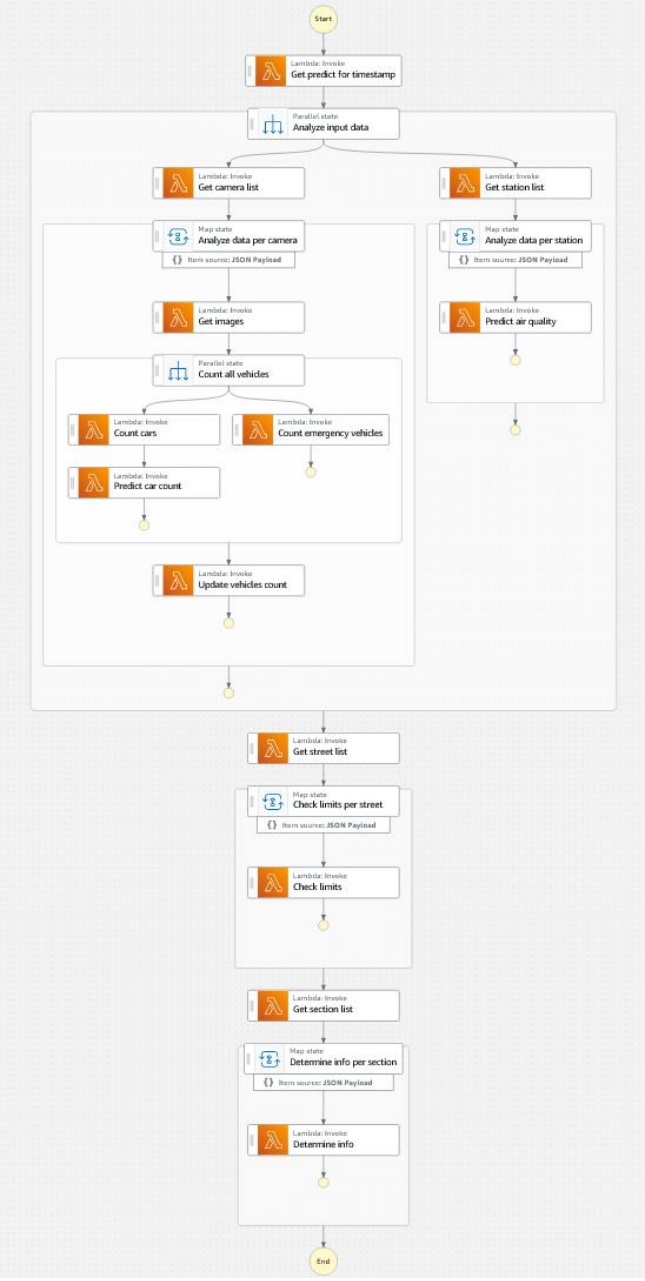
MS02 Submission

Workflow

Due to setup problems with Apollo, the workflow is implemented as an AWS Step Function.



In a first implementation all steps are implemented with Lambda functions. Should Lambda functions appear to be plain wrappers for boto3 calls to other AWS resources, it is conceivable to switch to direct interactions with these resources (e.g. DynamoDB:GetItem, etc.). Such an approach could reduce operational cost and at the same time eliminate cold start times of Lambda functions.

Data flow is not limited to the workflow alone but involves a single DynamoDB table and an S3 bucket for storing images.

The workflow is designed via AWS CDK. The resulting state machine definition can be found in the last chapter of this submission.

Task Definitions

Task: Get predict for timestamp

Creates a POSIX timestamp to define the time for which predictions/info updates are desired. For developing/debugging the workflow this value should be hardcoded considering the test dataset. For production this value should be generated from a system clock.

Return value to workflow:

- type: int
- value: the POSIX timestamp for which predictions/info updates are desired

Task: Get camera list

Gets a list of all available cameras and returns their IDs. Camera IDs are stored in the DynamoDB table and should be retrieved from the relations defined there.

Return value to workflow:

- type: list
- values: the IDs of all cameras in the system

Task: Get images

Gets all image URIs for a given camera considering the desired time of prediction. No images older than a defined amount of time should be considered. Image URIs are stored in the DynamoDB table.

