ubuntu

Kubernetes - Networking

Konstantinos Tsakalozos

CANONICAL

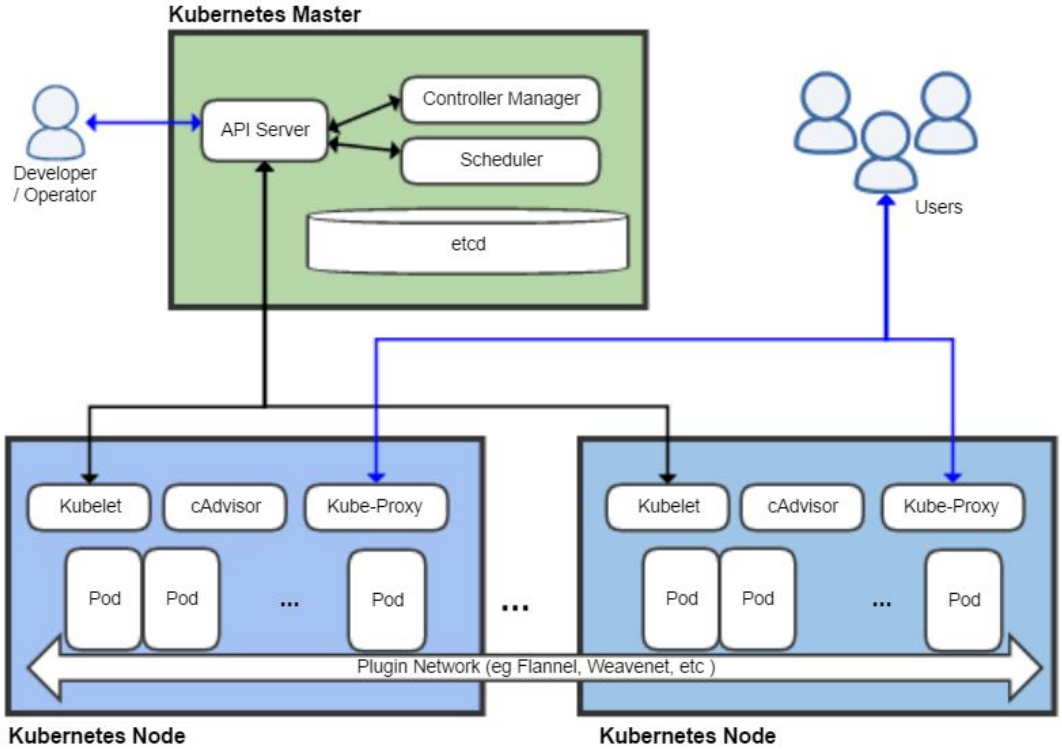
Kubernetes

From the greek word “κυβερνήτης” originally designed by Google and donated to the Cloud Native Computing Foundation.

Presented at "Large-scale cluster management at Google with Borg" Proc EuroSys 2015.

“Kubernetes is an *open-source* platform designed to automate deploying, scaling, and *operating* application *containers*.”

Kubernetes architecture



- Replicating app instances
- Balancing loads
- Self-healing apps
- Rolling updates
- Naming and discovering
- Monitoring resources

Pod - The Building Block

Pod: group of containers, always co-located

- Scheduling is on Pods
- Pod sets the Linux namespaces and cgroups
- Containers in a Pod:
  - Same IP
  - Shared port space
  - Contact each other with no restrictions via localhost
  - Can use standard IPC (inter-process communications)

Pods - Namespaces and Cgroups

Namespaces - restrict the view of the process. Virtualise network stack:

- Sockets, network interfaces, routing tables, ip tables

Cgroups - resource metering and limiting.

- Traffic priority

Pod-to-Pod Communication

Kubernetes fundamental network requirements:

- all containers can communicate with all other containers without NAT
- all nodes can communicate with all containers (and vice-versa) without NAT
- the IP that a container sees itself as is the same IP that others see it has

Reasoning:

- low-friction porting of apps from VMs to containers
- open to multiple implementations
- backwards compatibility

Container Network Interface

CNI: container network connectivity, release resources when containers are removed

Project: specification and libraries for writing plugins to configure network interfaces

Plugins using CNI:

- Project Calico - a layer 3 virtual network
- Weave - a multi-host Docker network
- Contiv Networking - policy networking for various use cases
- Amazon ECS CNI Plugins - a collection of CNI Plugins to configure containers with Amazon EC2 elastic network interfaces (ENIs)
- And many more

Simple Network - Flannel

Flannel designed for Kubernetes:

- layer 3 network fabric
- easy to configure

`flanneld` on each host:

- Allocating a subnet lease from preconfigured address space
- Network configuration, the allocated subnets, and any auxiliary data (such as the host's public IP) on etcd or via kube master.
- Packets are forwarded using one of several backend mechanisms including VXLAN and various cloud integrations.

