Log Collection

Hello! Congratulations on making it to this round of Cribl's interview process! The objective of this take home exercise is to demonstrate your design and implementation techniques for a real-world type of development problem. You can use any programming language or tech stack of your choice. However, please note that we are looking for solutions that **demonstrate your unique code**, so please limit external dependencies to a bare minimum.

We know that taking on a code submission assignment like this one is not a trivial request, and we really appreciate the time you put into this - it really helps us get to know you and how you work! We don't want you spending an inordinate amount of your time on it, though, so if you find that it's taking more than ~4 hours to complete, please reach out to us and we can help make sure you're pointed in the right direction.

You should develop and submit code to Cribl via a GitHub project. Please commit and push code changes as you normally would - your thinking and working style is an important part for us to understand. In your submission, please include a link to the public GitHub repo containing your code.

Of course, no great product is complete without clear documentation and testing. As part of your solution, please provide any **design, usage, and testing documentation** that you feel would be helpful to someone using your solution for the first time.

Problem statement:

A customer has asked you for a way to provide on-demand monitoring of various unix-based servers without having to log into each individual machine and opening up the log files found in /var/log. The customer has asked for the ability to issue a REST request to a machine in order to retrieve logs from /var/log on the machine receiving the REST request.

Acceptance criteria:

1. A README file describing how to run and use the service.
2. An HTTP REST API exposing at least one endpoint that can return the lines requested from a given log file.
3. The lines returned must be presented with the newest log events first. It is safe to assume that log files will be written with newest events at the end of the file.
4. The REST API should support additional query parameters which include
    1. The ability to specify **a filename** within /var/log
    2. The ability to specify the last **n number of entries** to retrieve within the log
    3. The ability to filter results based on basic **text/keyword matches**
5. The service should work and be reasonable performant when requesting files of >1GB

6. Minimize the number of external dependencies in the business logic code path. For example, if implementing your project with Node.js:
    1. Feel free to use Express or similar as the HTTP server as well as any of the built-in Node.js modules like *fs*.
    2. Please **do not** use external libraries for any file reads or working with the log lines after you've read them. We want to see your solution in this case using only what Node.js has built-in.

Bonus points:

If you finish the project early and want to add a little bit of polish, feel free to implement a basic UI to interact with the API. This can make it a bit easier for you to demo your project to us when the time comes, but isn't required for a complete submission (we're just as happy using curl/Postman/etc. to interact with the API).

Only if you really want to:

If you finish the assignment early and want to challenge yourself with a bit more, there's one additional feature you could add. Working on this part in no way affects the scoring of the project, but if you decide you want to do it anyway, it may lead to some interesting discussions during the next interview phase.

Add the ability to issue a REST request to one "primary" server which subsequently requests those logs from a list of "secondary" servers. There aren't any hard requirements for the protocol used between the primary and secondary servers, and the architecture is completely up to you.