

# Parking Rights Technical Design

...

# Architectural Overview



# Assumptions

1. Parking Right service runs in a private VPC.
2. PRDB is an external centralised service.
3. License Plate, OperatorId, Customer Type Id is unique together.
4. Parking registration can be completed in near real time.
5. After payment has been done successfully there is no business level validation required to register a parking spot.

# No server is easier to manage than no server.

- Compute : Lambda
  - Support for dynamic workloads.
  - Easy to deploy, in that case I used SAM.
- Storage : DynamoDb
  - No relational db required.
  - Fast
- Communication : SNS, SQS
  - Provides isolation between services.
  - SNS, able to send an event to more then 1 services.
  - SQS, process messages at services own pace
- Monitoring : Cloudwatch

# About Solution

There are two web api's in the solution running as lambda function behind API Gateway. My goal was to show how would I'd divide the responsibilities into microservices.

- Parking Rights Web API : Microservice that handles the process after a parking spot has been sold successfully. Post and Get functions implemented without any validation layer. I choose DynamoDB to store data because relational db isn't required. Swagger documentation can be found [here](#).
- Parking Registration Integration Web API : Microservice that will communicate with the external PRDB service. It is an isolation layer between 3rd party service and ParkingNow ecosystem.

# Weaknesses

I made some shortcuts for the sake of simplicity;

- Implementation details :
  - Validation
  - Functionality
    - Unique Id is created by concatenation of license-plate, customertype
    - Idempotency
  - Logging
    - It is possible to see the logs in Cloudwatch but a centralised tracing system is essential to be able to have a clear overview of the process.
  - Sync communication (Rest based) to async communication
  - Removing (dummy) credentials to create basic aws auth
  - Testing, only data access layer has been tested.
- Security : I'm using one role for both services. Role/resource based access control.
- CI/CD : Instead of SAM creating a pipeline and separate infrastructure from code.