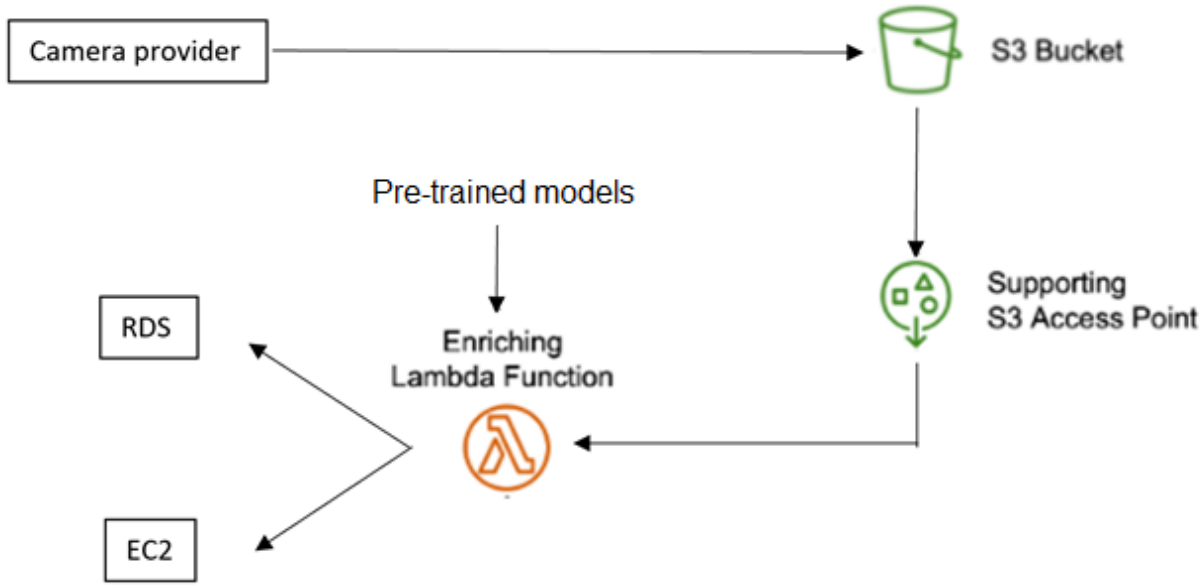
1. System reliability.
2. Capable of consuming big data.
3. Keep the costs low as possible.
4. Store backup.

Before we dive into the system, we need to understand why we have these requirements.
Let's jump right into the production stage, we have a lot of cameras and each camera produce a lot of pictures therefore we need to be able to handle big data. This is a ticketing system, so we need the system to be reliable and to save the data efficiently (low costs) because we need to keep proof of the speed violations.
To fulfill all these requirements, we'll use a couple of different components such as S3, EC2, Lambda, and RDS.

System Design

The following sketch will show us how the data flows in our system, and help us understand how the system can scale, be modified, and expand.

Note - in this sketch, we can see only one camera provider, but of course, there are many.



We know traffic on the roads is not constant and, there is time in the day the traffic is high, morning and evening, while in the middle of the night the traffic is probably low therefore Lambda function is perfect for this task.

We will use RDS to store pre-configured data and as backup, tables described below.

The first table is used to keep tracking all of the data that enter our system.

Camera ID	Timestamp	Manufacturer	Color	License plate

The second table is used to setup all the alerts' configurations

Alert ID	License plate	Manufacturer	Color	Location	Region	Distance	Ending timestamp

The third table is used to setup all the camera pairs configurations (speeding violations)

Camera A ID	Camera B ID	Distance	Speed Limit	Violation Fee

The last table is alerting for ticketing, speeding violations.

Camera A ID	Camera B ID	License plate	Threshold timestamp

S3 structure:

To be able to delete easily old pictures we decide the following saving format, {main_dir}/{picture_year}/{picture_month}/{picture_day}/{camera_id}_{timestamp}.{picture_ext (.jpg/.png)}

This structure will help us to delete easily picture that past certain date (3 years from the date the picture was taken).

We will use EC2 instance in order to analyze in real-time alerts.

EC2 memory structure:

1. Alerts' configurations can be configured via https requests.
2. Camera pairs configurations can be also configured via https requests.
3. List for each camera that contains a similar structure as the last table.

