Computing Systems and Infrastructure
LIACD | 2025-2026

Assignment 02 - Resilient Telemetry for Lunar Exploration Infrastructure

In the evolving landscape of Interplanetary Internets and the Cloud-to-Deep-Space continuum, the reliability of data transmission between surface assets (rovers) and orbital stations (gateways) is paramount. Unlike terrestrial datacenters where latency is negligible, extraterrestrial networks face high Bit Error Rates (BER), intermittent connectivity, and harsh environmental variables.

For this assignment, students are tasked with engineering a containerized architecture that simulates the telemetry pipeline of a Lunar Surface Rover communicating with a Gateway Orbiter. The objective is to deploy a resilient ingestion pipeline that decouples data generation (sensors) from data persistence (database) via a message broker, ensuring that critical operational data (ranging from life-support metrics to spectral analysis) is buffered, processed, and visualized in near real-time.

System Architecture

The solution requires the orchestration (`docker-compose.yaml`) of a Dockerized environment simulating a "Systems-of-Systems" approach. The pipeline is defined as follows: `Producers (Python/Simulators)` → `Message Broker (Kafka KRaft)` → `Ingestion Agent (Telegraf)` → `Time Series Database (InfluxDB)` → `Visualization & Alerting`. Figure 1 depicts the assignment system architecture.

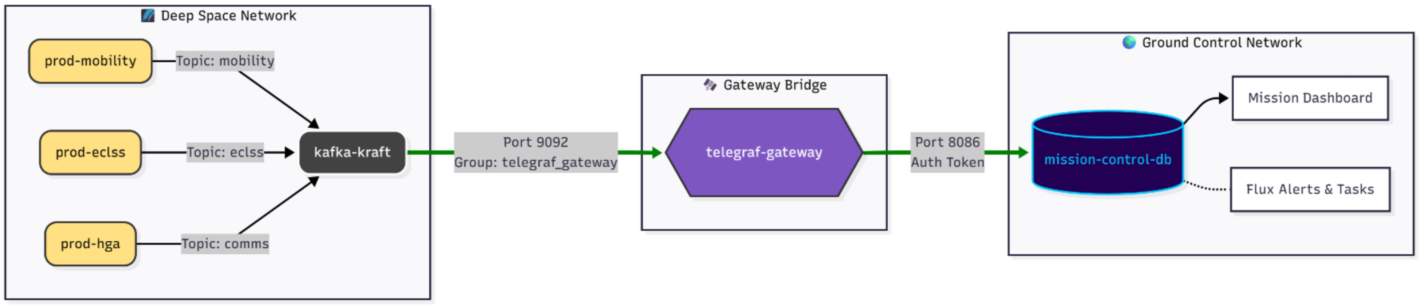

Figure 1. Assignment System Architecture

Students **must** implement <u>network isolation</u> to simulate the *air gap* between the operational technology (the rover) and the IT infrastructure (the database). Consequently, the solution must utilize two distinct Docker networks: `deep-space-net` (connecting producers to `Kafka`) and `ground-control-net` (connecting `Telegraf` to `InfluxDB`), with `Kafka` and `Telegraf` acting as the bridge services.

Service Implementation Guidelines

The Message Broke. Reflecting the industry shift toward simplified infrastructure management, this assignment requires the deployment of Apache Kafka in KRaft mode, eliminating the Zookeeper dependency. The Kafka service must be configured using the `bitnamilegacy/kafka:latest` image. The environment variables must establish the node as both a broker and a controller. Students must configure `KAFKA_CFG_PROCESS_ROLES=broker,controller`, set the `KAFKA_CFG_NODE_ID`, and define the listener security protocols to allow distinct internal (client) and controller traffic. The persistent volume `kafka-data` is required to ensure message durability in the event of container restarts, simulating the resilience required in fault-intolerant environments.

Telemetry Emulators (Python Producers). Students must develop three distinct Python-based containerized applications. These scripts simulate the sensor arrays of the rover and the orbital link. These producers must simulate *burst* transmission behavior (sending batches of data) to mimic limited transmission windows common in satellite links, rather than simple periodic sleep cycles.

- Producer A: Rover Mobility & Power Unit
  - Data: Battery Voltage (*V*), Motor RPM, and Wheel Traction (`0.0 to 1.0`).
  - Behavior: Voltage should decay over time and recharge cyclically. Traction should exhibit random drops to simulate regolith slippage.
  - Topic: `telemetry-mobility`.
- Producer B: Environmental & Life Support (ECLSS)
- Data: External Temperature (Celsius, ranging `-170 to +120`), Radiation Levels (`µSv/h` ranging `0.06 to 0.3` -normal-; `2.1 to 2.8` -commercial aircraft flights-; `5.0 to 9.4` -Chernobyl-), and Internal Cabin Pressure (`kPa`).
  - Topic: `telemetry-eclss`.
- Producer C: High-Gain Antenna (HGA) Diagnostics
  - Data: Signal-to-Noise Ratio (`dB`), Bit Error Rate (`BER`), and Latency (`ms`).
  - Behavior: Values must simulate signal degradation (high BER, low SNR) intermittently.
  - Topic: `telemetry-comms`.

All scripts must utilize the `kafka-python` library and serialize messages into the InfluxDB Line Protocol before transmission. All the scripts must share the same `Dockerfile` and set the `entrypoint` accordingly in the compose file based on the producer. The values produced by the producers must be compatible with the components/modules being simulated.

