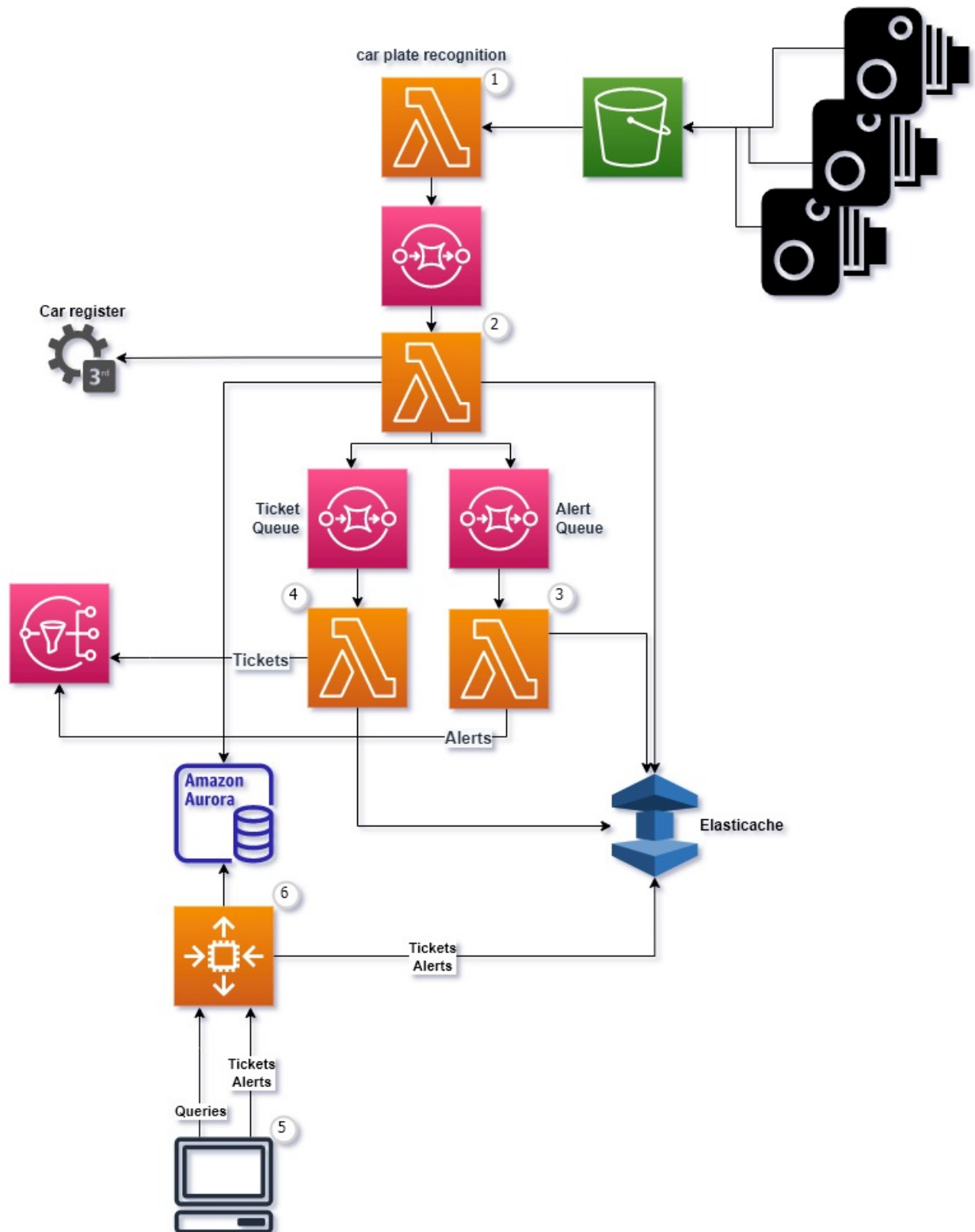


Cloud Computing Final Project

Table of Contents

ARCHITECTURE.....	3
HIGH LEVEL DATA FLOW	4
DATABASES.....	4
<i>Aurora</i>	4
<i>ElastiCache</i>	6
COMPUTATIONAL WORKFLOW	7
<i>Car Plate Recognition lambda</i>	7
<i>Event lambda</i>	7
<i>Alerts lambda</i>	8
<i>Tickets lambda</i>	8
<i>SQS</i>	9
<i>SNS</i>	10
<i>Management Service</i>	10
BILLING AND COSTS	12
GENERAL ALTERNATIVES	13
ATHENA	13
ALTERNATIVE: USING KUBERNETES	13
GROWTH POTENTIAL	13