System Flow

Each camera uploads a zip file to S3 containing the actual image, and a JSON file containing the camera ID and timestamp the image was created.
The zip file will be routed into a Lambda function that will do a couple of things:

1. Extract the zip file.
2. Renaming the image to "{camera_id}_{timestamp}" and moving it to the right path.
3. Extract using pre-trained models the following feature:
    a. Car manufacturer.
    b. Car color.
    c. License plate.
4. Putting the data into RDS table (the first table) and send to our EC2 the same data in the same structure.
5. Removing the zip and JSON files.

The EC2 instance has a big role in our system. Let's simulate how new data is being procced to understand the flow. When new data arrives, the EC2 will go over the alerts configurations and see if there is a match (can be multiple matches). If there is a match the instance will fire a new email or SMS. SMS can be sent via Twilio service. (prices are in U.S)

NUMBER USED	TEXT MESSAGES		PICTURE MESSAGES		MESSAGING SERVICE FEATURES*
	TO SEND †‡§	TO RECEIVE †§	TO SEND †‡§	TO RECEIVE †§	
LOCAL NUMBERS	$ 0.0075	$ 0.0075	$ 0.0200	$ 0.0100	INCLUDED
TOLL-FREE NUMBERS	$ 0.0075	$ 0.0075	$ 0.0200	$ 0.0200	INCLUDED
SHORT CODES	$ 0.0075	$ 0.0075	$ 0.0200	$ 0.0100	INCLUDED

Next, we need to check in the camera pair configurations if there is a match between Camera ID to Camera A ID, if there is a match then add it to the list of camera ID a new alert to be checked when the car will pass in Camera B ID. The structure of the alerts is as in the last table.
Camera A ID - current data Camera ID
Camera B ID - Camera B ID (from the configuration)
License plate - current data License plate
Threshold timestamp - If in the future data, when the car will pass Camera B ID the timestamp will be smaller than the threshold then sends new speed violation ticket.

Overall, the EC2 instance can handle this kind of load because we know we have a maximum of 5000 cameras therefore our instance can handle all the lists well. We decided to use this approach because it's a waste to query the database for every little thing, this will cause high costs and will take more time significantly.

Pricing

RDS pricing

In the first year, the trial run, 150 cameras that produce 12,500 pictures per camera results total of 1,875,000 pictures per day. Each picture will generate 18 bytes of data, we can see our estimation as the following table

Camera ID	Timestamp	Manufacturer	Color	License plate
2 bytes	4 bytes	1 byte	1 byte	10 bytes

Camera ID - we have 5000 cameras → 2 bytes range is 0 to 65,535 (unsigned values).
Timestamp - 4 bytes range is 0 to 4,294,967,295 and currently Unix time is 1,627,818,460.
Manufacturer - we can assume there is no more than 255 (1 bytes) manufacturers in the system. Note - If in the future we will be needed to also save the model then extend this field to 2 bytes.
Color - again, we can assume there is no more than 255 (1 byte) car colors.
License plate - this is a little bit tricky, there are many kinds of license plates around the world, so we chose to declare this field with 10 bytes (there are license plates with characters).

Total we have - $2_{bytes} + 4_{bytes} + 1_{bytes} + 1_{bytes} + 10_{bytes} = 18_{bytes}$

$$1{,}875{,}000_{pictures} \cdot 18_{bytes} = \frac{33{,}750{,}000_{bytes}}{1024} = \frac{32{,}958.98_{KB}}{1024} = \frac{32.18_{MB}}{1024} = 0.0314_{GB}$$

In the first year our system produces 0.0314_{GB} per day, total in the first year 11.47_{GB}
We need to configure one RDS for MySQL, the instance we chose for the first year is:
db.m4.xlarge with the specification: vCPU: 4, Memory: 16 GiB.

Pricing options:

1. OnDemand - $511\$_{month} \cdot 12$ = **6,132$** in one year.
2. Reserved for one year up front - **3,531$**

There is huge difference between on demand and upfront, we can afford to pay upfront we'll chose this option.

