1.

Events to track:

* Enter to the parking lot
* Exit of the parking lot

Event structure:

* Action: (IN|OUT)
* Parking lot ID
* User ID
* Timestamp

Tech stack:

* MQTT
* Microservices
* DBMS

Analytical tech stack:

* Kafka, Kinesis, Pub/sub
* Spark, Flink, Storm
* Atlas, Data Catalog, AWS Glue Data Catalog
* S3, gs, Azure blob storage
* Delta Lake
* Superset, Tableau, Data Studio

Text, whiteboard

Description automatically generated

2. the backend system could be designed based on spring-based microservices, running on a Kubernetes cluster with autoscaling process to adjust the amount of resources based on the amount of events during the day.

The event should contain only a minimal number of fields to reduce the processing time on the client side (parking lot tolls, etc.) and this information could be extended on the backend side, using the previously collected data (user information, credit card, etc.) in order to complete a full service on the parking lot without any other interaction from the client rather than scanning the QR code or read NFC tag.

3. As a good microservice practice, there should be a audit log on every critical step of the process, and that log is the perfect candidate to be ingested by the analytical process without interfering the operational infrastructure.

There are different ways to do this like, having the DBMS bin logs or message interception between microservices, all of these are sent to an event stream platform who serves the data to the analytical architecture, and this should post the generated models for the business and should be able to generate insights based on the business activity to promote new ideas based on data.

4. to automate the whole process, I suggest to use a flexible and dynamic operation of new features, based on CI/CD, across 3 environments (dev, qa, staging) and scale the amount of tests based on the scenario (on dev we should be able to guarantee the stability of the previous and actual code through unit and manual tests; qa should be able to run automatic and manual tests to guarantee a functional execution and data quality in place; staging should be able to involve the user and the business in the platform tests to see everything is running according their expectations), all of these environments have a conditional pass-trough to guarantee each team has the correct feedback across the entire process and can act according to that information to serve solution at each stage. The tech stack for this kind of process should also be dynamic to correctly support any case scenario that could not be handled by an standard platform (generating more support work on the platform than the actual work on test scenarios).

Finally there a final gate to update the actual production environment with the latest components (usually marked as a manual step), this gate also generates feedback about the update process and some update execution paradigms (most of then comes from the microservices paradigm, like blue/red or canary).