Return value to workflow:

- type: dictionary
- keys: POSIX timestamps as integers
- values: the key under which the image is available within the bucket

Task: Count cars

Counts cars in a given list of images by using Rekognition.

Return value to workflow:

- type: dictionary
- keys: POSIX timestamps as integers
- values: the number of cars in each image as an integer

Task: Predict car count

Uses car counts for specific times to predict the car count for another time using linear regression or any other reasonably simple prediction model.

Return to workflow:

- type: int
- value: the number of cars predicted for the given time

Task: Count emergency vehicles

Counts emergency vehicles in the latest image of an image series by using Rekognition.

Return value to workflow:

- type: integer
- value: the number of emergency vehicles in the latest image

Task: Update vehicles count

Takes car count prediction and emergency vehicle count for a specific camera and a specific prediction time and stores the result in the central DynamoDB table.

Task: Get station list

Gets a list of all available stations and returns their IDs. Station IDs are stored in the DynamoDB table and should be retrieved from the relations defined there.

Return value to workflow:

- type: list
- values: the IDs of all stations in the system

Task: Predict air quality

Retrieves measurements for a specific station from the DynamoDB table and makes a prediction for a specified time using linear regression or any other suitably simple model. No measurements older than a defined amount of time should be considered. The prediction is stored in the DynamoDB table again.

Task: Get street list

Gets a list of all available streets and returns their IDs. Street IDs are stored in the DynamoDB table and should be retrieved from the relations defined there.

Return value to workflow:

- type: list
- values: the IDs of all streets in the system

Task: Check limits

Determines if limits for a given street are currently being exceeded, e.g.:

- current traffic load as percentage of maximum capacity
- active emergency vehicles (true/false)

Input data is retrieved from the DynamoDB table:

- Traffic prediction/analysis for all cameras on the street
- Air quality prediction for the station covering the street

The result is stored in DynamoDB directly.

Task: Get section list

Gets a list of all available sections and returns their IDs. Section IDs are stored in the DynamoDB table and should be retrieved from the relations defined there.

Return value to workflow:

- type: list
- values: the IDs of all sections in the system

Task: Determine info

Determines the info to display for a given section and a given time, e.g.:

- adjusted speed limit
- additional information (traffic jam, emergency vehicles active)

Input data is retrieved from the DynamoDB table:

- current traffic load as percentage of maximum capacity
- active emergency vehicles (true/false)
- unrestricted speed limit for the given section

The result is stored in DynamoDB directly.