Network Policies

Implemented by network plugins (eg calico, canal)

Policies: pod Ingress & Egress

- ingress: whitelist ingress rules allowing traffic which matches both the from and ports sections.
- egress: rules allowing traffic which matches both the to and ports sections

```
ingress:
- from:
  - ipBlock:
      cidr: 172.17.0.0/16
  ports:
  - protocol: TCP
    port: 6379
```

```
egress:
- to:
  - ipBlock:
      cidr: 10.0.0.0/24
  ports:
  - protocol: TCP
    port: 5978
```

Pod-to-Service Communications

Pods are not reliable. May crash and be rescheduled at any time!

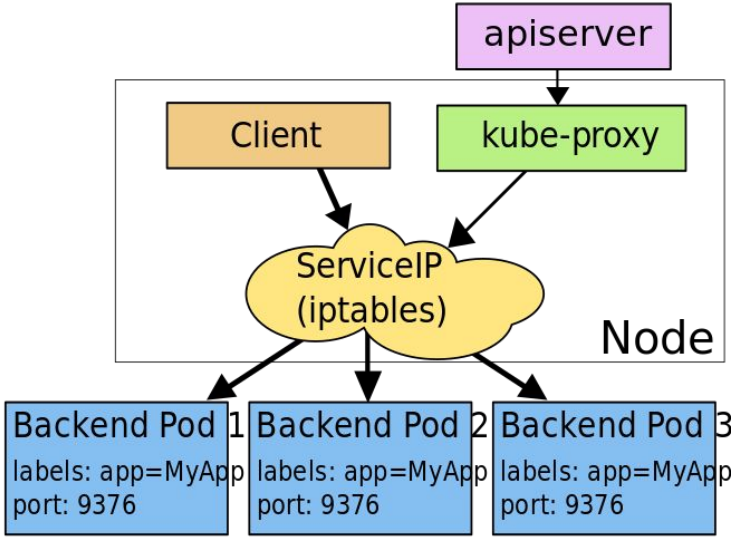
Service: abstraction over a) logical set of Pods and b) access policy

Services: abstract other kinds of backends. Model endpoints of external services (eg databases).

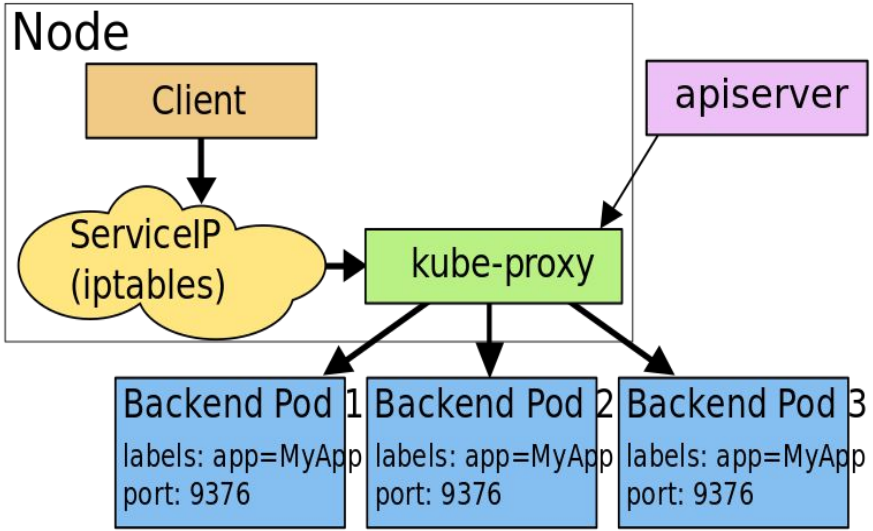
ServiceTypes

- ClusterIP: Service on a cluster-internal IP
- NodePort: Service on each Node’s IP at a static port
- LoadBalancer: Service on a cloud provider’s load balancer. NodePort and ClusterIP are used

Pod-to-Service Communications



Requests are routed through iptable rules



Requests hit kube-proxy and bounce to a Pod

Service Discovery

DNS server running as service inside your cluster.

The DNS server watches the Kubernetes API for new **Services.**

> Example, a **Service** called **"my-service"** in **Namespace** **"my-ns".** Results of **"my-service.my-ns"** is the cluster IP.

DNS SRV (service) records for named ports.

> Example, **"my-service.my-ns"** **Service** has a port named **"http"** with protocol **TCP**, a DNS SRV query for **"_http._tcp.my-service.my-ns"** to discover the port number for **"http"**.

Services - Notes

Pods: IP addresses -> route to a fixed destination

Service: Virtual IPs -> not pointing to a single host

Routing, IP tables and DNS for Services are dynamic populated

Two proxy modes - userspace and iptables, which operate slightly differently.