In the second stage, from the second year we have $\frac{2500+500}{2} = 3750$ cameras that each of them produces 17,500 pictures. $3750 \cdot 17{,}500 = 65{,}625{,}000$ pictures per day.

$$65{,}625{,}000_{pictures} \cdot 18_{bytes} = \frac{1{,}181{,}250{,}000_{bytes}}{1024} = \frac{1{,}153{,}564.45_{KB}}{1024} = \frac{1{,}126.52_{MB}}{1024} = 1.1_{GB}$$

From the second year our system produces 1.1_{GB} per day, in total 401.5_{GB} per year.
We need to upgrade the instance to be able handle much more requests, so upgrade to

db.m4.2xlarge with the specification: vCPU: 8, Memory: 32 GiB. Again, there is option to pay on demand and upfront, 7063$ for 1 year. We also need to extend our storage. On each month we produce $1.1_{GB} \cdot 30 = 33_{GB}$ and the cost of this additional storage is 7.59$.

$$\sum_{m=1}^{48\ (48\ months\ in\ 4\ years)} \left(7.59 \cdot (48 - m + 1)\right) = \mathbf{8925.84}\$$$

For backup storage, the pricing for 33_{GB} is 3.13$:

$$\sum_{m=1}^{48\ (48\ months\ in\ 4\ years)} \left(3.13 \cdot (48 - m + 1)\right) = \mathbf{3680.88}\$$$

For the 3,4,5 years we will upgrade again our RDS instance to db.m4.2xlarge with the specification: vCPU: 16, Memory: 64 GiB. This time we can pay upfront for 3 years! 27,470$

Total costs for 5 years - $3531 + 7063 + 27{,}470 + 8925.84 + 3680.88 = \mathbf{50{,}670.72}\$$

Notes:

1. we ignored the memory that we received from the upgraded instances in the calculations, there is no big difference and it's made the calculations and the explanations of the process harder.
2. In our system design the EC2 will also use the RDS component but we can assume that its cost is neglect according the rest of the cost.

Lambda pricing

To be able to calculate the lambda function cost, we will have to make some assumptions about the models the function uses. let's assume each model is a pre-trained VGG16 neural network model, each model weighs 528_{MB}, we have 3 of those, detecting the manufacturer, color, and the license plate.

Each model takes approximately 648_{ms} on CPU, means the function will need 2_{GB} allocated memory, with 2000_{ms} per request.

In the first year, $57{,}031{,}250$ invocations per month which cost $3{,}806.63\$$

Total cost for the first year - $3{,}806.63\$_{month} \cdot 12 = 45{,}679.56\$$

In the second stage, $1{,}996{,}093{,}750$ invocations per month which cost $133{,}465.54\$$

Total cost for the second stage per year - $133{,}465.54\$_{month} \cdot 12 = 1{,}601{,}586.48\$$

Total cost for 5 year plan - $133{,}465.54\$ + 1{,}601{,}586.48\$ \cdot 4_{\text{years}} = \mathbf{6{,}539{,}811.46}\$$

S3 pricing

We estimate that the size of each picture is 100_{KB} in order to be able make pricing estimation.
For the trial run the system produces 1,875,000 per day and for each month 57,031,250 pictures with the total size of $57{,}031{,}250 \cdot 100_{KB} = \frac{5{,}703{,}125{,}000_{KB}}{1024} = \frac{5{,}569{,}458_{MB}}{1024} = \frac{5{,}438.9_{GB}}{1024} = 5.31_{TB}$
We know that we need to save the pictures for 3 years which is 36 months then,
S3 Standard cost (monthly): $125.06\$$
Total cost of the first year data - $125.06\$ \cdot 12_{months} \cdot 36_{months\ to\ save} = \mathbf{54{,}025.92}\$$
If we'll use S3 Intelligent-Tiering, we can reduce our cost to 15.31$ monthly.
The setting used for S3 INT is 5%, 5%, 5%, 85%.
Total cost of the first year data - $15.31\$ \cdot 12_{months} \cdot 36_{months\ to\ save} = \mathbf{6{,}613.92}\$$

For the second stage we need to divide it 2 parts, in the first part is the data of the second year and the third year. The second part is the data of the fourth year and the fifth year. We need to divide this because in the fourth and fifth year we don't calculate for 3 years ahead.

For the second stage each month we produce 1,996,093,750 pictures.

$$1{,}996{,}093{,}750 \cdot 100_{KB} = \frac{199{,}609{,}375{,}000_{KB}}{1024} = \frac{194{,}931{,}030.27_{MB}}{1024} = \frac{190{,}362.33_{GB}}{1024} = 185.9_{TB}$$

S3 Standard cost (monthly): $4{,}239.16\$$
S3 Intelligent-Tiering (same setting as before) cost (monthly): $538.15\$$

For the second and third years we need to save the data for 3 full years (S3 Standard),
$2 \cdot (4{,}239.16\$ \cdot 12 \cdot 36) = \mathbf{3{,}662{,}634.24}\$$

And for S3 Intelligent-Tiering,
$2 \cdot (538.15\$ \cdot 12 \cdot 36) = \mathbf{464{,}961.6}\$$

For the fourth and fifth year we can come up with the following equation (S3 Standard)

$$\sum_{m=1}^{24} \left(4{,}239.16 \cdot (24 - m + 1)\right) = \mathbf{1{,}271{,}748}\$$$

And for S3 Intelligent-Tiering

$$\sum_{m=1}^{24} \left(538.15 \cdot (24 - m + 1)\right) = \mathbf{161{,}445}\$$$

Explanation - the first month of the fourth year we need to save 24 months, then the second month we need to save only 23 month and the last month we need to save only one month.