State Machine Definition

```
{
    "StartAt": "Get predict for timestamp",
    "States": {
      "Get predict for timestamp": {
        "Next": "Analyze input data",
        "Retry": [
          {
            "ErrorEquals": [
              "Lambda.ClientExecutionTimeoutException",
              "Lambda.ServiceException",
              "Lambda.AWSLambdaException",
              "Lambda.SdkClientException"
            ],
            "IntervalSeconds": 2,
            "MaxAttempts": 6,
            "BackoffRate": 2
          }
        ],
        "Type": "Task",
        "ResultSelector": {
          "predictFor.$": "$"
        },
        "Resource": <get-predict-for-timestamp Lambda ARN here (in quotes)>
      },
      "Analyze input data": {
        "Type": "Parallel",
        "ResultPath": null,
        "Next": "Get street list",
        "Branches": [
          {
            "StartAt": "Get camera list",
            "States": {
              "Get camera list": {
                "Next": "Analyze data per camera",
                "Retry": [
                  {
                    "ErrorEquals": [
                      "Lambda.ClientExecutionTimeoutException",
                      "Lambda.ServiceException",
                      "Lambda.AWSLambdaException",
                      "Lambda.SdkClientException"
                    ],
                    "IntervalSeconds": 2,
                    "MaxAttempts": 6,
                    "BackoffRate": 2
                  }
                ],
                "Type": "Task",
                "ResultPath": "$.cameraIds",
                "Resource": <get-camera-list Lambda ARN here (in quotes)>
              },
              "Analyze data per camera": {
                "Type": "Map",
                "End": true,
                "Parameters": {
                  "cameraId.$": "$$.Map.Item.Value",
                  "predictFor.$": "$.predictFor"
                },
                "Iterator": {
                  "StartAt": "Get images",
                  "States": {
                    "Get images": {
                      "Next": "Count all vehicles",
                      "Retry": [
                        {
                          "ErrorEquals": [
                            "Lambda.ClientExecutionTimeoutException",
                            "Lambda.ServiceException",
                            "Lambda.AWSLambdaException",
                            "Lambda.SdkClientException"
                          ],
                          "IntervalSeconds": 2,
                          "MaxAttempts": 6,
                          "BackoffRate": 2
                        }
                      ],
                      "Type": "Task",
                      "ResultPath": "$.imageUris",
                      "Resource": <get-images Lambda ARN here (in quotes)>
                    },
                    "Count all vehicles": {
                      "Type": "Parallel",
                      "ResultPath": "$.counts",
                      "Next": "Update vehicles count",
                      "Branches": [
                        {
                          "StartAt": "Count cars",
                          "States": {
                            "Count cars": {
                              "Next": "Predict car count",
                              "Retry": [
                                {
                                  "ErrorEquals": [
                                    "Lambda.ClientExecutionTimeoutException",
                                    "Lambda.ServiceException",
                                    "Lambda.AWSLambdaException",
                                    "Lambda.SdkClientException"
                                  ],
                                  "IntervalSeconds": 2,
                                  "MaxAttempts": 6,
                                  "BackoffRate": 2
                                }
                              ],
                              "Type": "Task",
                              "InputPath": "$.imageUris",
                              "ResultPath": "$.carCount",
                              "Resource": <count-cars Lambda ARN here (in quotes)>
                            },
                            "Predict car count": {
                              "End": true,
                              "Retry": [
                                {
                                  "ErrorEquals": [
                                    "Lambda.ClientExecutionTimeoutException",
                                    "Lambda.ServiceException",
                                    "Lambda.AWSLambdaException",
                                    "Lambda.SdkClientException"
                                  ],
                                  "IntervalSeconds": 2,
                                  "MaxAttempts": 6,
                                  "BackoffRate": 2
                                }
                              ],
```

```
                    "Type": "Task",
                    "ResultSelector": {
                      "carCountPrediction.$": "$"
                    },
                    "Resource": <predict-car-count Lambda ARN here (in quotes)>
                  }
                }
              },
              {
                "StartAt": "Count emergency vehicles",
                "States": {
                  "Count emergency vehicles": {
                    "End": true,
                    "Retry": [
                      {
                        "ErrorEquals": [
                          "Lambda.ClientExecutionTimeoutException",
                          "Lambda.ServiceException",
                          "Lambda.AWSLambdaException",
                          "Lambda.SdkClientException"
                        ],
                        "IntervalSeconds": 2,
                        "MaxAttempts": 6,
