Final Assigment:

Architecture for a cloud system

Amit Zeevi - 206171902

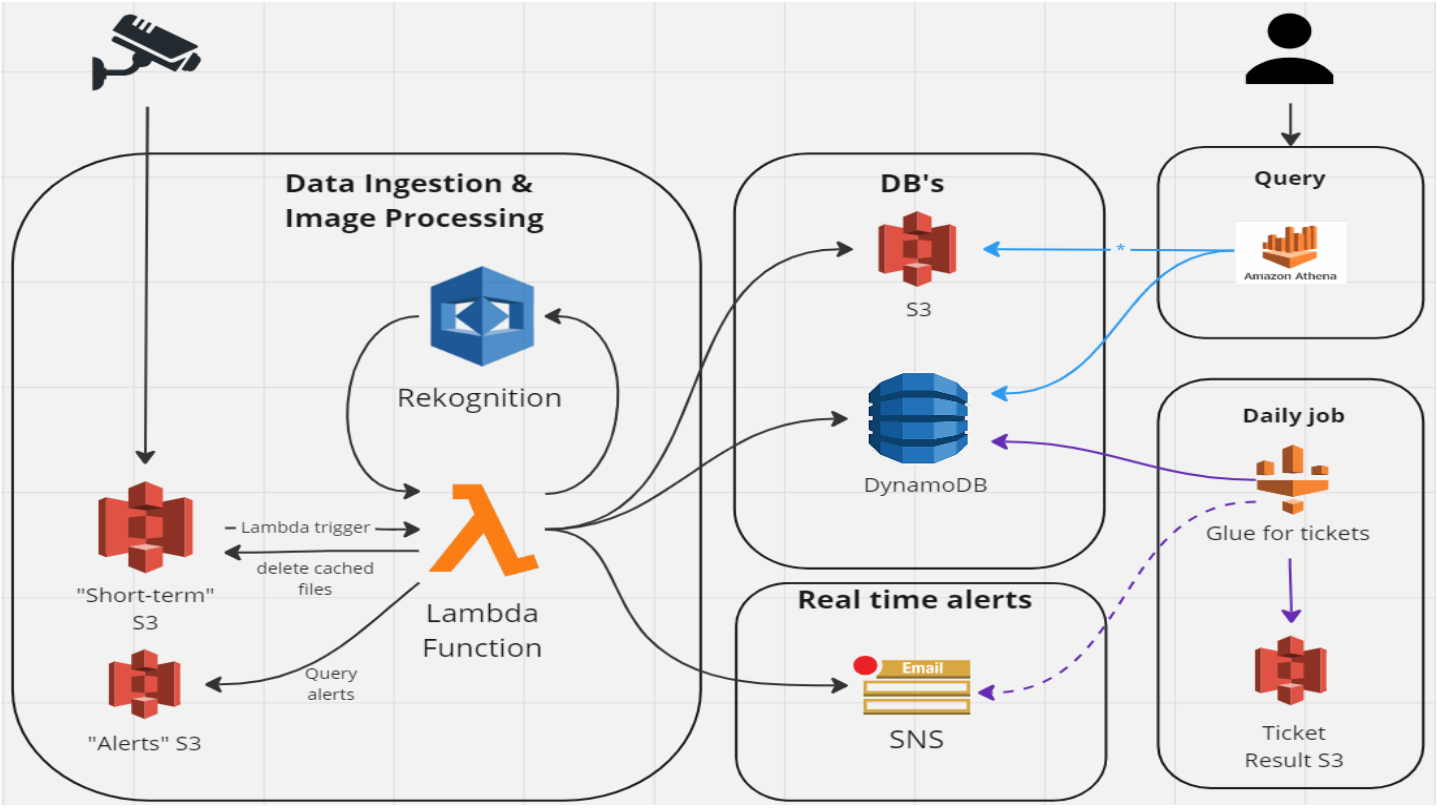
Amit Inbar – 208688374

August 2023

Overview – High level workflow

1. A new ZIP file is sent to an S3 bucket.
2. A Lambda event is triggered, which creates a new lambda function.
    a. The lambda function sends the image to AWS Rekognition service, and the response returned to the lambda are the details about the event – Car license, car color, etc.
    b. The lambda will check the output data against a set of conditions (Alerts) stored in an additional small S3 “alerts” bucket. Whenever a data point matches one or more of these conditions, it will call the API of SNS to trigger notifications via email and/or SMS.
    c. The actual image is stored in an additional S3 image bucket and returns a path.
    d. The lambda function packages the response along with the JSON metadata given and the path to the image as structured data - to store in DynamoDB.
3. An ETL Glue job will run daily, running logic to create the tickets from the last day.
4. A user would be able to query the database using AWS Athena, which will query the Dynamo DB and will get a response with a link to the image from the S3 image bucket.

Architecture



Detailed explanation of each component

Component #1 - Data Ingestion & Image Processing

The Data Ingestion & Image Processing component is responsible for receiving the images from the cameras, parsing them using Rekognition, and storing the results in the DB's. The component also generates alerts by comparing the results of the image processing and the current alerts stored in the "S3 alerts" bucket. Main services used in this component include:

- **Amazon S3 "Short-term" DB**
  - Purpose:

    Zip files created by the traffic cameras will be sent to the "short-term" DB for processing purposes. When the image is stored in the S3 DB, it will trigger a lambda event in order to start the process (explain further under Lambda service). Lastly, after the images are sent to Reckognition, they will be deleted from this DB such that each image will be stored in the DB (at most!) 1 hour.

  - Alternatives & Decisions:
    - **Amazon DynamoDB**:

      DynamoDB is a performance DB cloud service suitable for structured data. The reason why we've decided to go with S3 over DynamoDB is because S3 is more cost-effective to handle small amounts of data, and since each zip file will only be stored in this bucket for (presumably) only a few minutes until it is parsed and stored by the Lambda function the DB will be relatively small at any given point in time.

