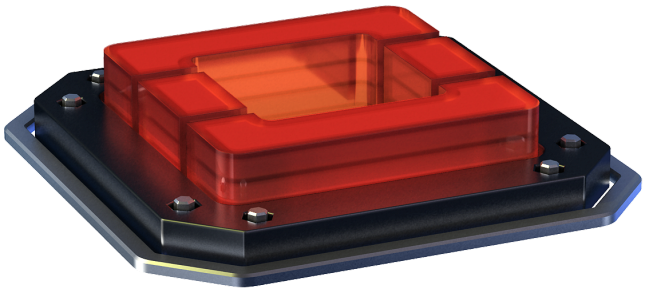
US HEADQUARTERS

525 Almanor Ave., 4th Floor

Sunnyvale, CA 94085

+1-650-963-9828 Phone

+1-650-963-9723 Fax



PoC

Calico for k8s + CCP hybrid cloud

Aleksandr Didenko

Table of Contents

Status of the document

Ready for review.

Changelog

Version	Date	Changes	Contributors
v1	26.07.2016	First version with 3 proposed designs.	Aleksandr Didenko
v2	22.08.2016	Addressing comments, adding conclusion and known issues.	Aleksandr Didenko
v3	23.08.2016	Proposed design #3 is over-engineering, removed.	Aleksandr Didenko
v4	29.08.2016	Added link to demo, addressed some comments, preferred design changed, added PoC for design #2 section.	Aleksandr Didenko
v5	13.09.2016	Updates in design #2 section: cross-workload security, issues and fixes, link to the demo.	Aleksandr Didenko, Alexander Saprykin
v6	22.10.2016	Minor style and wording fixes	Aleksandr Didenko

Scope

This document covers the ideas of networking design for containers and VMs on bare metal (k8s on bare-metal) and CCP on top of it using Calico as a network plugin.

The goal of this document is to research connectivity options between OpenStack and Kubernetes workloads and provide resulting design and implementation as a PoC.

- Overall networking design for k8s and CCP
- Integration points between Kubernetes and CCP
- Exposing Kubernetes endpoints to OpenStack VMs and vice versa
- Deployment scripts to reproduce PoC

Glossary

Pod

Set of containers working in same networking namespace and using same interface.

Worker node

Bare Metal Server that runs Docker runtime, Calico SDN, and Kubernetes internals to run Pods.

Calico

Calico is an open source solution for virtual networking in cloud data centers which provides a simple, pure layer 3 networking approach with no overlays for networking "workloads" such as VMs and containers.

Endpoint

In Calico's terminology, an endpoint is an IP address and interface. It could refer to a VM, a container, or even a process bound to an IP address running on a bare metal server.

From Kubernetes standpoint: an IP address and a port assigned to a Pod.

Service

Kubernetes load balancing and workload exposition mechanism, which provides the access to Pods (generally endpoints). Usually traffic is handled by iptables, which is managed by kube-proxy in the same way on every node.

Cluster IP

IP address from *service-cluster-ip-range* which is configured in kube-apiserver to access Service from Kubernetes cluster.

External IP

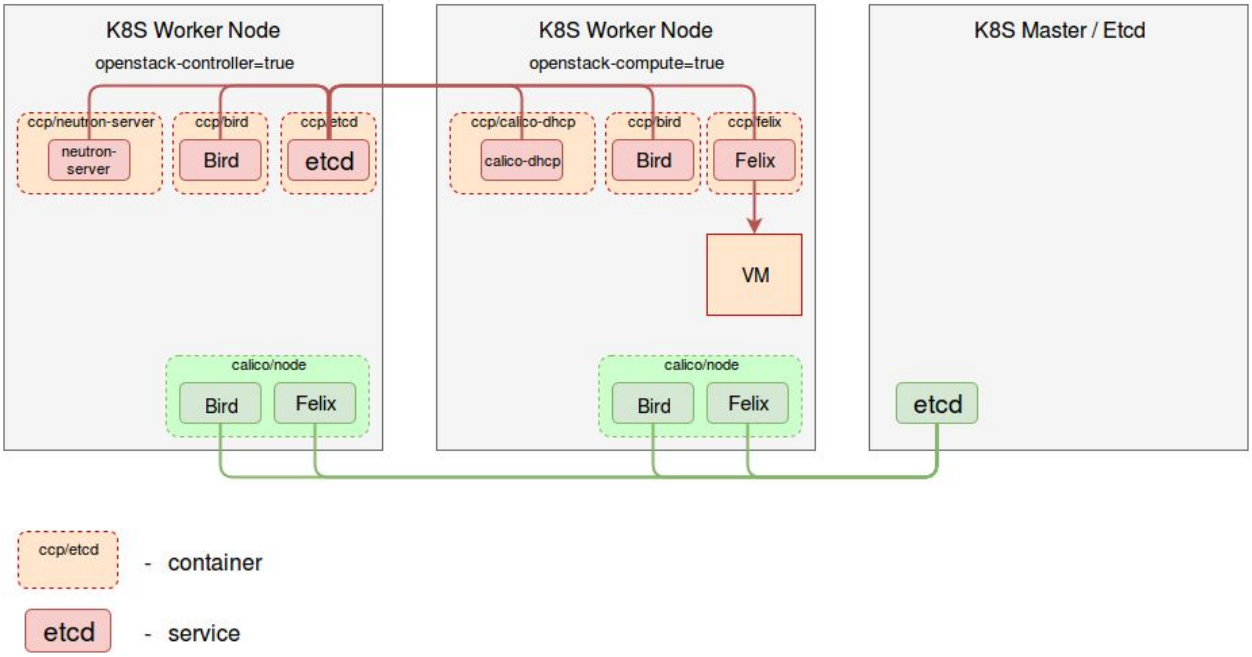
IP address provided by end user which can be used to access Service from outside of Kubernetes cluster.