Architecture



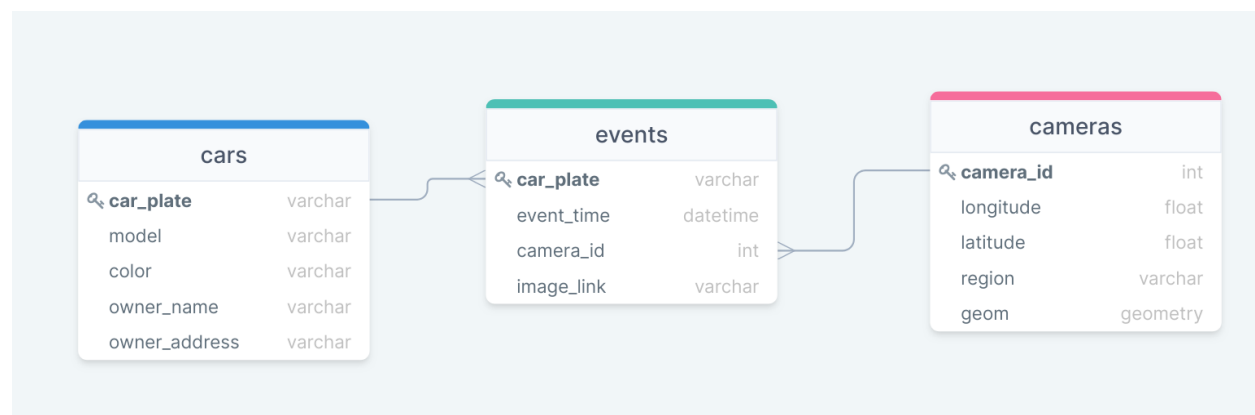
High Level Data Flow

0. Cameras uploads to S3 a zip file with actual image and its metadata in JSON file: the Camera ID and the picturing datetime.
1. The **Car Plate Recognition lambda** is triggered when a new zip file is uploaded to S3 bucket. In this lambda:
 - a. The actual image and the metadata will be retrieved from the zip file.
 - b. The Car plate number will be retrieved using OpenCV python package.
 - c. The result will be sent to the Event queue.
2. Then, Event queue triggers the **Event lambda** service for:
 - a. Calling the 3rd party API for collecting all needed data about the car and its owner, such as: car model, car color, owner name, owner address (any other data needed for the ticketing later).
 - b. Persist the event record in Aurora
 - c. Put event in ElastiCache with a short TTL period
 - d. Putting the data in the Alert and Ticket queues.
3. Alert queue triggers the **Alerts lambda** with all the event data. This lambda will:
 - a. Check each event with all the configured alerts in the durable ElastiCache
 - b. Sends alerts if found to the SNS service
4. Ticket queue triggers the **Tickets lambda** in parallel to the Alerts lambda. This lambda will:
 - a. Retrieve previous event for the same car from ElastiCache Events
 - b. Issue a fine to SNS if the cars passed the permitted speed
5. System User can manage the tickets/alerts and query events from a web application
6. All web application FE & BE are managed in the **Management service**.

Databases

Aurora

We will store the Events, Car Details and Cameras tables in Aurora serverless with the following schema.



Our data has a well-defined schema with relations between tables. Aurora provides built-in security, continuous backups, serverless compute, read replicas, and multi-Region replication. Aurora has storage scaling.

This DB will retain its data forever. It will be used for queries by the system user.

Costs:

We calculated the storage needed for the 5-year period according to SQL data type sizes.

	Year 1	Year 2-5	5 Years
# cameras	150	5,000	
# pics per day pre camera	12,500	17,500	
# pics per day	1,875,000	87,500,000	
# pics per second	22	1,013	
#new cars per day	18,750	875,000	
storage per day (bytes)	71,212,500	3,323,250,000	13,364,212,500
storage per year (GB)	24.22	1,130.46	4,546

event table			
camera_id	smallint	2	bytes
car_plate	varchar	9	bytes
event_time	datetime	6	bytes
image_link	varchar	20	bytes
total:		37	

cars table			
car_plate	varchar	9	bytes
model	varchar	9	bytes
color	varchar	10	bytes
owner_name	varchar	20	bytes
owner_address	varchar	50	bytes
total:		98	

*We only save the relative path in S3 bucket for the image link.

Since Aurora has storage scaling, we will not need to pay for the full storage in the 1st year.