    - **Amazon Redis:**

      Redis is an in-memory key-pair cache database. Redis is extremely fast and good for accessing data frequently. But since in our case, the data is only accessed once, and our datatype stored in the database is a zip file which Redis does not support, it does not fit our requirements.

  - Cost:
    - Trial Run
      - Total images per day: 1,875,000
      - Size of each image: 250kb
      - Total size of images per day: 1,875,000 * 250kb = 468.75gb
      - Storage size per hour: 468.75gb / 24 hours = 20.35gb
      - Storage cost per day: $0.023/gb * 20.35gb = $0.47
      - Upload network traffic cost per day: 0$
      - PUT requests cost per day = (1,875,000/ 1000) * $0.005 = 9.375$
      - GET requests cost per day = (1,875,000/ 1000) * $0.0004 = 0.75$

- Download network traffic cost per day: $0.09/gb * (468.75gb - 100gb free) = $33.18
- Total cost per day: $0.47 + $33.18 = $33.55
- **Total cost per year (365 days): $33.55 * 365 = 12,245.75$**

- Full Deployment
  - Total images per day: 87,500,000
  - Size of each image: 250kb
  - Total size of images per day: 87,500,000 * 250kb = 21875gb
  - Storage cost per hour: 21875gb / 24 hours = 911.5gb
  - Storage cost per day: $0.023/gb * 911.5 gb = $20.96
  - Upload network traffic cost per day: 0$
  - PUT requests cost per day = (87,500,000/ 1000) * $0.005 = 437.5$
  - GET requests cost per day = (87,500,000/ 1000) * $0.0004 = 35$
  - Download network traffic cost per day: $0.09/gb * (21875gb - 100gb free) = 1,959.75$
  - Total cost per day: $20.96 + $437.5 + $35 + $1959.75 = $2,453.21
  - **Total cost per year (365 days): $237.2 * 365 = $895,421.65**

- **Lambda Function**
  - Purpose:

    The Lambda function is used to process the images that are uploaded to S3 and store them in the DB's for future access. The function will follow these steps:

    1. The function will be triggered by an image inserted in the S3 bucket and will send it to Reckognition to process objects in the images.
    2. The result processed data will first be compared to the alerts currently available in the S3 "alerts" DB and send an alert using Amazon's SNS service if any of the objects are of interest.
    3. Then, the data is split into 2 blocks – the image to store in S3 and all other required data (including path to image stored in S3) to be stored in DynamoDB.

