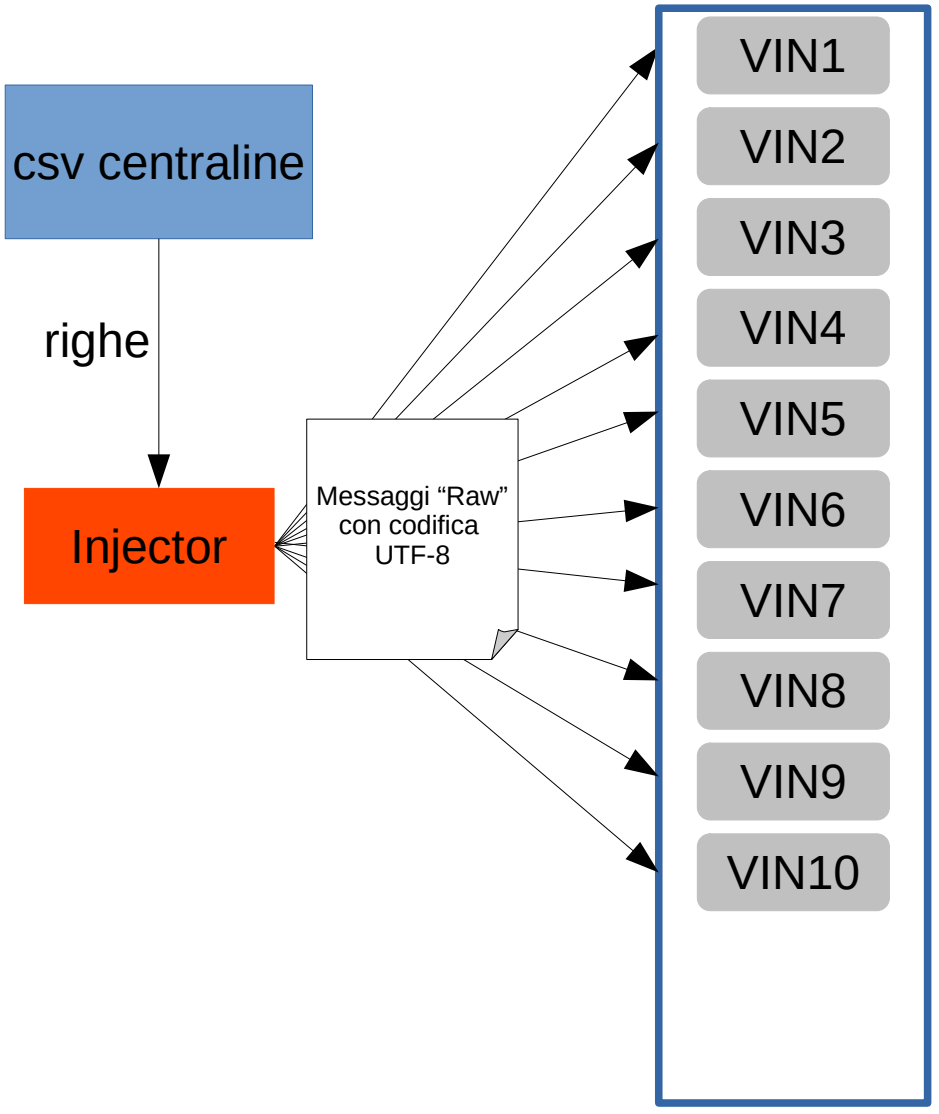