Total cost for 5 years plan (2 options):

1. With S3 Standard - $54{,}025.92\$ + 3{,}662{,}634.24\$ + 1{,}271{,}748\$ = \mathbf{4{,}988{,}408.16}\$$
2. With S3 Intelligent-Tiering - $6{,}613.92\$ + 464{,}961.6\$ + 161{,}445\$ = \mathbf{633{,}020.52}\$$

Assume that we can use S3 Intelligent-Tiering, then the total is $\mathbf{633{,}020.52}\$$

Note - for S3 INT we picked those setting because the data don't need to be access, so, in our thinking only 15% approximately of the data will be moving out of the Deep Archive Access.

EC2 pricing

For the first year we can use Linux machine, m4.xlarge with 4 vCPUs and memory of 16 GiB.
The cost for this instance for 1 year is 1,012.66$.
For the second stage upgrade our EC2 to m4.2xlarge with 8 vCPUs and memory of 32 GiB.
The cost for this instance for 1 year is 2,026.19$
The cost for this instance for 3 years is 3,952.51$
If this EC2 is not enough we can always upgrade it, the price for this component is neglect according to the rest of the system.

Total cost for 5 years plan - $1{,}012.66\$ + 2{,}026.19\$ + 3{,}952.51\$ = \mathbf{6{,}991.36}\$$
Note - All costs paid upfront.

Summary pricing:

1. RDS - $50{,}670.72\$$
2. Lambda - $6{,}539{,}811.46\$$
3. S3 - $633{,}020.52\$$
4. EC2 - $6{,}991.36\$$

$50{,}670.72\$ + 6{,}539{,}811.46\$ + 633{,}020.52\$ + 6{,}991.36\$ = \mathbf{7{,}230{,}494.06}\$$
We can estimate that the total cost for all the system 5 years plan is $\mathbf{7{,}200{,}000}\$$

Notes:

1. All the calculations used US East (Ohio) as the region, different regions different prices.
2. If we change the size of each picture the final cost will be totally different.
3. We provided the SMS pricing table but excluded from the calculation due lack of data.

Improving idea:

As we can see most of the cost comes from the Lambda function, we can try to reduce it by the following solution.

We can get new EC2 instance such as c5a.24xlarge with 96 vCPUs and memory of 192 GiB, for 73,000$~ for 6 years upfront. The instance will hold a queue of jobs, each job is to extract the features with the models and upload to our RDS server. Before the Lambda will do it, it will ask first the EC2 if is available to do the job, if EC2 is available then Lambda will skip the features extraction & won't upload the data to RDS, if EC2 isn't available then the Lambda will compute everything.

Why this idea is good? we can save a lot of time-computing and memory in the Lambda invocations and if there is high pressure on the system Lambda function will take place, such as in the morning and in the evening. We can see the new instance as kind of "middle-man", we chose that machine because we need more vCPUs than GiB and this kind of machines have great ratio of $\frac{\mathbf{vCPUs}}{\mathbf{GiB}}$.

It's hard to make correct estimation how much can save with this solution but $\frac{73{,}000}{6{,}539{,}811.46} = 0.011$, so, we need to assign only 1% of the jobs to be even. Obviously, this machine is a beast and can handle more than 1% of the total jobs.

Assuming it can handle 20% of the jobs → 1,307,962$ saving.

We can set-up multiple EC2 instances in order to take all the load of the Lambda and leave the Lambda only to handle peak times.

With that method we can do rough estimate that

$$50{,}670.72\$ + 500{,}000\$_{\text{Lambda}} + 633{,}020.52\$ + 6{,}991.36\$ + (73{,}000\$ \cdot 5_{\text{EC2}})$$
$$= \mathbf{1{,}555{,}682.6}\$$$

We can estimate now that the total cost for all the system 5 years plan is $\mathbf{1{,}500{,}000}\$$

This is a huge drop from $7.2M\$$ to $1.5M\$$