    The Lambda function will act as an intermediate of data and will not require any complicated computation, therefore we can use the base 128MB of memory, and assume execution time of each invocation will be 10 seconds (includes time to get image from S3: 3-5 seconds and call to Reckognition: 2-5 seconds).

  - Alternatives & Decisions:
    - **Amazon EC2**:

The alternatives to using a Lambda function for this task would be to use Amazon EC2 pool of machines to query the S3 bucket and perform the corresponding operations instead+ of the Lambda instance. In this case the EC2 machine would have to be constantly running (at least some of them) even when idle, which is not cost-effective.

- Cost:
  - Trial Run
    - Number of invocations: 1,875,000
    - Cost of invocation: $0.000000002 * 1,875,000 * 10,000 (=10 seconds) = $37.5
    - Cost of memory (128MB): $0.0000005 * 0 = $0
    - Total cost per day: $37.5
    - **Total cost per year: $13,687.5**
  - Full Deployment
    - Number of invocations: 87,500,000
    - Cost of invocation: $0.000000002 * 87,500,000 * 10,000 (=10 seconds) = $1750
    - Cost of memory (128MB): $0.0000005 * 0 = $0
    - Total cost per day: $1750
    - **Total cost per year: $1750 * 365 = $638,750**

- **Rekognition**
  - Purpose:

    AWS Rekognition is a service that uses machine learning to identify objects, people, text, and scenes in images and videos. In our case we will use the service in order to parse the images sent by the cameras in order to detect parameters such as car color, license plate, etc. To enable us future querying and alerting on the event.

  - Alternatives & Decisions:
    - **Clarifai:**

    Clarifai is a cloud-based AI platform that can be used to identify objects, faces, text, and other features in images similar to Rekognition. Clarifai is more customizable and flexible than Rekognition allowing you to choose your own models and supports wider forms of data. However, we chose Rekognition over clarifai because it is considered more accurate and reliable and also more scalable than clarifai which will allow it to grow with our system.

    - **Google Cloud Vision API:**

Like Clarifai, Google Cloud Vision API is also more customizable and flexible than Rekognition. Furthermore, Cloud Vision is also (a little) more affordable than Reckognition. However, also in this case we decided to prioritize the accuracy of the service and chose to go with Reckognition.

- Cost:

We will use the "Group 2" pricing tier of the service such that cost is $0.001 for first 1 million images, then $0.0008 for next 4 million images, then lastly $0.0006 for next 30 million images.

  - Trial Run:
    - Total number of images per day: 1,875,000 (56M per month)
    - Cost of first 1M images: 1,000,000 * $0.001 = $1000
    - Cost of next 4M images: 4M * $0.0008 = $3,200
    - Cost of next 30M images: 30M * $0.0006 = $18,000
    - Cost of next 21M images: 21M * $0.00025 = $5,250
    - Total cost per month: $27,450
    - **Total cost per year: $329,400**
  - Full Deployment:
    - Total number of images per day: 87,500,000 (2.6B per month)
    - Cost of first 1M images: 1M * $0.001 = $1000
    - Cost of next 4M images: 4M * $0.0008 = $3,200
    - Cost of next 30M images: 30M * $0.0006 = $18,000
    - Cost of next 2.59B images: 2.59B * $0.00025 = $647,500
    - Total cost per month: $669,700
    - **Total cost per year: $8,036,400**

S3 Alerts DB

- Purpose:

The purpose of this very small component is to allow uploading rules for alerts for the Lambda function to query and trigger. For example, you might upload rule to "alert if a blue Toyota is seen in the South Region for the next 6 hours". Then every Lambda invocation will query the rules in the S3 "Alerts" DB to determine if it needs to raise an alert.