CCP

OpenStack Containerized Control Plane - openstack services running in containers as kubernetes Pods and Services on top of Kubernetes cluster.

Reference design

This chapter covers the reference design for both k8s and OpenStack services (in our case CCP) with Calico networking plugin



This is how network related services would look like if we deploy k8s (green part) and CCP (red part) following Calico installation documentation. Think of those two parts as of different standalone deployments of Kubernetes+Calico and OpenStack+Calico. This design shows only services that are directly related to providing network connectivity for Pods and VMs.

Kubernetes (green) part

- K8s uses Calico CNI plugin and calico-node container to handle IP assignment to Pods running in Kubernetes.
- K8s part uses main Etcd cluster that's also used for Kubernetes deployment to store configuration data.
- **calico-node/felix** is responsible for local routing and iptables rules to provide network connectivity for containers.
- **calico-node/bird** is responsible for detecting IP/interface/routing changes on a Host system and propagating those changes through BGP.
- Calico components use the main Etcd cluster as configuration data source.

OpenStack CCP (red) part

- CCP uses Networking-calico which includes neutron ml2 plugin and calico-dhcp-agent to handle IP/nic/ports assignment to VMs.
- Networking-calico uses its own Etcd cluster (ccp/etcd pod in our case) to store its configuration data.
- **ccp/felix** is responsible for creation of local routing and iptables rules for VMs to provide network connectivity.
- **ccp/bird** is responsible for detecting IP/interface/routing changes on a Host system and propagating those changes with peers using BGP protocol.
- Calico components use separate Etcd pod/service (ccp/etcd) as configuration data source.

Pros and cons

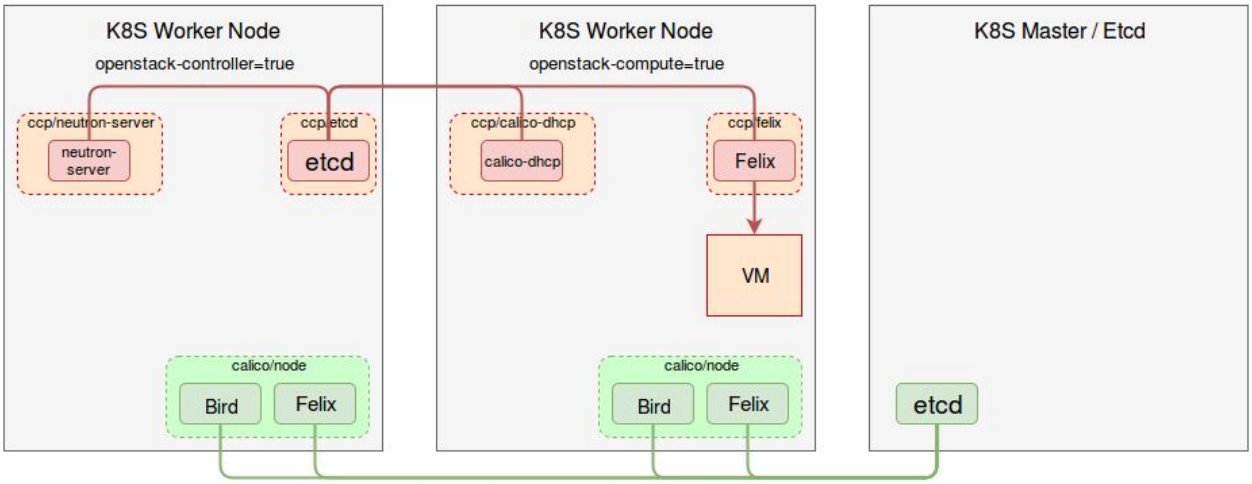
Pros:

- Reference architecture for both Kubernetes and Calico, no need to change the code.
- No interference into the main Etcd cluster from Pods.