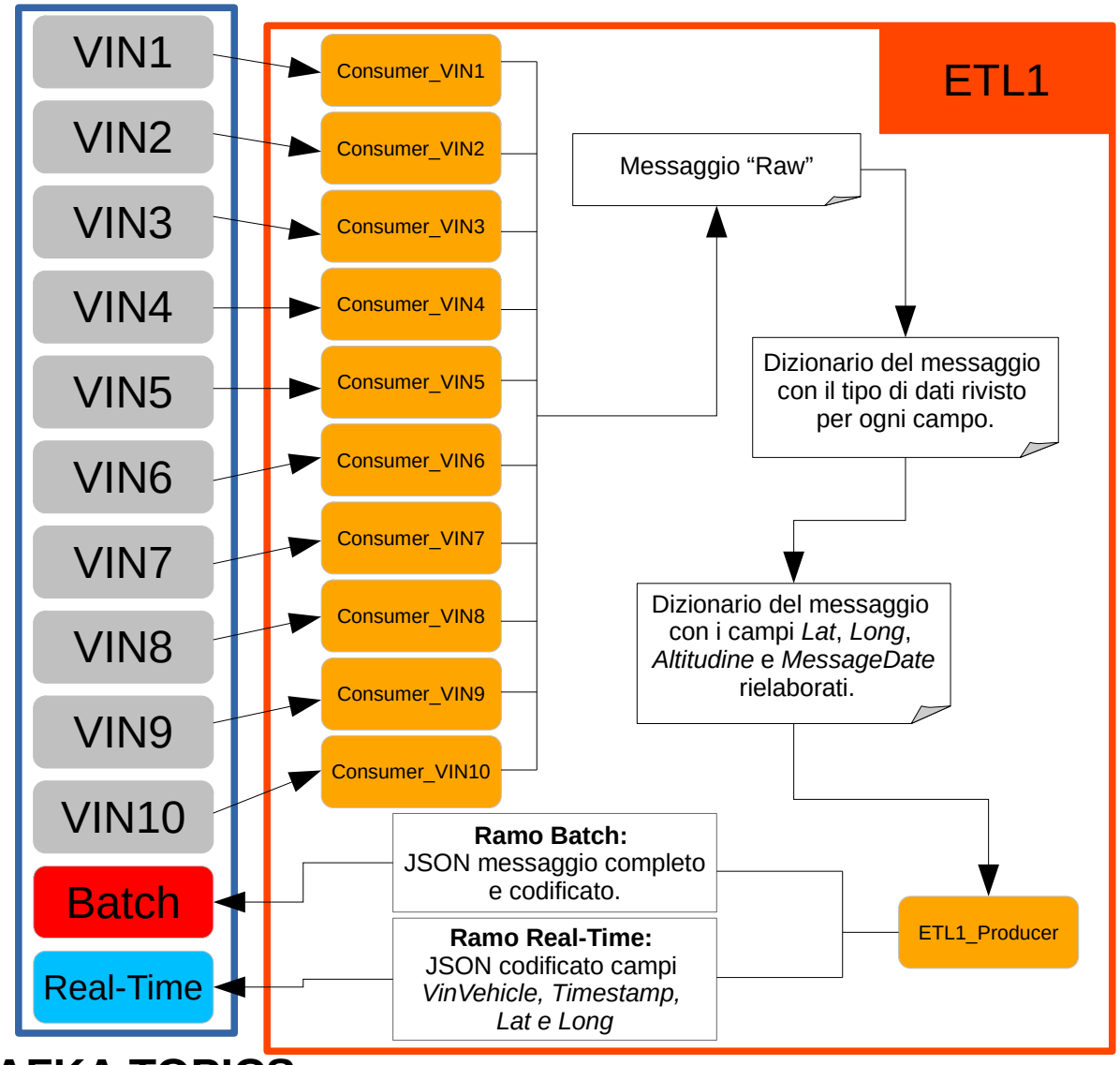