Ingestion Agent (Telegrapf). It functions as the sole link between the isolated `deep-space-net` and the `ground-control-net`, ensuring strict network separation. When configured with the `inputs.kafka_consumer` plugin, it subscribes to telemetry topics (`telemetry-mobility, telemetry-eclss, telemetry-comms`) provided by the

Kafka broker. It effectively batches this high-frequency data stream, enforces InfluxDB Line Protocol serialization, and securely transmits the data to the InfluxDB storage layer via the `outputs.influxdb_v2` plugin, thus separating data collection from storage infrastructure. To ensure proper interaction between mission control and this component, the `hostname` for the ingestion agent in InfluxDB **should be** set to `lunar-gateway`.

Persistence and Visualization (InfluxDB). The database container (`influxdb:latest`) serves as the Ground Control archive. Setup requires initializing a specific organization (`org: esa-sic`) and a bucket (`bucket: lunar-mission`).

- Dashboard Requirements. Students must design a *Mission Control* dashboard. Be creative during this task and put your personal touch on the data that will be visualized. To differentiate from standard list views, the dashboard must utilize:
  - Gauge Indicators. For real-time Critical Life Support (Cabin Pressure).
  - Heatmaps. To visualize the correlation between Motor RPM and Wheel Traction over time.
  - Stat Single Metrics. Displaying current maximum Radiation levels.
  - Graph with Thresholds. Visualizing Battery Voltage with a visible red line indicating the *Critical Discharge* level.
  - Your dashboard should be exported to a file (`dashboard.json`) and submitted inside the dashboard folder of your assignment.

Alerts & Processing. While the dashboard provides visual monitoring, *Ground Control* requires an automated alerting system to detect anomalies immediately. You must implement these logic checks using the `Flux` scripting language within InfluxDB.

- For each of the producers developed during the assignment, students must define a `.flux` alert that should monitor the data for `Ok, INFO, WARN, and CRIT` values.
- The `.flux` files (`alert_eclss.flux, alert_hga.flux, alert_mobility.flux`) **must** be submitted inside the flux folder of your assignment.

General Guidelines

General Architecture & Workflow:

- Self-Contained Submission. Ensure the Docker Compose file is completely self-contained. It must not depend on pre-existing custom images, volumes, or networks external to the `compose.yaml` file.
- Relative Paths. Never use absolute file paths (e.g., `C:/Users/David/...`) in your volumes or configuration files. You must strictly use relative paths (e.g., `./telegraf/telegraf.conf`) to ensure the project runs on any machine.
- Environment Variables. Use a `.env` file to manage configuration values (such as InfluxDB tokens and credentials) rather than hardcoding them into Python scripts or Compose files.

- Work Plan. Establish a clear work plan early. Do not attempt to integrate the entire stack at the last minute; validate the "Producers to Kafka" link first, then "Kafka to Telegraf".

Docker & Orchestration:

- Official Images. Rely on official Docker images rather than creating custom base images unless necessary.
- Volume Persistence. You must explicitly define volumes for Kafka and InfluxDB ensure data persistence across container restarts.
- Health Checks. Implement health checks for critical services.
- Network Isolation: You must implement network segregation. The solution requires at least two distinct networks (i.e., `deep-space-net` and `ground-control-net`).

Kafka Configuration (Kraft Mode):

- Correct Image Tag. Use the `bitnamilegacy/kafka:latest` image as specified in the lab documentation.
- Listeners Configuration. Pay close attention to `KAFKA_CFG_LISTENERS` and `KAFKA_CFG_ADVERTISED_LISTENERS`. The internal Docker network communication often fails if the advertised listener does not match the service name defined in Docker Compose.
- Avoid Zookeeper. Do not include a Zookeeper container. This assignment uses the modern KRaft mode (Kafka Raft Metadata), where the Kafka node acts as both broker and controller.

Data Ingestion & Processing:

- Line Protocol. Ensure your Python producers format the data string strictly according to the InfluxDB Line Protocol. Incorrect spacing or data types will cause Telegraf to drop messages silently.

What to Avoid (Common Pitfalls):

- Network Flatness. Avoid putting all containers on the default bridge network. You must demonstrate isolation by ensuring the Database cannot see the Producers directly.
- Missing Dependencies. Do not assume containers start instantly. If your Python script crashes with "NoBrokersAvailable," it is likely because it tried to connect before Kafka was ready. Use `depends_on` with `condition: service_healthy`.
- Hardcoded Tokens. Avoid hardcoding the InfluxDB Admin Token inside the `telegraf.conf`; pass it as an environment variable ($INFLUX_TOKEN) for better security and portability.

To consider for your solution:

- You should be familiar with the implementation details of your solution. You will be asked to answer from an engineering perspective during the defense, including making changes to your solution to respond to new requirements.

- You should follow the guidelines provided as well as the best practices for your solution. Any unjustified behavior of your solution will be penalized.
- The defense will have two moments:
    1. Your solution must be deployed outside your personal computer and function without further configuration. Any student group that does not meet this requirement will receive a **grade of 0.**
    2. For visualization, alert, and processing, the student should have at least **30 minutes** of data to better confirm the dashboard and alerts in the system. For this moment, the student should use their computer with the same dashboard and `.flux` files submitted to the system. If during the defense it is detected that these files are different without a proper justification, the group will receive a **grade of 0.**
- Utilize the file structure provided to develop your solution to prevent further complications.

Submission and Evaluation

- The assignment should be completed in pairs of two students. Individual projects will only be accepted in justified and professor-approved cases.
- Submit a single `.zip` file to InforEstudante containing the complete project structure, including all the necessary files.
- **Enroll in one of the available defense time slots on InforEstudante**.
- **Submission Deadline:** 14/12/2025