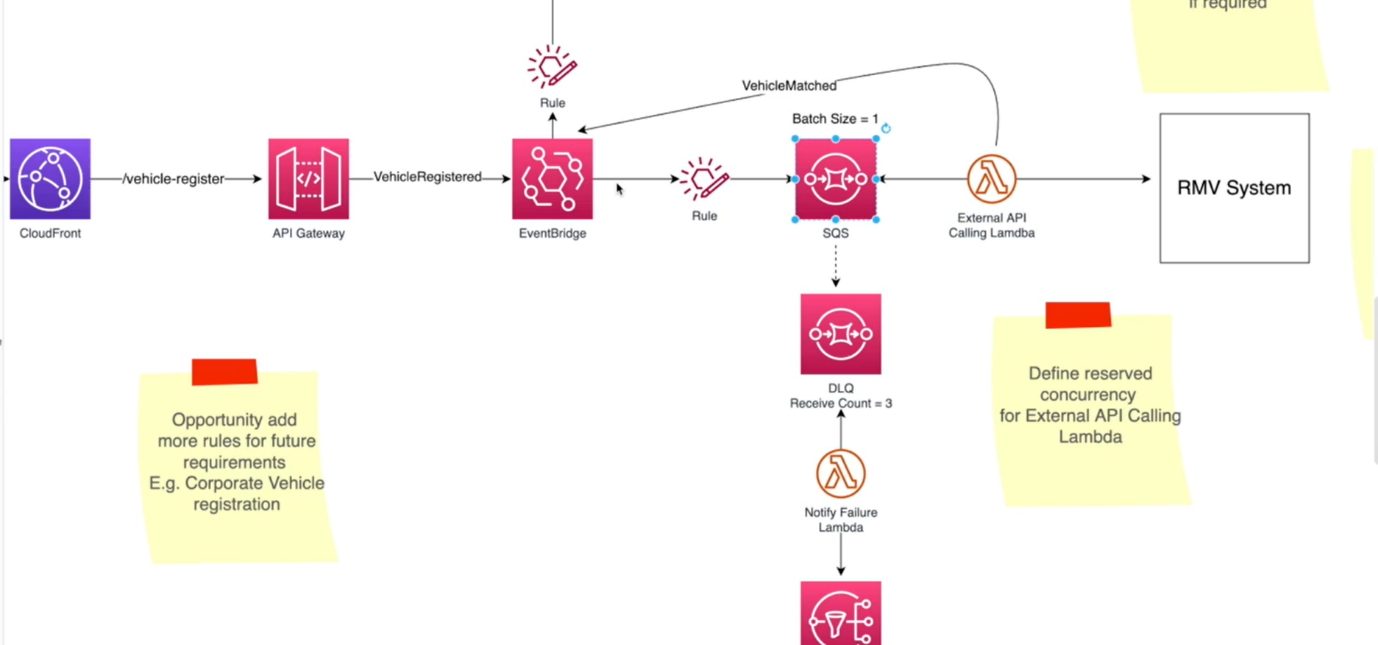
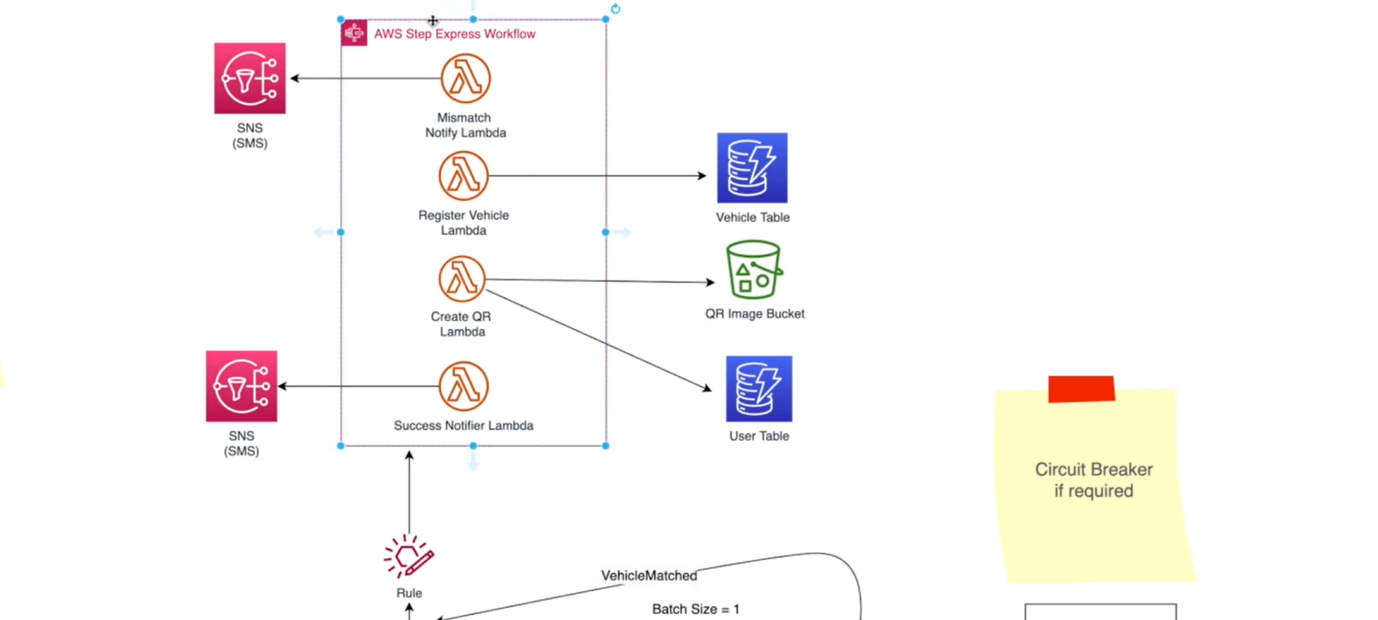
**10. Rate Controlling with Serverless Queue**