- Alternatives & Decisions:
  - **No service at all:**

Theoretically, you could upload the rules directly into the lambda function "hard-coded". Though technically possible (and of course cost efficient), it is not an

elegant solution and will require changing the core functionality of the function often, which is definitely not best practice.

- **DynamoDB:**

Another possible solution is to store the rules in a DynamoDB which will work just the same as the rules data is very structured. DynamoDB was a valid second option, but we decided to go with S3 because it is more cost efficient at small scales such as the case of storing the alerts.

- Cost:
  - Trial Run:
    - Number of calls per day: 1,875,000
    - GET requests cost per day = (1,875,000/ 1000) * $0.0004 = 0.75$
    - Storage cost per day: free under 5gb = $0
    - Total cost per day: $0.75
    - **Total cost per year: $0.75 * 365 = $273.75**
  - Full Deployment:
    - Number of calls per day: 87,500,000
    - GET requests cost per day = (87,500,000/ 1000) * $0.0004 = 35$
    - Storage cost per day: free under 5gb = $0
    - Total cost per day: $35
    - **Total cost per year: $24 * 365 = $12,775**

Component #2 – Data bases

NOTE #1 – We chose to split the Images and the ephemeral data JSON into 2 DB's because we decided that DynamoDB will be a better fit storing only the data about the cars in very large scales (retention time is forever), and yet will not be able to store the images efficiently since they are not structured data. For the task of storing the images for 3 years, S3 is perfect for the task. So, we concluded that storing the car data in DynamoDB with a pointer (URL) to the image in S3 (within their retention time) will be the most efficient solution.

- **Amazon S3 "Long-term" DB**
  - Purpose:

    This S3 DB will be used to store the Images for the entire duration of their retention time after processing is completed. We will also be using the lifecycle management feature to delete the images after 3 years (or other specified retention time).

    NOTE #2 – We chose to split the images "short-term" and "long-term" DB's into 2 DB's because in our calculation we observed, because S3 does not require us to pay for

inbound network traffic, this separation will incur very little added cost deleting the images and transferring them from to the short-term DB (since the image is downloaded by the Lambda function anyways) - a total of 1 GET and 1 PUT request per image. On the other hand, we benefit from this in 2 ways: firstly, images and ephemeral data sent by that have not yet been processed will not be scanned by GLUE jobs which will save us cost and time querying the data. Secondly, it makes more sense to separate the 2 DB's development wise as they have separate responsibilities.

- Alternatives & Decisions:
  - **DynamoDB:**

    For the same reasons mentioned for the "short-term" S3 DB.

  - **DigitalOcean Spaces:**

    Spaces is a similar cloud service to S3 provided by DigitalOcean. The service is a Generally less expensive than S3 for large amounts of data with most of the same functionality. But even though Spaces is less mature and considered less reliable than S3, in addition Spaces also offers fewer features (like Data lifecycle management which we to delete images by retention).

  - **Google Cloud Storage / Microsoft Azure Blob Storage:**

    Both services are similar to S3 with somewhat similar pricing plans and features. We chose S3 over Both GCS and ABS because S3 offers much better integrations to our other services such as Athena, Glue etc. All managed by AWS.

- Cost:
  - Trial Run:
    - Total images per day: 150 * 12,500 = 1,875,000
    - Size of each image: 250kb
    - Total size of images per day: 1,875,000 * 250kb = 468.75gb
    - Additional storage cost per day: $0.023/gb * 468.75 gb = $10.78
    - Upload network traffic cost per day: 0$
    - PUT requests cost per day = (1,875,000/ 1000) * $0.005 = 9.375$
    - **Total cost per year (365 days): ($10.78 + 9.375$) * 365 = 7,356.575$**
  - Full Deployment
    - Total images per day: 5000 * 17,500 = 87,500,000
    - Size of each image: 250kb
    - Total size of images per day: 87,500,000 * 250kb = 21875gb
    - Additional storage cost per day: $0.023/gb * 21875gb = $503.125
    - Upload network traffic cost per day: 0$
    - PUT requests cost per day = (87,500,000/ 1000) * $0.005 = 437.5$
    - Total cost per day: $503.125 + $437.5 = $940.625

- **Total cost per year (365 days): $940.625 * 365 = $343,328**

- **DynamoDB for storing car data**
  - Purpose:

    The DynamoDB is used to store car data forever. This data includes camera information, Reckognition results, etc. And any data relevant for data querying by users. The data will be stored as structured JSON data.