1st year: We will need 25GB of storage we have 22 events/sec as a baseline. We assumed 4 hours a day of peak activity with a 3x IO. We also need full backup. Total cost for the 1st year will be 1,271.88\$

2nd-5th years: At the end of the 5 years, we will need less than 5TB of storage. We have 1013 events/sec, we assume the same 4 hours of peak hours and 3x IO. We will have 1,502.20 \$ per month at the end of the period. Since our IO stays the same during this period and the storage grows linearly, we will take the average cost per month to be about 1,300\$.

Total cost for all the 5 years will be 63,671.88\$

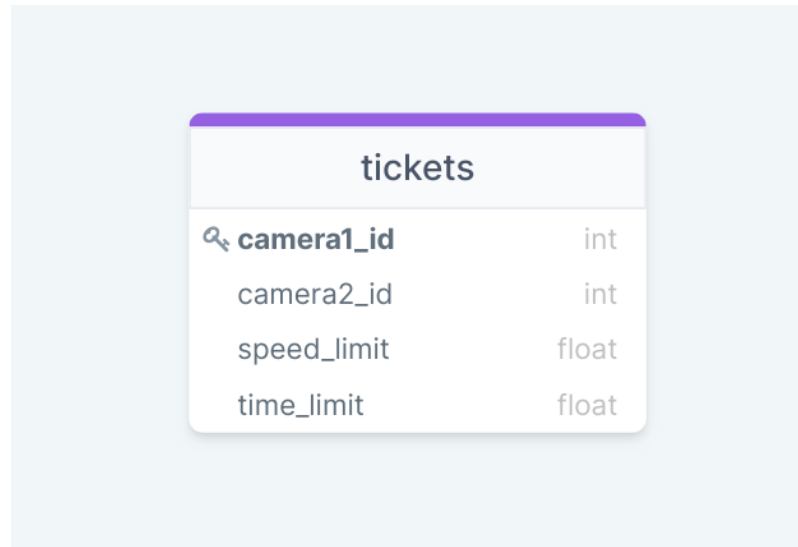
Alternatives:


- One option is to use a NoSQL database such as MongoDB, RavenDB, DynamoDB, Etc. However, our data have well defined schema, no nested objects and all the fields are required thus it's more suitable for SQL DB solution.
- Second option is to use RDS SQL other than Aurora serverless, in this case we would need to manage scaling up, security and other database related Devops management issues. As we assume high daily peaks for a short period of time, we will need a powerful machine, thus the serverless solution will be more suitable.

ElastiCache

We will store the Tickets, Alerts and recent events in ElastiCache for Redis.

Tickets schema:



tickets	
 camera1_id	int
camera2_id	int
speed_limit	float
time_limit	float

This schema contains the camera pairs, defined speed limit and the minimum amount of time possible to pass between two cameras under the speed limit, which will be calculated upon record insertion.

Alerts:

As there are many optional fields for configuring alerts conditions thus the schema is flexible, so we need a NoSQL database. For time-based alerts we will have an expiration date field, so we will be able to select only active alerts when needed.

Both alerts and tickets will be stored in durable ElastiCache for Redis.

Events:

As we only need recent events to find tickets and the only other use for the events table is for querying by a web application user, we will store only the most recent events in memory. As we don't anticipate two events that will result in a ticket being more than 6 hours apart, we choose to store 6 hours of event records.

Since we expect up to 3000 events/sec in peak hours, and for each event we need to query both alerts, tickets and recent events, we choose to store them in memory.

Cost:

For the 1st year: assuming 66 events/sec during peak time, each query takes 5ms, we choose 2 nodes comprised of cache.t2.small. For 1-year reserved plan the annual cost will be 396.72\$

For the 2nd-5th years: assuming 3000 events/sec peak, each query takes 5ms, we choose 2 nodes comprised of cache.m4.2xlarge vCPU: 8 with 29.7 GiB.

For 3-year reserved plan with upfront payment 4994\$ and 221.92\$ monthly cost. For a 4-year period the total cost will be around 17,310.83\$

Alternatives:

- One option is to save the alerts and tickets in some NoSQL database not in memory and using the Aurora serverless database for the events without storing recent records in memory. However, this will result in putting heavy load on the database (3000 queries/sec in peak time). Furthermore, RDS has maximum data API requests of 1000/sec for each region.
- Second option is to use EC2/physical machine and run Redis/Memcached on it. This option will add extra complexity that might require additional team members to deal with Devops related issues.

Computational Workflow

Car Plate Recognition lambda

This lambda will be triggered by a new image insertion to S3. This lambda decompresses the zip file and uses the OpenCV and pytesseract which are open-source libraries for car plate recognition (Apache 2.0 licence). This lambda will send the result to the Event queue upon completion.

