Initial Questions

Some high level questions to be answered as a seperate document

Questions:

1. **Identifying Improvements:**
    - How do you systematically identify areas for improvement in a microservice-based system?
    - What tools and methods do you use to gather data and insights about the system's performance, reliability, and user experience
2. **Prioritizing Improvements:**
    - Once potential improvements are identified, how do you prioritize which ones to address first?
    - What factors influence your decision-making process when prioritizing improvements?
3. **Implementing Improvements:**
    - Describe your approach to implementing improvements in a live microservice-based system.
    - How do you ensure that changes are thoroughly tested and do not introduce new issues or regressions?
4. **When to Improve:**
    - How do you determine the right time to make improvements to the system?
    - What indicators or metrics do you monitor to decide when an improvement is necessary or urgent?
5. **Continuous Improvement:**
    - What strategies do you recommend for fostering a culture of continuous improvement within the development team?
    - How do you balance the need for new feature development with the need for ongoing system improvements?
6. **Personnel Management:**
    - How do you manage and mentor other developers in a microservice-based project to ensure high performance and continuous improvement?
    - What approaches do you take to handle conflicts, encourage collaboration, and maintain a positive team dynamic?
    - How do you ensure that all team members are aligned with the project's goals and have the necessary skills and resources to contribute effectively?

Technical Test

To be presented to use in the form of a Github repository. Challenges are open to all senior and mid level candidates to get a good understanding of technical ability and approach. If uncomfortable with Challenge 1 then Challenge 2 can be attempted as a standalone task. If attempting both then challenge 2 should be a branch off challenge 1 so that we can diff the changes.

Challenge 1

We need to download data from Sepolia as efficiently as possible as part of an L2 project being developed. The contract in question is 0x761d53b47334bee6612c0bd1467fb881435375b2. We need to query for this topic 0x3e54d0825ed78523037d00a81759237eb436ce774bd546993ee67a1b67b6e766 and store the L1 info root data along with the block time and parent hash of the block associated with the log. You can use any key value store i.e. Bolt, LevelDB to hold this information. We need to have a single entry for each event keyed by an incrementing index starting at 0.

Caveats:

- some blocks will contain multiple events so we must increment the index in this case based on the log index within the block
- consider efficient serialisation/deserialisation of the data we need to store we should have a type to represent the data when retrieved from the DB
- The solution should be written in Go

Challenge 2

To help with the above challenge we need to have an efficient way to make use of multiple RPC endpoints so that we can spread the load out and download the data as quickly as possible. Come up with an algorithm that spreads the load out effectively.

Caveats:

- The fastest RPC nodes should be allowed to respond quickly, we do not want to be constrained by the slowest endpoint

Challenge 3

- take an issue from github repo https://github.com/0xPolygonHermez/cdk-erigon with the tag "challenge" and implement it.