Cons:

- Services duplication (Felix, Bird) which may be a problem for Bird - we have two BGP agents doing basically the same on the Host level. This scheme creates more components thus there is a bigger chance of failures. Additionally to that additional services require more hardware resources [1]

Proposed design #1



Kubernetes (green) part

- Remains intact comparing to the reference design.

OpenStack CCP (red) part

- Don't deploy **ccp/bird** - **calico-node/bird** is watching changes on a Host system so it should be able to detect changes related to both containers and VMs and propagate them. May require changes in bird configuration.

Pros and cons

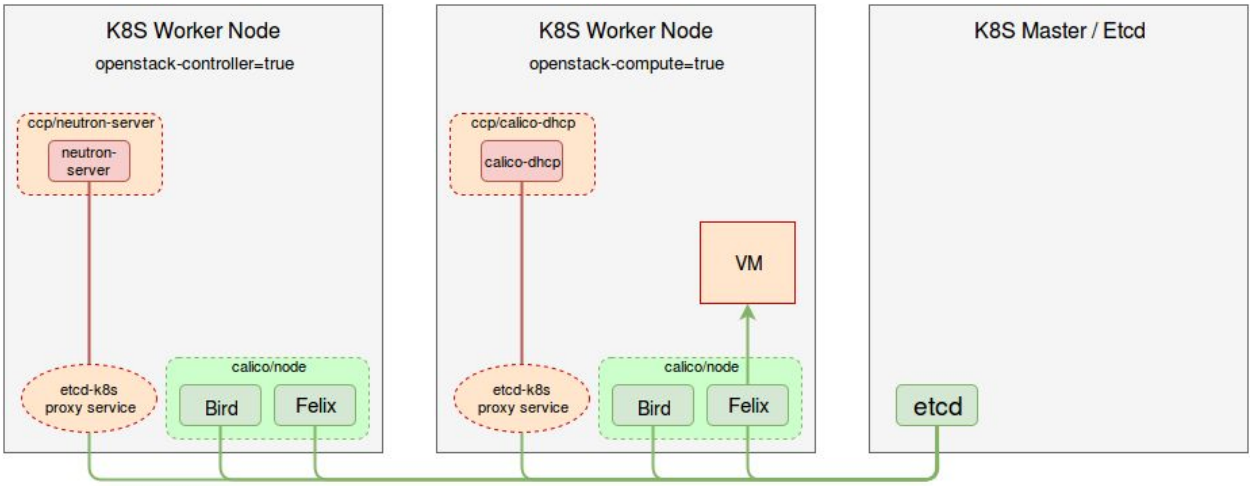
Pros:

- No interference into the main Etcd cluster from Pods.
- Felix duplication: since Kubernetes and OpenStack are not connected, each of them may require different Calico codebase versions to work properly. This design allows us to ship different versions of Felix containers.

Cons:

- Felix duplication: we have two services doing the same in a single namespace which is not supported by Calico design.

Proposed design #2



Kubernetes (green) part

- From K8s configuration point of view design remains intact. However, from the functional point of view it is affected by OpenStack part since Calico components from both OpenStack and K8s share the same configuration data source - Etcd cluster.
- Additional Kubernetes proxy service is required to provide the connectivity from CCP pods to the main Etcd cluster (they cannot connect to etcd-proxy on a localhost since some containers are running in isolated network space, for example neutron-server).

OpenStack CCP (red) part

- Don't deploy **ccp/bird** - **calico-node/bird** is watching changes on a Host system so it should be able to detect changes related to both containers and VMs and propagate them. May require changes in bird configuration.
- Don't deploy **ccp/felix** - **calico-node/felix** is watching Etcd and should be able to detect new workloads (posted by networking-calico neutron ml2 plugin) and create needed routes and iptables rules on a host system.

Pros and cons

Pros:

- No services duplication

Cons:

- Using the same Etcd cluster for both workloads is quite risky since any failures and/or misconfigurations on CCP/Calico layer may affect Kubernetes and thus may break the whole platform.
- Deployment scheme becomes more complex since we need to provide access to the main Etcd cluster for Pods.
- Upgrades: using the same Felix for both OpenStack and Kubernetes Calico parts may not work well or get broken during upgrades of one of the components due to difference in versions of Calico components.