Cost:

For the 1st year for 1,875,000 request/day with 150ms duration for each request (100 ms for the recognition step and 50ms for the rest) the cost will be 29.23\$ per month, resulting in 350.76\$ annual cost.

For the 2nd-5th years: for maximal 87,500,000 request/day, with similar duration, the monthly cost will be 1364\$, resulting in 65,472\$ for the 4-year period. The total cost will be 65,822.76\$.

Alternatives:

We can use the AWS Rekognition service, however this service has a limit of 1,000,000,000 images per month while our system requires to process over 2.6 billion images per month. 1,000,000,000 images processing will result in monthly cost of 263,450\$, much higher cost than the option we choose.

Event lambda

This lambda will be triggered by a new queue entry in the Event queue. The lambda will check if the car already exists in our DB, otherwise it will call a 3rd party API (the official national car register) for collecting all the needed data about the car and its owner, such as: car model, car

color, owner name, owner address (any other data needed for the ticketing later). It will persist the event record in Aurora and put the event in ElastiCache (only recent records as explained). The lambda will send the result to the Ticket queue and the Alert queue.

Cost:

For the 1st year, assuming 1% of daily new cars resulting in 18,750 request/day with 300ms duration for each request, the cost will be 0.47\$ per month, resulting in 5.64\$ annual cost. The other 99%, 1,856,250 request/day with 50ms duration for each request, the cost will be 17.17\$ per month, resulting in 206.04\$ annual cost. The total cost will be 211.68\$

For the 2nd-5th years: assuming 1% of daily new cars resulting in 875,000 request/day with 300ms duration for each request, the cost will be 21.95\$ per month, resulting in 263.4\$ annual cost.

The other 99%, 86,625,000 request/day with 50ms duration for each request, the cost will be 802.36\$ per month, resulting in 9,628.32\$ annual cost.

resulting in 39,566.88\$ for the 4-year period. The total 5-year cost will be 39,778.56\$.

Alerts lambda

This lambda will be triggered by a new queue entry in the Alert queue. The lambda will retrieve all active (not expired) alerts from the configured alerts in the durable ElastiCache and check each event against them. Once a relevant alert should be raised the lambda will send alert to the SNS service.

Cost:

For the 1st year for 1,875,000 request/day with 300ms duration for each request (10ms for retrieving the alerts configurations from ElastiCache and 150ms to check the event against these configurations) the cost will be 30.42\$ per month, resulting in 365.04\$ annual cost.

For the 2nd-5th years: for maximal 87,500,000 request/day, with similar duration, the monthly cost will be 1,419.44\$, resulting in 68,133.12\$ for the 4-year period. The total cost will be 68,498.16\$.

Tickets lambda

This lambda will be triggered by a new queue entry in the Ticket queue. The lambda runs in parallel to the Alerts lambda.

We assume that tickets settings will only be configured for adjacent cameras, thus we will need to retrieve only the previous event for the same car from ElastiCache Events.

The lambda will:

1. Retrieve previous event from ElastiCache.
2. Query the Tickets and check if the two cameras are defined as a pair and the time difference between the two events is less than the time_limit value.

3. Issue a fine to SNS if needed.

Cost:

For the 1st year for 1,875,000 request/day with 35ms duration for each request (10ms for retrieving the pervious event from ElastiCache and additional 25ms to query the Tickets collection) the cost will be 15.57\$ per month, resulting in 186.84\$ annual cost.

For the 2nd-5th years: for maximal 87,500,000 request/day, with similar duration, the monthly cost will be 726.36\$, resulting in 34,865.28\$ for the 4-year period. The total cost will be 35,052.12\$.

Alternatives:

As it's not urgent to send tickets immediately unlike alerts, therefore we can replace this lambda with EC2 that handles 1013 events/sec (87,500,000 request/day) throughout the day, so we will have some delays at peak hours and catch-up during downtime.

The EC2 server will retrieve a batch of events from the ElastiCache and will preform the same aforementioned steps. It holds the last processed event's timestamp, thus it doesn't require a queue.

Alternatives Cost:

For 1st year, we only need to process 22 events/sec, where each event takes 35ms, then a t2.micro will be enough and it's part of the free-tier.

For the 2nd-5th years: To process 1013 events/sec, where each event takes 35ms, we will need c5.9xlarge (36 cores). With all upfront payment 3-years reserved plan the Monthly cost will 410.34\$ resulting in 19,696.32\$ for the 4-year period.

If issuing tickets immediately isn't essential, we can save money by choosing this alternative. However, this solution is less scalable.

SQS

The SQS will be used to deal with asynchronous calls between the lambda, it will insure fault tolerance. We will use 3 queues: Event, Alert and ticket queues. Each one has a dedicated lambda function.