  - Alternatives & Decisions:
    - **Amazon S3:**

    As mentioned before Amazon S3 is a scalable, durable, and highly available object cloud storage service. In our case, for storing structured data in very large volumes, DynamoDB is a much more cost-effective option with significantly lower prices for both storage and data transfer.

    - **Amazon Redshift**

    Amazon Redshift is a fully managed, petabyte-scale relational data warehouse service that offers high performance and scalability. It is a popular choice to store and analyze large amounts of data. Though Redshift would allow us to scale our data. It is not as good as an option as DynamoDB because it is less cost effective than DynamoDB especially in our case of a write-heavy application compared to the read-heavy which Redshift is optimized for.

  - Cost:

Assume for this exercise that each car data entry is 1Kb (1024 characters).

- Trial Run:
  - Number of entries per day: 1,875,000
  - Average entry size: 1 kb.
  - Number of write-requests per month: 1,875,000 * 30 days = 56,250,000.
  - Cost of monthly write-requests: $1.25 * 56,250,000 / 1,000,000 = $66.25.
  - Cost of yearly write-requests: $66.25 * 12 months = $795.
  - Size of entries monthly: 1,875,000 * 1kb * 30 days = 5.625 GB
  - Cost of storage monthly: $0.25 * 5.625 GB = $1.41
  - Cost of storage yearly: $1.41 * 12 months = $16.92.
  - **Total cost yearly: $795 + $16.92 = $811.92.**
- Full Deployment:
  - Number of entries per day: 87,500,000
  - Average entry size: 1 kb.

- Number of write-requests per month: 87,500,000 * 30 days = 2,625,000,000.
- Cost of monthly write-requests: $1.25 * 2,625,000,000 / 1,000,000 = $3281.25.
- Cost of yearly write-requests: $3281.25 * 12 months = $39375.
- Size of entries monthly: 87,500,000 * 1kb * 30 days = 26.25 GB.
- Cost of storage monthly: $0.25 * 26.25 GB = $6.56
- Cost of storage yearly: $6.56 * 12 months = $78.72.
- **Total cost yearly: $39,375 + $78.72 = $39,453.72.**

Component #3 – Alerts

- **SNS (Simple Notification Service)**
  - Purpose:

    AWS SNS (Simple Notification Service) is a fully managed messaging service that provides a reliable, scalable, and flexible way to send messages in a variety of formats. In our case, we will use the SNS service to alert the relevant parties

  - Alternatives & Decisions:
    - **AWS SES (Simple Email Service)**

      SES has similar functionalities to SNS, both are free up-to 1 million messages and 1,000 emails so cost is not a factor either. Ultimately, we chose SNS over SES because it is considered simpler to use and has additional supporting features such as message retrying, message dead-lettering etc. That might be useful in the long run.

    - **Twilio**

      Twilio is a cloud communications platform that enables developers to build communications into their web and mobile applications. Twilio offers a wide range of features, including SMS, MMS, Voice, Email, Chat, etc. Twilio meets all our requirements and more, but it is a more expensive offer, especially since it does not allow 1 million free messages.

  - Cost:

    (Assume 1/1000 cars will trigger an alert. And 50% are SMS and 50% are Email) SNS allows a free tier of 1 million SMS messages and 1,000 email messages per month. After that, the price is $0.50 per million SMS's and $2.00 per 100,000 emails.

