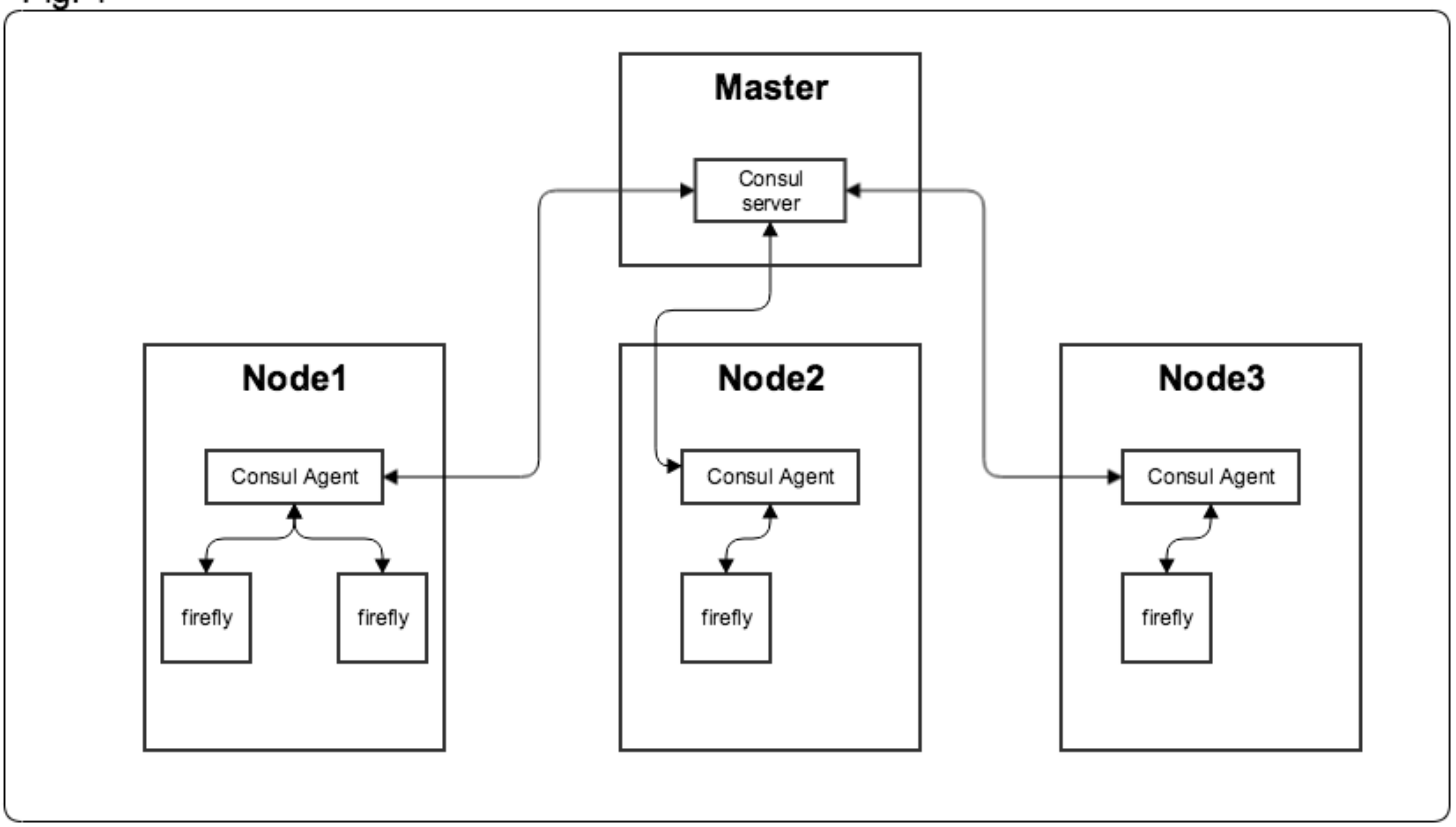
In the first diagram, is a conceptual design of the auto-discovery using Consul. Consul is required to be installed on every Kubernetes nodes with one of them designate as 'server'. The rest are 'agents'. Consul share its states between server and agents. When firefly register itself with its agent, that information is shared with the server and other agents. This allow firefly to query for other firefly instances within the same cluster.

In this design, firefly would register itself with its Consul Agent via a RESTful API with information like IP, port, and a way to do 'health check'. With these information, Consult will maintain a list of all active firefly instances running in the cluster. Periodically, firefly would query the Agent for the active list of firefly instances and establish TCP cache replication with all of them.