Cost:

For 1st year: for 3 queues, each queue will deal with 56,250,000 request/sec the monthly cost will be 22.20\$ per queue. The total annual cost for the 3 queues will be 799.2\$.

For the 2nd-5th years: for 3 queues, each queue will deal with around 2.6 billion request/sec the monthly cost will be 1,051.6\$ per queue. The total 4-year annual cost for the 3 queues will be 151,430.4\$.

Alternatives:

- One alternative is to use asynchronous invocation directly from another lambda. This will not cost us, however there will be no fault tolerance, no error handling and we will need to add additional logic to the lambdas.
- Second option is to use Step Functions, however the **monthly** cost during the 2nd-5th year period will be more than 250,000\$.

SNS

Cost:

For the 1st year: assuming 5000 tickets and alerts per day, the monthly cost for email notification will be 2.98\$, total annual cost is 35.76\$

For 2nd-5th years: assuming 50,000 tickets and alerts per day, the monthly cost for email notification will be 30.32\$, total 4-year cost is 1,455.36\$. total cost is 1,491.12\$

Management Service

This instance will handle all the tickets and alerts configurations and event queries from a web application.

Once a user defines a new alert, the instance will insert the new configuration to the Alerts in the ElastiCache.

Once a user defines a new ticket, the instance will insert a new record to the Tickets with the calculated time_limit value according to the camera pairs distance and the speed limit defined.

Once a user submits a query, the instance will query the Events table in Aurora DB and return the results (the images will be returned as S3 pre-signed URL).

This server will contain the frontend and backend code and opensource server for user management and authentication.

We assume 10,000 active users with 1000 requests/hour in the web application.

For purposes of fault tolerance, scalability, high availability we will considering using EC2 load balancing.

Cost:

This system will be idle most of the day, also it's acceptable for the queries' results to take a couple of seconds. At worst case there will be no more than 10 web users at the same time.

Therefore, we don't need a much powerful machine, so we choose a1.xlarge. For 1st year we'll use the free tier with t2.micro as we expect only few calls per day. For the 2nd-5th years, with upfront payment 3-year reserved plan, the monthly cost will be 28.83\$ with total cost of 1,383.84\$

Alternatives:

We will build a serverless web application, The web pages, CSS, JS files etc', will be hosted in AWS Amplify and the web page in browser will do rest API calls to API Gateway and lambda. In this case will use AWS Cognito for user management and authentication.

Cost:

We have continuous delivery; therefore, the monthly cost of AWS Amplify hosting will 10\$. Gateway and Lambda monthly costs are 0.25\$ and 3.2\$ respectively (assuming 1000 requests/hour). Furthermore, with 10,000 active users, AWS Cognito's monthly cost will be 14.25\$, resulting in annual cost of 332.4\$. The total cost for 5-years is 1,662\$.

The costs of the solution and alternative are similar, however the alternative is a more complex system.

Billing and Costs

	1 st year	2 nd -5 th years
Car Plate Recognition lambda	\$350.76	\$65,472.00
Event lambda	\$211.68	\$39,566.88
Alerts lambda	\$365.04	\$68,133.12
Tickets lambda	\$186.84	\$34,865.28
Management service	\$-	\$1,383.84
SNS	\$35.76	\$1,455.36
SQS	\$799.20	\$151,430.40
Aurora	\$1,271.88	\$62,400.00
ElastiCache	\$396.72	\$17,310.83
Total Infrastructure Cost:	\$3,617.88	\$442,017.71
5 Year Cost:	\$445,635.59	

General Alternatives

Athena

We considered using Athena, since we already save the image and the meta data in S3, we thought that we will save storage and costs by eliminating extra DB infrastructures. Upon further inspection we found that using Athena queries is very expensive (millions of \$ per month).

Alternative: Using Kubernetes

Kubernetes is a powerful tool to orchestrate, manage, monitor and scale services. it is widely used, and some cloud providers have their own implementation, for example Amazon EKS. In this alternative, we will replace the lambdas with micro services and ElastiCache with Redis pods on Kubernetes cluster. We will need an additional queue to connect between S3 and the Kubernetes cluster. We will keep using the AWS Aurora, SQS and SNS, however with Kubernetes we can also manage these services. This solution will enable us to have more control and less reliance on cloud services and it will save infrastructure costs. However, we will need more Devops and developers resulting in high overall costs.

Growth Potential

Our solution is comprised mainly of serverless services. We took into consideration dealing with peak hours and possible expansion to new regions, thus we made sure that every part is scalable, resilient, and high performance.