* So why did we use SQS?
* Can't we just have a lambda function?
* Usually events are JSON messages, so do not get confused. We can send these events to a lambda target and then this lambda can send all these requests to the backend or the downstream service.
* In this case, the RMV system and do the verification process. but imagine we will receive thousands of vehicle registration requests concurrently. So, when you are designing a system, we must prioritize on concurrent requests, not the number of requests. number of requests can be handled one way or another but handling the concurrency is the challenge.
* So, if we have 1000 concurrent request, then event bridge, we will invoke 1000 lambda functions in
* parallel. Then this RMV system, whatever the third-party system will receive all these requests. So, it will receive massive number of requests at once and this third-party system, we don't know how scalable they are and if we are not designed for scalability, then these systems will surely fail.
* So, we want to stop that. So how can we stop that?
* The easiest way is to introduce a queue. So, once we have a queue, the queue can ingest all this traffic but the workers who will consume the queue or who will be queue the queue can be controlled because these this part, the left side part is at our control.
* This RMV system, the third-party system, we do not have any control. So, we can control whatever at our end. Now here in order to control the number of requests we are sending to the downstream service. I set the bed size to one. Now here we connect SQS with lambda using event source mapping, meaning that there will be managed Lambda Service.
* It will pull the messages from a SQS and send it to these lambda functions so we can set the batch size up to, I think ten.
* So, the lambda function will receive just one message at a time but then again, how it goes is when one lambda function receives one message and if there are more messages in the queue, the lambda will start to scale. At first it will scale up to five lambda functions and then will start consuming one message each and if there are more messages, it will keep on scaling. So, he also even if he set the batch size to one, we must manage the concurrency of this lambda function execution as well.
* Now in the previous episode I discussed about reserved concurrency and here also we can define a reserve concurrency for this external API calling lambda function so that we will control that concurrency of this lambda function as well.
* For example, if he set this concurrency to five at most there'll be only five of these lambda functions executing in parallel and we can adjust this batch size and also the reserved concurrency depending upon how capable is our third party system.
* So in this case, this is just an assumption to set the batch size to one and the reserve concurrency to five. Now one might wonder, hey, if we control this executing part of this lambda function, there will be a lot of messages in flight. In flight, as in there'll be a lot of messages in the queue. So those users will not be served immediately. Is that correct?
* Yes, that is correct. That is why in the requirements I particularly mention, users should receive an SMS once the QR code is ready.
* So, we will remove that synchronous call to receive the QR code as soon as they click register. Instead of that, we will be using an asynchronous flow. So as soon as users click that registration button this button, the call will be received by the API Gateway and the API gateway will add that event or put that event-to-event bridge and then event bridge will send a message. Okay API gateway. I receive your message and that response will be sent to the front end and from the front end we can show a message to the user. Your data is being processed.
* You will be notified once the processing is complete and we will send you an SMS. So, users really do not have to wait to get their QR code immediately, but they will know. Okay, they will eventually receive an SMS and that is SMS will have a link to get their QR code. So, when we are designing scalable system, this type of asynchronous event driven architectures are really helpful to make these systems very scalable.

**RMV system**

* So now let us discuss when this external API calling lambda sends a request to this external system. With the request information like the vehicle details and the user detail and the external system,
* in this case the RMV system will check against their database. If the information is accurate, they will send. Okay, this is a legitimate request. This is a legitimate user and his vehicle information.
* So, at that time, this Lambda will put another event into the event bridge and we can call this event maybe event matched and this event will have a body of JSON and that will include whether this is successful or unsuccessful and this event will also be added to the event bridge and we can write another rule listening to these types of events. Event matched type of events and then that will be routed to a step Functions Express Workflow.



* I will discuss in detail as to why we take this decision but before that, let's discuss a failure situation. Now, although we try to control our traffic to the backend system or the external system, sometimes
* this system can be offline. So, in that case, the Lambda will try to connect and it cannot. It'll try a few times and if it is not successful, then the message will go back to the queue. Now we can define a DLQ or dead letter queue to our SQS.
* The DLQ is nothing but another SQS based queue which we can attach to the original SQS
* So, in this configuration we can say if a message is being received by this many times, then send it DLQ. Now here I have said the receive count or how many times this message has returned to the queue to three and the return count of this message is three. Then it will be added to a DLQ.
* Then we can again attach a lambda function that will consume the message from the DLQ and I call this notified failure lambda. So, what it does is it will send an SMS to the user. The vehicle registration process is not successful and they will receive an SMS once the vehicle registration system is back online.
* So, they know something has gone wrong and they will wait until they will receive an SMS from the system.