Cynamics

Cynamics - Network Data Processing Pipeline

Please read the instructions carefully. This exercise emphasizes code design, OOP, and modularity, as well as clean code.

Project Intro

We need to create a pipeline that processes various network traffic samples (normalized flows) into the system, in **batches**. The pipeline consists of multiple steps. A batch is fully processed by each step and only then passed to the next step.

Requirements and components

1. Input and Parsing

The input and parsing component should support receiving data from various input sources (Be it reading a CSV/JSON/Parquet file, or receiving on a socket, etc..).
Once data is received, it should be parsed into a strong, defined data model to be used throughout the pipeline.
Pay attention to modularity and extensibility.
The component can be configured in initialization to a single input source.
****Note: Not every entry in the data contains all fields***

2. Enrichment

Once data is modeled, it then needs to go through an enrichment component to enrich the data with additional fields (SRC and DST separately):

1. Subnet Class A (/8) (e.g. 1.2.3.4/8 -> 1.0.0.0)
2. Subnet Class B (/16) (e.g. 1.2.3.4/16 -> 1.2.0.0)
3. Subnet Class C (/24) (e.g. 1.2.3.4/24 -> 1.2.3.0)
4. Processing Timestamp (ISODate string/unix timestamp of the processing start time)

3. Filtering

Once data is enriched, it needs to be filtered through a chain of simple rules, in order.
If data matches a rule, it is applied immediately and rule-processing is stopped. If it does not match, it continues to the next rule, in order (see example below).
****Note: rules are applicable per-entry, not as a batch***

Rules can:

- ALLOW or DENY
- Use data fields from a preconfigured, static list of fields (see below)
- Filter using a single value (e.g. protocol == 17)
- Filter using a list of values (e.g. dst port in [22,23,443])
- Filter fields using a wildcard by default (any value matches)

The default rule is ALLOW ALL. Denied data is dropped.

4. Output and Publishing

Once data is through the filter, it then goes through a configurable output and publishing component that can forward the data to multiple destinations (HTTP request, TCP Socket, JSON/CSV File, etc..), **one batch at a time**.
The component can be configured at initialization with its destination configurations.

Cynamics

Coding Instructions

Note that alongside this document is attached a sample_data.json file that contains data you can use to simulate and run your code.

1. Language - Any language would do, we prefer Python.
2. Best practices - Use the best practices in whatever language you choose. Emphasize code quality, readability, project and code structure, etc..
3. Implementation notes:
    a. Pipeline - Note this project is meant to be a pipeline. Code as if future steps will be added or some existing ones will be removed.
    b. Parsing - **Implement support for JSON only. No need to implement support for other formats. Make sure your code will support new formats with little to no change.**
    c. Enrichment - Implement all new fields required. **Note some fields should be duplicated per side (SRC, DST)**
    d. Filtering - **Emphasize modularity and pay attention to how rules are defined and evaluated.**
    e. Publishing - **Only print the data to console (plain json format). Make sure your code will support new formats and destinations with little to no change. You can print the entire batch at once.**

4. Add a main file that will initialize your components and read the sample data.
5. Under the Input and Output components, implement only what is necessary to run your code end to end. The focus in these components needs to be on modularity and future compatibility.
6. Rules - Implement the given examples in this document.
7. Filtering component - Here we'd like to see modularity in your design, to enable much more complex rules in future features. These can vary, here are some future examples:
    a. Support multiple filters in a single rule (e.g. dst port == 22 AND protocol == 6)
    b. Filter by range (src port between 100-120)
    c. Bidirectional traffic by IP and Port (e.g. src port in [22, 23] OR dst port in [22, 23])
    d. Generic support for new fields, types, and operators
8. **No need to implement unit tests. Do implement a single assert of the pipeline output.**

This exercise focuses on modularity, extensibility, and clean code.
Send the code as a .zip file to your interviewer

Cynamics

Rules

Data Fields

Allow filtering for these fields only:

- SRC IP
- DST IP
- SRC port
- DST port
- Protocol
- SRC Subnet class A
- SRC Subnet class B
- SRC Subnet class C
- DST Subnet class A
- DST Subnet class B
- DST Subnet class C

Examples (Implement these)

1. ALLOW: Src Subnet Class A == 10.0.0.0
2. ALLOW: Dst Subnet Class A == 10.0.0.0
3. ALLOW: Protocol in [1, 17] AND srcPort == 123
4. DENY: Dst port == 443
5. DENY: Dst port == 80
6. ALLOW ALL(*) - **default**

Partial data examples:

1. Src IP: 10.4.3.2, Dst port: 443 -> Allowed by the rule #1
2. Dst IP: 17.0.0.1, Src port: 123, Protocol: 17 -> Allowed by rule #3
3. Src IP: 17.0.0.1, Dst port: 443, Protocol: 6 -> Denied by rule #4
4. Dst IP: 17.0.0.1, Dst port: 80, Protocol: 17 -> Denied by rule #5
5. Dst IP: 17.0.0.1, Dst port: 22, Protocol: 6 -> Allowed by rule #7 (default)