Comparison

Metric/Feature	Design #1	Design #2
Possibility to share IP pools	No	No
Possibility to use different IP pools	Yes	Yes
Networking-calico interference into the main Etcd cluster	No	Yes
Calico-Felix duplication	Yes	No
Cross-workload security	No	Yes
Maintenance costs	Low	Medium
Working PoC	Yes	Yes

The preferred solution is design #2, because:

1. Running two copies of Felix in the same namespace is not supported.
2. Using single Etcd DB for both Kubernetes and OpenStack Calico components will enable cross-workload security.

PoC for design #1

Demo

https://asciinema.org/a/83702

Known issues

1. **calico-node/felix** flushes some iptables rules added by **ccp/felix** (for example, metadata DNAT in felix-PREROUTING chain). It's possible to resolve this by using different FELIX_PREFIX for iptables rules in **ccp/felix** [2]. Requires additional research.
2. File /etc/resolv.conf on VMs should be updated to k8s one in order to allow VMs to see k8s services/DNS.
3. Nova metadata service is not working. We can try to use felix NAT redirection to nova-api k8s service CLUSTER_IP:METADATA_PORT.
4. **ccp/etcd** container is not stateful.
5. Iptables conflicts and packet loss when Calico pool for Kubernetes is configured with --ipip option (IPIP tunneling). Solvable via custom prefix [3].
6. **calico-node/bird** does not announce routes for IPs that are not in calico pools. An update to bird/confd configuration is needed to resolve this [4] [5].
7. No integration points due to different Etcd DBs and thus no cross-workload security.

PoC for design #2

Demo

https://asciinema.org/a/85509

Known issues

1. **calico-node/bird** does not announce routes for IPs that are not in calico pools. An update to bird/confd configuration is needed to resolve this [4] [5].
2. **ccp/networking-calico** and **calico-node/felix** use different /calico/v1/config/InterfacePrefix ('tap' and 'cali' respectively). Which causes conflicts between OpenStack and Kubernetes endpoints: ml2 plugin sets InterfacePrefix to 'tap' in etcd (hardcoded) and after that **calico-node/felix** can not assign IPs for new Kubernetes PoDs (Validation failed for endpoint. Interface u'caliadcbb9926f6' does not start with 'tap') [6].
3. File /etc/resolv.conf on VMs should be updated to k8s one in order to allow VMs to see k8s services/DNS.
4. Nova metadata service is not working.
5. **calico/ctl:v0.20.0** replaces endpoint name with generated by default. Fixed by a patch [7], available in docker image **calico/ctl:latest**.

Integration points

ccp/networking-calico implements neutron security groups by mapping security groups to calico profiles. Each security group is represented as a calico security profile, that is stored in the etcd by the following location:

```
/calico/v1/policy/profile/openstack-sg-{SECGROUP_ID}
```

By default security group allows inbound connections only inside that security group. Restrictions by security group IDs are implemented using profile tags. Each calico profile related to neutron security group by default has tag that matches security group ID.

```
$ etcdctl get \
    /calico/v1/policy/profile/openstack-sg-4fe422ed-4730-4af4-b835-38d09235bd4a/tags
["4fe422ed-4730-4af4-b835-38d09235bd4a"]
```

Since security group related profile is managed by networking-calico, it is not recommended to use it directly, by assigning it to the endpoint of k8s pod. Instead of this it is proposed to create another security profile with appropriate tag assigned to it.

Example:

```
$ calicoctl profile add k8s-to-sg1
$ calicoctl profile k8s-to-sg1 tag add 4fe422ed-4730-4af4-b835-38d09235bd4a
$ calicoctl profile k8s-to-sg1 \
      rule add inbound allow from tag 4fe422ed-4730-4af4-b835-38d09235bd4a

$ etcdctl get /calico/v1/policy/profile/k8s-to-sg1/tags
["k8s-to-sg1", "4fe422ed-4730-4af4-b835-38d09235bd4a"]

$ etcdctl get /calico/v1/policy/profile/k8s-to-sg1/rules |python -m json.tool
{
    "id": "k8s-to-sg1",
    "inbound_rules": [
        {
            "action": "allow",
            "src_tag": "k8s-to-sg1"
        },
        {
            "action": "allow",
            "src_tag": "4fe422ed-4730-4af4-b835-38d09235bd4a"
        }
    ],
    "outbound_rules": [
        {
            "action": "allow"
        }
    ]
}
```

Deployment scripts

- In order to reproduce the PoC from the scratch (starting from virtual lab creation) you can use the following scripts:
    - Proposed design #1: https://github.com/adidenko/vagrant-k8s (see ccp.yaml)
    - Proposed design #2: https://github.com/adidenko/vagrant-k8s
- In order to deploy CCP on top of preinstalled Kubernetes cluster you can use the following ansible playbooks (networking-calico requires k8s with calico CNI, neutron-ovs could be deployed on any k8s cluster): https://github.com/adidenko/fuel-ccp-ansible

Please also check “Known issues” in the chapters above.