Arguments for CNI Long lived Daemon (gRPC)

- **Pros**
    - The ability to apply network related configuration at the node level in addition to pod level configuration (e.g. ip-masq-agent)
    - Storing and sharing state for clean-up purposes. Today this is done for each plugin which means each plugin must roll its own even though the state could be shared causing duplication.
    - Plugin chaining is brittle and difficult to take actions on failure, no self-healing. For example, delete operations that fail to clean up, or retries due to some temporary failure.
    - Protos explicitly define the interface.
    - It allows flexibility to change the network configuration dynamically.
        - E.g. growing/shrinking network subnet mask lengths
    - Changes to the plugin are less likely to require changes to the client side and decouple the implementation with the specification.
    - Plugin chaining can be more flexible allowing circular dependencies or any-order deletion. Today deletion is implied to be the reverse of the add operation chaining order.
    - Solve bootstrapping the CNI configuration, since that can be coupled with the long running daemon instead of a separate process that just does this operation.
    - Easier plugin installation, no writable FS required
    - Easier distribution of secrets to drivers (through existing orchestration mechanisms like kube secrets)
    - Consistency with evolving CSI spec and Kubernetes device driver spec, similar in form to Docker drivers
    - More honest about the overhead incurred running drivers
    - More transparent to orchestration systems
    - Most non-trivial network plugins already have a long-running daemon anyway (eg, to watch for NetworkPolicy changes)
- Cons
    - Must have a babysitter process to ensure the daemon stays up
    - Probably not identical to Docker drivers
    - More persistent overhead