                        "BackoffRate": 2
                      }
                    ],
                    "Type": "Task",
                    "InputPath": "$.imageUris",
                    "ResultSelector": {
                      "emergencyVehicleCount.$": "$"
                    },
                    "Resource": <count-emergency-vehicles Lambda ARN here (in quotes)>
                  }
                }
              }
            ],
            "ResultSelector": {
              "carCountPrediction.$": "$[0].carCountPrediction",
              "emergencyVehicleCount.$": "$[1].emergencyVehicleCount"
            }
          },
          "Update vehicles count": {
            "End": true,
            "Retry": [
              {
                "ErrorEquals": [
                  "Lambda.ClientExecutionTimeoutException",
                  "Lambda.ServiceException",
                  "Lambda.AWSLambdaException",
                  "Lambda.SdkClientException"
                ],
                "IntervalSeconds": 2,
                "MaxAttempts": 6,
                "BackoffRate": 2
              }
            ],
            "Type": "Task",
            "Resource": <update-vehicles-count Lambda ARN here (in quotes)>
          }
        }
      },
      "ItemsPath": "$.cameraIds",
      "MaxConcurrency": 40
    }
  }
},
{
  "StartAt": "Get station list",
  "States": {
    "Get station list": {
      "Next": "Analyze data per station",
      "Retry": [
        {
          "ErrorEquals": [
            "Lambda.ClientExecutionTimeoutException",
            "Lambda.ServiceException",
            "Lambda.AWSLambdaException",
            "Lambda.SdkClientException"
          ],
          "IntervalSeconds": 2,
          "MaxAttempts": 6,
          "BackoffRate": 2
        }
      ],
      "Type": "Task",
      "ResultPath": "$.stationIds",
      "Resource": <get-station-list Lambda ARN here (in quotes)>
    },
    "Analyze data per station": {
      "Type": "Map",
      "End": true,
      "Parameters": {
        "stationId.$": "$$.Map.Item.Value",
        "predictFor.$": "$.predictFor"
      },
      "Iterator": {
        "StartAt": "Predict air quality",
        "States": {
          "Predict air quality": {
            "End": true,
            "Retry": [
              {
                "ErrorEquals": [
                  "Lambda.ClientExecutionTimeoutException",
                  "Lambda.ServiceException",
                  "Lambda.AWSLambdaException",
                  "Lambda.SdkClientException"
                ],
                "IntervalSeconds": 2,
                "MaxAttempts": 6,
                "BackoffRate": 2
              }
            ],
            "Type": "Task",
            "Resource": <predict-air-quality Lambda ARN here (in quotes)>
          }
        }
      },
      "ItemsPath": "$.stationIds",
      "MaxConcurrency": 40
    }
  }
}
]
},
"Get street list": {
```

```
      "Next": "Check limits per street",
      "Retry": [
        {
          "ErrorEquals": [
            "Lambda.ClientExecutionTimeoutException",
            "Lambda.ServiceException",
            "Lambda.AWSLambdaException",
            "Lambda.SdkClientException"
          ],
          "IntervalSeconds": 2,
          "MaxAttempts": 6,
          "BackoffRate": 2
        }
      ],
      "Type": "Task",
      "ResultPath": "$.streetIds",
      "Resource": <get-street-list Lambda ARN here (in quotes)>
    },
    "Check limits per street": {
      "Type": "Map",
      "ResultPath": null,
      "Next": "Get section list",
      "Parameters": {
        "streetId.$": "$$.Map.Item.Value",
        "predictFor.$": "$.predictFor"
      },
      "Iterator": {
        "StartAt": "Check limits",
        "States": {
          "Check limits": {
            "End": true,
            "Retry": [
              {
                "ErrorEquals": [
                  "Lambda.ClientExecutionTimeoutException",
                  "Lambda.ServiceException",
                  "Lambda.AWSLambdaException",
                  "Lambda.SdkClientException"
                ],
                "IntervalSeconds": 2,
                "MaxAttempts": 6,
                "BackoffRate": 2
              }
            ],
            "Type": "Task",
            "Resource": <get-section-list Lambda ARN here (in quotes)>
          }
        }
      },
      "ItemsPath": "$.streetIds",
      "MaxConcurrency": 40
    },
    "Get section list": {
      "Next": "Determine info per section",
      "Retry": [
        {
          "ErrorEquals": [
            "Lambda.ClientExecutionTimeoutException",
            "Lambda.ServiceException",
            "Lambda.AWSLambdaException",
            "Lambda.SdkClientException"
          ],
          "IntervalSeconds": 2,
          "MaxAttempts": 6,
          "BackoffRate": 2
        }
      ],
      "Type": "Task",
      "ResultPath": "$.sectionIds",
      "Resource": <get-section-list Lambda ARN here (in quotes)>
    },
    "Determine info per section": {
      "Type": "Map",
      "ResultPath": null,
      "End": true,
      "Parameters": {
        "sectionId.$": "$$.Map.Item.Value",
        "predictFor.$": "$.predictFor"
      },
      "Iterator": {
        "StartAt": "Determine info",
        "States": {
          "Determine info": {
            "End": true,
            "Retry": [
              {
                "ErrorEquals": [
                  "Lambda.ClientExecutionTimeoutException",
                  "Lambda.ServiceException",
                  "Lambda.AWSLambdaException",
                  "Lambda.SdkClientException"
                ],
                "IntervalSeconds": 2,
                "MaxAttempts": 6,
                "BackoffRate": 2
              }
            ],
            "Type": "Task",
            "Resource": <gdetermine-info Lambda ARN here (in quotes)>
          }
        }
      },
      "ItemsPath": "$.sectionIds",
      "MaxConcurrency": 40
    }
  }
}
```