Microservizio INJECTOR

Input:
File .csv dati centraline

Elaborazione Dato:
Il dataset viene pulito dai Nan e dai duplicati.
I nomi delle colonne del dataset vengono cambiati per eliminare caratteri che possono dar problemi lungo lo stream (es. “.” sostituito con “_”).
I dati ricavati vengono trattati riga per riga.
Ogni riga è trattata singolarmente come un messaggio singolo della centralina del veicolo con VIN “X”.
Il messaggio viene codificato con codifica UTF-8 e dato al producer che lo pubblica sul topic relativo al VIN del messaggio.

Output:
Il messaggio “raw” codificato utf-8 è pubblicato
sul topic relativo al VIN number della riga in analisi.
Un producer pubblicherà il messaggio codificato sul topic corrispondente.
Si avranno n topics con n uguale al numero totale di VIN nel dataset (10 VIN).



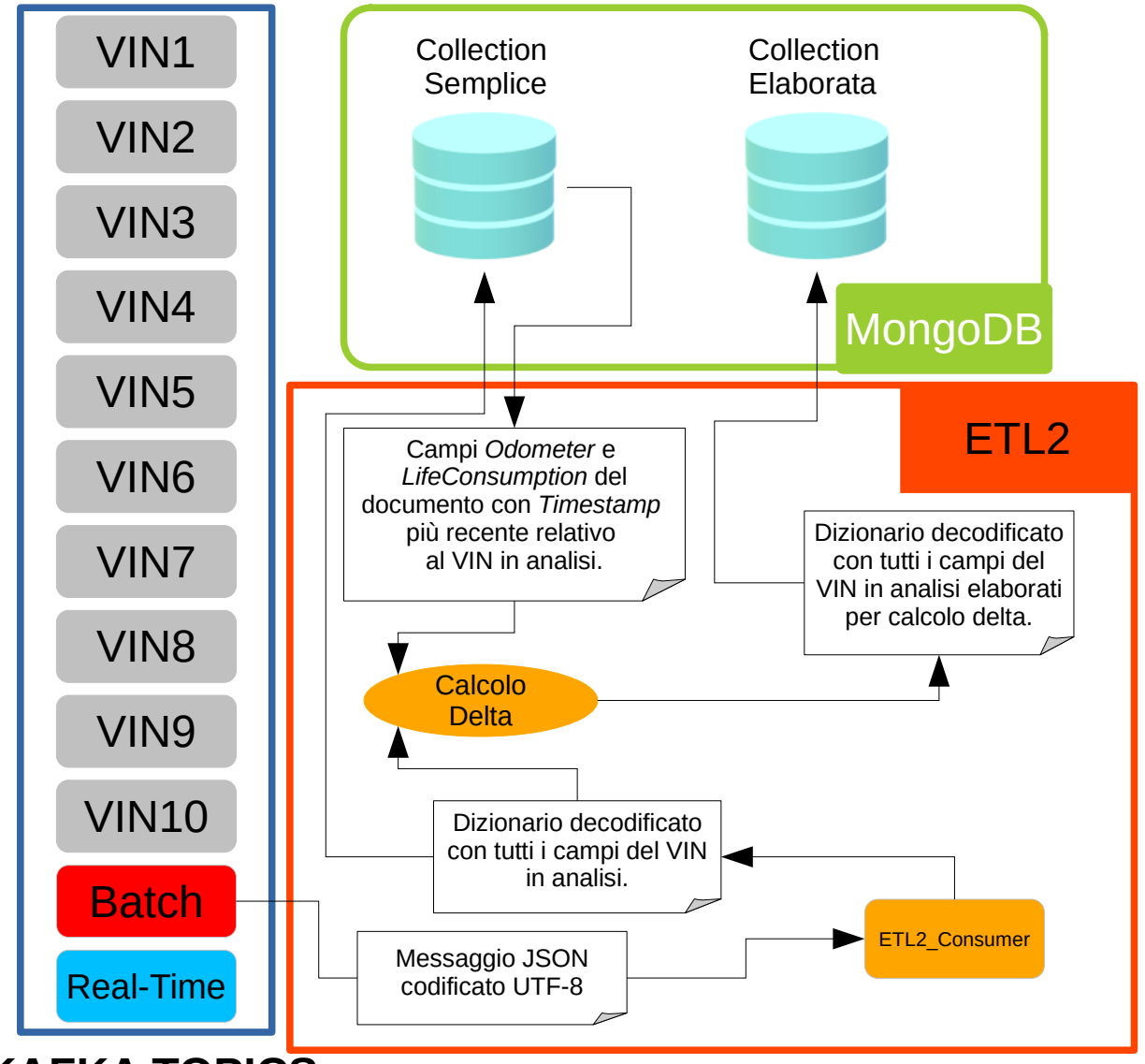
Microservizio ETL1

Input:
10 consumers (1 per VIN number) prelevano iterativamente i messaggi “raw” dal topic corrispondente pubblicati in precedenza dal producer del microservizio Injector.