Iptables operations slow in large clusters e.g 10,000 Services

- IPVS for load balancing, in-kernel hash tables, performance consistency
- IPVS-based kube-proxy with sophisticated load balancing algorithms (least conns, locality, weighted, persistence)

Ingress Services

Kubernetes cluster hosts APIs

Ingress is a Service:

> Give services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting

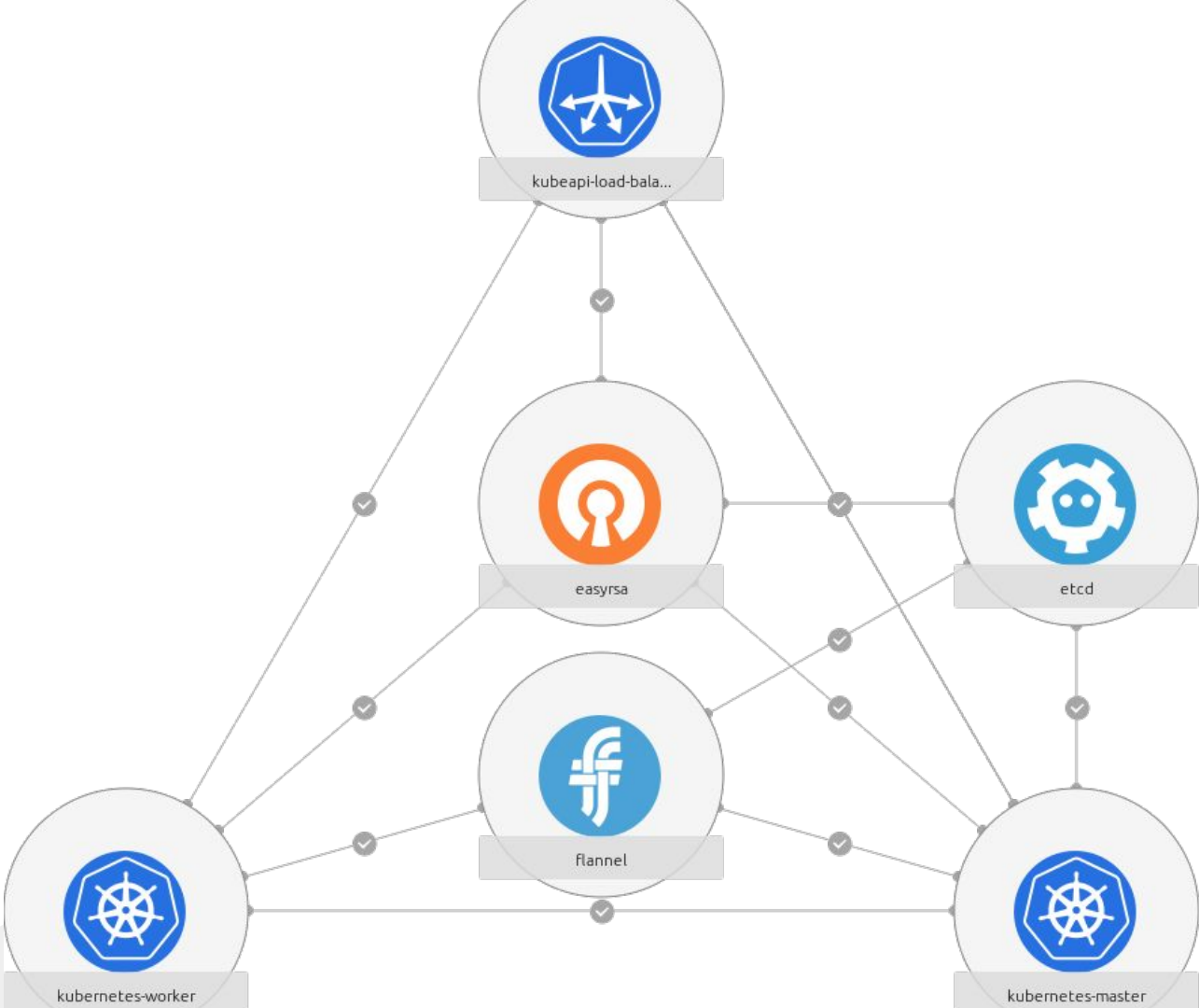
Example #1, Simple fanout:

```
foo.bar.com -> 178.91.123.132 -> / foo    s1:80
                                 / bar    s2:80
```

Example #2, Name based virtual hosting:

```
foo.bar.com --|                 |-> foo.bar.com s1:80
              | 178.91.123.132  |
bar.foo.com --|                 |-> bar.foo.com s2:80
```

The CDK, Canonical Distribution of Kubernetes Bundle



Canonical Distribution of Kubernetes

Canonical packages, distributes and offers support for Kubernetes. Highlights:

- Vanilla Kubernetes
- Clean upgrade path
- Cloud Neutral: AWS, Azure, Google, Oracle, Rackspace, SoftLayer, or private VMware, OpenStack or bare metal.
- Multi-arch: amd64, s390x (others to come)

Thanks!

Resources

K8s Docs:

- Kubernetes: https://kubernetes.io/docs/home/

CDK source:

- Canonical Distribution of Kubernetes: https://www.ubuntu.com/kubernetes
- Kubernetes upstream: https://github.com/kubernetes/kubernetes/tree/master/cluster/juju
- Juju-solutions: https://github.com/juju-solutions

You can reach us at:

- Mailing list: https://lists.ubuntu.com/mailman/listinfo/juju
- Bugs: https://github.com/juju-solutions/bundle-canonical-kubernetes/issues