    - Trial Run:
      - Number of messages per day: 1,875,000 / 1000 = 1,875
      - Number of SMS per day: 1,875 /2 = 938

- Number of SMS per month: 938 * 30 = 28,140
- Cost of SMS messages per month: $0 (free tier)
- Number of Email per day: 1,875 /2 = 938
- Number of Email per month: 938 * 30 = 28,140
- Cost of Email messages per month: $2 (free tier)
- **Total cost per year: $2 * 12= $24**

- Full Deployment:
  - Number of messages per day: 87,500,000 / 1000 = 87,500
  - Number of SMS per day: 87,500 / 2 = 43,750
  - Number of SMS per month: 43,750 * 30 = 1,312,500
  - Cost of SMS messages per month: (1,312,500–1,000,000)/1,000,000 * $0.5 = $0.5
  - Number of Email per day: 87,500 / 2 = 43,750
  - Number of Email per month: 43,750 * 30 = 1,312,500
  - Cost of Email messages per month: (1,312,500-1,000)/100,000*$2 = $28
  - **Total cost per year: $28 * 12= $336**

Component #4 – Query

- **Amazon Athena**
  - Purpose:
    Amazon Athena is a serverless interactive query service provided by AWS. Its main purpose is to enable users to analyze and query data directly from Amazon S3 using standard SQL queries. It allows you to quickly gain insights from large-scale data without the need to set up and manage complex infrastructure.
    We choose this service for several reasons:
    i. **Integration with Data Sources**: Since our data is already stored in Amazon S3 and DynamoDB, Athena provides seamless integration with these services, allowing direct querying without data movement.
    ii. **Performance and Scalability**: Amazon Athena is optimized for handling large-scale data sets, ensuring fast query performance even when dealing with massive traffic data.
    iii. **Ease of Use**: With its support for standard SQL, Athena allows developers and analysts to utilize their SQL skills effectively, streamlining the querying process.
  - Alternatives & Decisions:
    - **Google BigQuery:**
      Google BigQuery is a fully-managed serverless data warehouse provided by Google Cloud Platform (GCP). It allows you to run SQL-like queries on

massive datasets with high performance and scalability. BigQuery supports real-time data analysis and integrates well with other GCP services and popular data tools. Pricing is based on a pay-as-you-go model, charging for the amount of data processed. Since the value we get here is very similar to our choice, for the ease of use we decided to stay in the same Amazon environment.

- **Presto:**
  Presto is an open-source distributed SQL query engine that can query data from various data sources including S3.
  It is designed for high performance and can handle large datasets efficiently. Presto is highly customizable and can be integrated with other data processing frameworks. In this case, we preferred Amazon Athena since it's a fully managed and integrated system with S3 that will allow us to easier maintenance when the system scales.

- Cost:
  Amazon Athena works with "per query billing", meaning you pay for the amount of data scanned by each query.
  - Trial run
    - Price per query per 1 TB of data scanned – 5$
    - Total number of images per day: 1,875,000
    - Size of metadata and Json per image – 1Kb
    - Daily data size - 1.78 GB
    - Yearly data size – 650 GB
    - Assumming 4 queries per day, Monthly cost: 650GB = 1 TB * 4 * 5$ *30 days = 600$
    - **Total Yearly cost: 600$ * 12 = 7,200$**
  - Full deployment
    - Price per query per 1 TB of data scanned – 5$
    - Total number of images per day: 18,750,000
    - Size of metadata and Json per image – 1Kb
    - Daily data size – 83 GB
    - Yearly data size – 30 TB
    - Assumming 4 queries per day, Monthly cost: 30TB * 4 * 5$ * 30 days = 18,000$
    - **Total Yearly cost: 18,000$ * 12 = 216,000$**

Component #5 – Ticketing generation

- **Amazon glue job**

- Purpose:
  A daily glue job will handle ticket creation. Each pair of cameras will be configured in the code, together with their distance and speed limit. Once a day the glue job will process all the data generated from the previews day and with simple logic will create the speed ticketing. Tickets will be stored in an S3 bucket.
  (Though not mentioned specifically in the requirements, tickets will be stored in an S3 bucket and could easily be sent to the relevant party using the SNS service)

- Alternatives & Decisions:
  - **Talend Data Integration:**
    Talend Data Integration is a comprehensive and widely-used data integration platform that provides ETL and data quality capabilities. We chose the glue over it because of its visual interface and the fact that it is a fully managed service. We assume this will allow easier configuration of pairs of cameras in the future.