Elaborazione Dati:
I messaggi ottenuti vengono elaborati in modo da decodificarli in stringa e riformattarli in un oggetto dizionario. Il tipo di dato dei campi viene riadattato in accordo alle specifiche del progetto.
I campi *Lat* e *Long* vengono invalidati a -1 se il numero di satelliti è inferiore a 3.
Il campo *Altitude* è settato a 0 se questo è minore di 0.
Viene aggiunto un campo *MessageDate* che altro non è che il *Timestamp* trasformato in formato ISO-8601.
Il dizionario (messaggio) viene trasformato in JSON e codificato UTF-8 dal producer ETL1 sia per il ramo *Batch* che per quello *Real-time*.

Output:
<u>- Ramo Batch:</u>
Il dato in formato JSON, completo di tutti i campi rivisitati viene inviato sul ramo *batch* dal producer che lo pubblica per intero sul topic *“Batch”* di Kafka.

<u>- Ramo Real-time:</u>
Il dato in formato JSON, relativo ai soli campi *VinVehicle*, *Timestamp* (questi due per avere una chiave univoca per l’ETL3 succesivamente), *Lat* e *Long* viene pubblicato dal producer sul topic *“Real-Time”* di Kafka.

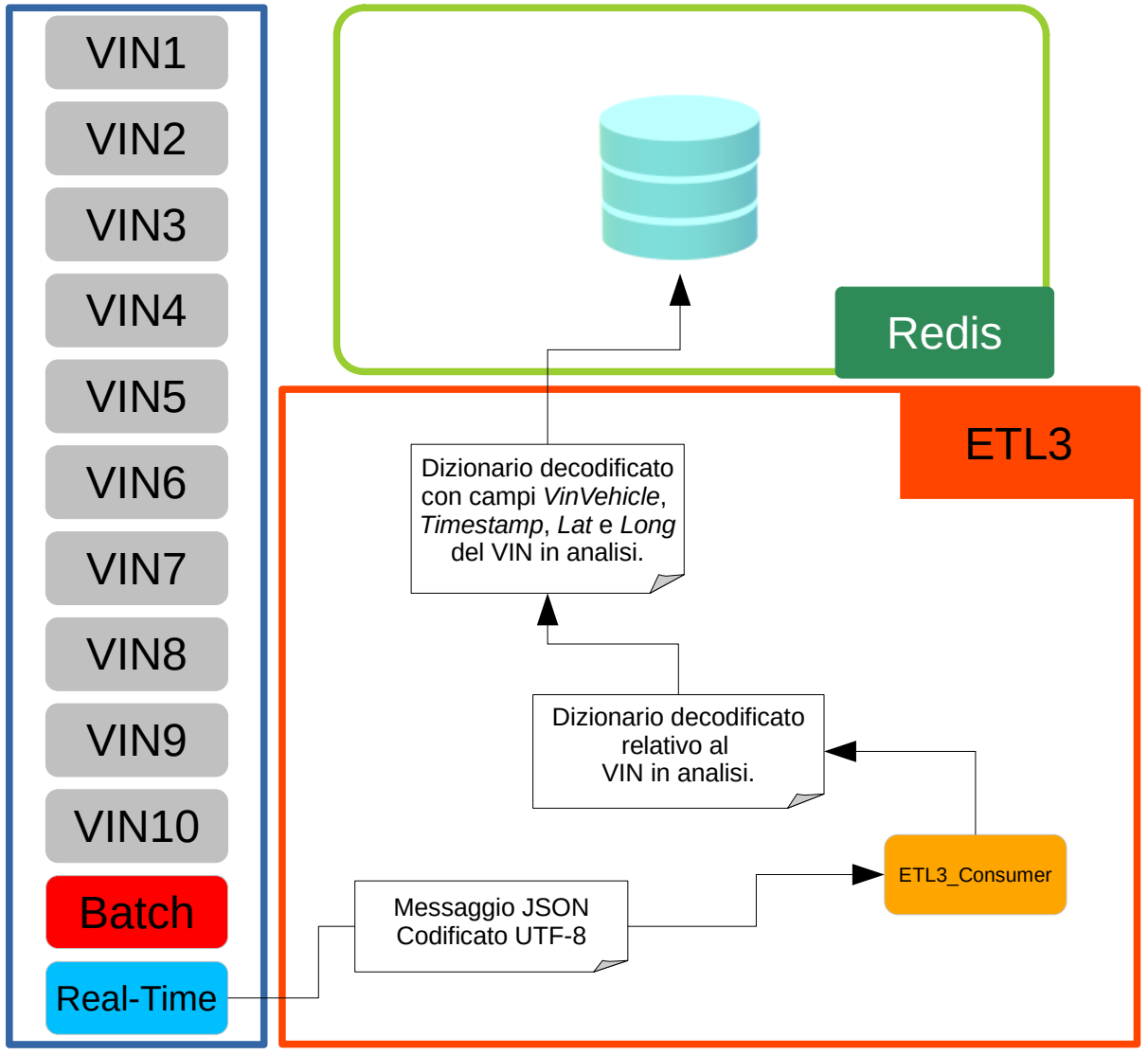


Microservizio ETL2

Input:
Il microservizio ETL2 prende come input i messaggi pubblicati nel topic *Batch* tramite il consumer ETL2.

Elaborazione Dati:
Il consumer ETL2 al momento della ricezione del messaggio dal topic *Batch* lo decodifica e lo trasforma in un dizionario Python e ne ricava il VIN.
L'ETL2 preleva il documento più recente del VIN in analisi dalla *Collection Semplice* di MongoDB e lo salva su un altro dizionario.
L'ETL2 invia il messaggio ottenuto dal consumer sulla collection semplice di MongoDB tramite il comando update/upsert con chiave i campi *VinVehicle* e *Timestamp* per evitare duplicati.
Usa il dizionario appena inviato a MongoDB e quello ottenuto con la query precedente per il calcolo dei campi *DeltaOdometer* e *DeltaLifeConsumption*.