Fig. 1

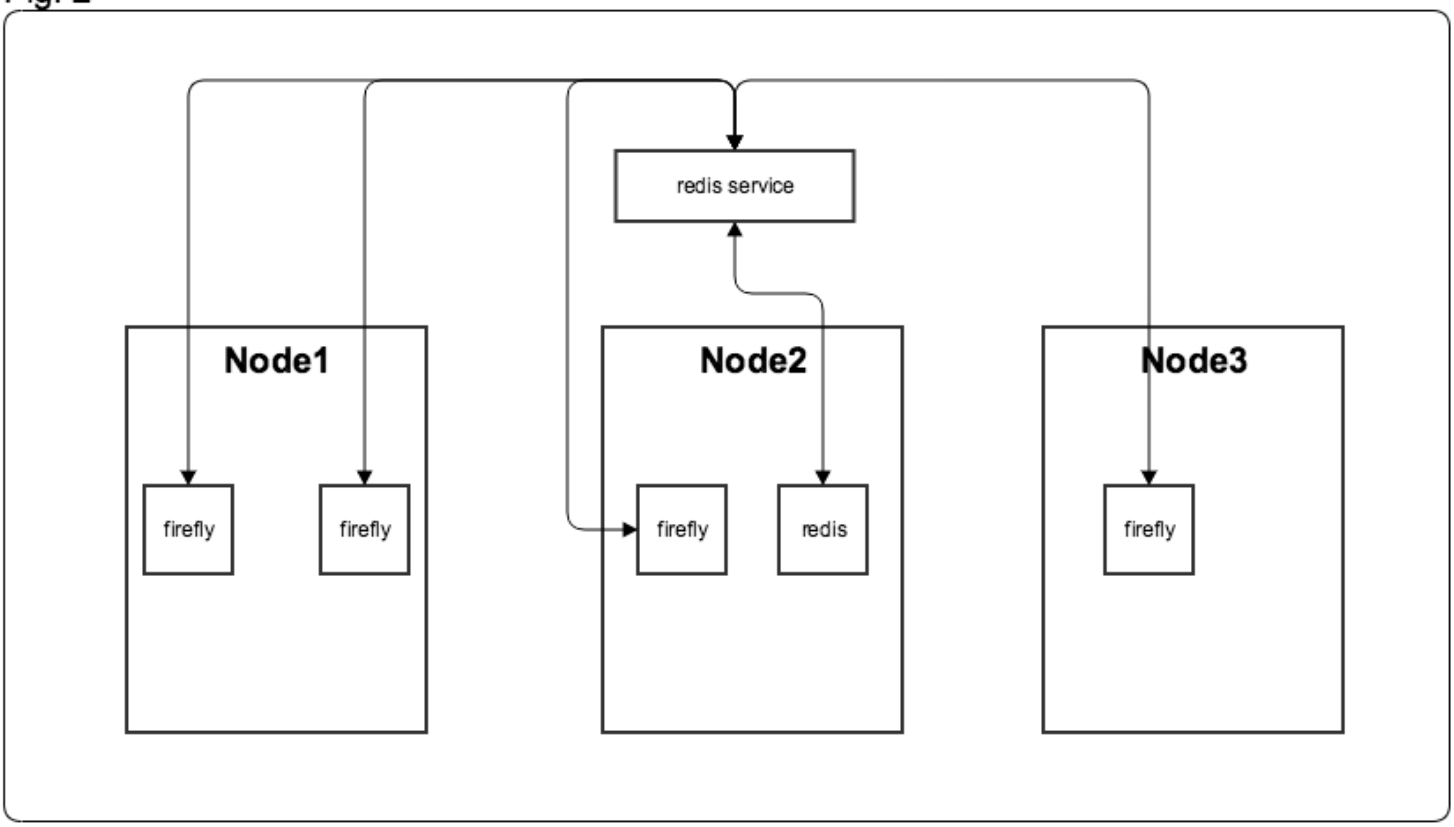


In this diagram, is the same auto-discovery using redis. Redis is a simple in-memory data store. It will be responsible for storing the states of the firefly instances. A service is defined for redis so that firefly have a common way to reference redis regardless of where it may be deployed to.

In this design, a 'heart-beat' with information like IP, port, and timestamp is placed into redis periodically. At the same time when a 'heart-beat' is sent, it will query redis for all of the others entries to form a list of active firefly instances. This list is then use to establish TCP cache replication among them.

In this setup, a firefly deployment would consist of one redis, and one or many firefly pods. If the redis pod get evicted, a new one will replaces it, and within a cycle, all hear-beat information will be recreated.

Fig. 2



Conclusion:

The redis solution is recommended due to its simplicity. The container is lightweight and can easily be bundled with firefly's deployment.
And unlike Consul, it does not add additional dependencies to the cluster setup.