  - **Microsoft Azure Data Factory:**
    Azure Data Factory is a cloud-based data integration service provided by Microsoft Azure. It offers ETL and data orchestration capabilities to build data pipelines for data movement and transformation. Since the value we get here is very similar to our choice, for the ease of use we decided to stay in the same Amazon environment.

- Cost:
  - Trial run
    - Total number of rows to process – 1,875,000 = 1.87 GB
    - Price per hour per DPU – 0.44$
    - Dynamo DB read cost (monthly) – 7.13$
    - Storage cost per year- 55 MB, free under 5 GB, hence 0$
    - Assuming loosely 2 hours for the process
    - Yearly price – 7.3$ * 12 months + (2hours * 0.44$) * 365 days = 408.8$
  - Full deployment
    - Total number of rows to process – 18,750,000 = 87.5GB
    - Price per hour per DPU – 0.44$
    - Dynamo DB read cost (monthly) – 71.3$
    - Storage cost per year- 1.5 GB, free under 5 GB, hence 0$
    - Assuming loosely 4 hours for the process
    - Yearly price – 73.1$ * 12 months + (4 hours * 0.44$) * 365 days = 1,519$

Growth Potential

Theoretically, the growth potential of the service is infinite. All services we have chosen as part of the architecture described above (including S3, DynamoDB, Lambda, etc.) have unbounded scale and can be grown to fit whatever scale the system will require.

That being said, in order to preserve the systems integrity and efficiency at extremely large scales, we can take steps to best prepare for such situations:

1. Analyze how often is data accessed via the system per time frame and adjust: For example, if we observe entries older than X years are very rarely accessed, we can consider moving large amounts of data older than X years to Glacier type storage to significantly reduce costs and load off our DB's and Athena/GLUE jobs.
2. We can also "shard" the DB's based on grouping of geographical areas (per city/state) to allow more specific searches which would be more efficient both in time and cost of query.

Breakdown of estimated cost (in $):

Purpose	Service	cost for a year of pilot	cost for a year of full phase
Data Ingestion & Image Processing	Amazon S3 "Short-term" DB	12,245	895,421
	Lambda Function	13,687	638,750
	Rekognition	329,400	8,036,400
	S3 Alerts DB	273	12,775
Data Bases	Amazon S3 "Long-term" DB	7,356	895,421
	DynamoDB for storing car data	811	39,453
Alerts	SNS	24	336
Query	Amazon Athena	7,200	216,000
Ticketing generation	Amazon Glue job	408	1,519
Total		371,404	10,736,075

Appendix 1 - Data sizes assumptions

Pilot

Item	Size of 1	Size in day**	Size in month	Size in year
Image	250 KB	467GB	14.1TB	170TB
JSON (metadata)	1 KB	1.87GB	56.1GB	682.6GB
Tickets row*	1 KB	150 KB	4.5MB	54.7MB

full activity

Item	Size of 1	Size in day***	Size in month	Size in year
Image	250 KB	21.9TB	656.3TB	7.984PB
JSON (metadata)	1 KB	87.5GB	2.625TB	31.937TB
Tickets row*	1 KB	5MB	150MB	1.8GB

* Assuming each camera sends out one ticket per day

** 12,500 * 150 = 1,875,000 images per day

*** 17,500 * 5,000 = 87,500,000 images per day

Appendix 2 - Dynamo DB example of Data:


```
{
  "uuid":1,
  "date": "24.06.2023",
  "time": "00:00 UTC",
  "cameraId": 123,
```

```
  "cameraLocation": "453:263:000:856"
  “RekognitionResults”: {
     "licingNumber": 123-123-123,
     "carColor": "Red",
     "carType":"Toyota",
  },
  "imageLink": " https://s3.amazonaws.com/..."
}
```