Elimina i campi *Odometer* e *LifeConsumption* sostituendoli con i due delta ammesso che sia stato possibile calcolarli, altrimenti li setta a 0 (questo nel caso del primo messaggio del VIN sul database).
Pubblica il dizionario con i due Delta sulla C*ollection Elaborata* di MongoDB.

Output:
Le due collections aggiornate su MongoDB:
- Semplice = completa di tutti i campi "originali"
- Elaborata = con i due campi "Delta".



KAFKA TOPICS

Microservizio ETL3

Input:
Il microservizio ETL3 prende come input i messaggi pubblicati nel topic *Real-Time* tramite il consumer ETL3.

Elaborazione Dati:
Il consumer ETL3 al momento della ricezione del messaggio dal topic *Real-Time* lo decodifica e lo trasforma in un dizionario Python e ne ricava il VIN ed il Timestamp.
L'ETL3 pubblica il dizionario su *Redis* usando come chiave univoca per il record da inviare, una stringa formata dalla concatenazione del VIN e del Timestamp.

Output:
Database Redis aggiornato con i dati del topic *Real-time.*

WORKFLOW

Passo 1) Avviare Containers/servizi:

A) ***Kafka*** e ***Zookeeper***: il client del microservizio comunicherà con *localhost:9092*
B) ***MongoDB***: il client del microservizio comunicherà con *localhost:27017*
C) ***Redis***: il client del microservizio comunicherà con *localhost:6379*

Passo 2) Avviare un IDE come VisualStudio Code

A) La cartella del progetto avrà la seguente struttura:

+ Esame_parte_ETL
    - dati_centraline.csv
    - injector.py
    - etl1.py
    - etl2.py
    - etl3.py

B) Aprire VScode nella cartela “Esame_parte_ETL”

Passo 3) Avviare in maniera sequenziale i microservizi tramite terminale di VScode.

Avviare i microservizi con questo flow: injector → etl1 → etl2 → etl3.
Importante aspettare la fine delle computazioni di ogni microservizio prima di passare a quello successivo.
E’ possibile visualizzare i risultati dei microservizi con strumenti GUI come Kafkatool, MongoDB_Compass e RDM (per Redis).