# Kubernetes

**Thorsten Scherf**
**Red Hat**

**SLAC 2015, Berlin**

# Agenda

- What is Kubernetes
- Kubernetes Architecture
- Configuration and Cluster Setup
- Demo

# DOCKER PROS AND CONS

- Pros
  - Ease of use, makes Linux containers consumable
  - Very easy to create and work with derivative images
  - Fast boot on containers


- Cons
  - Host centric solution, not aware of anything else
  - Cannot handle networking between containers on separate hosts
  - No higher level provisioning to connect related containers

Kubernetes is not limited to Docker, support for AppC based containers (like rkt) and other application containers is coming (*).

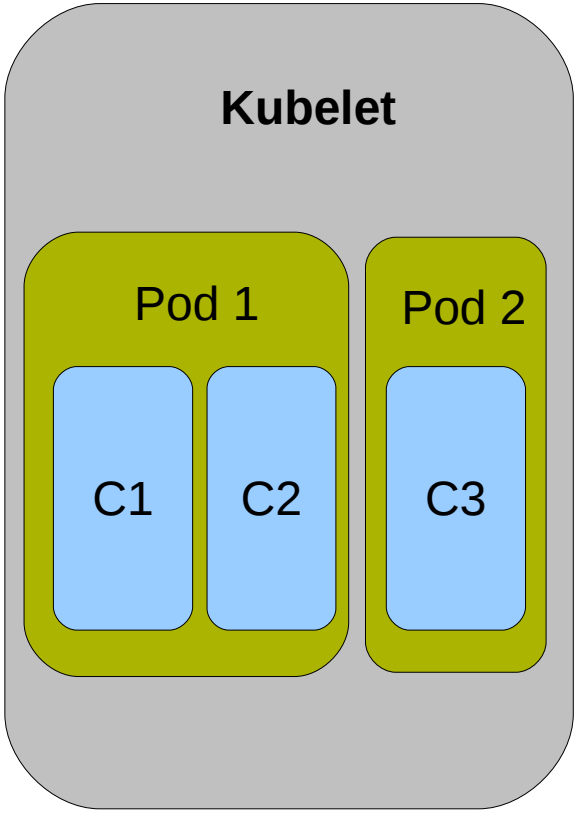
(*) https://xkcd.com/927/

# What is Kubernetes?

- A highly collaborative open source project originally conceived by Google
  - Google has 10+ years experience w/ containerized apps
- Start, stop, update, and manage a cluster of machines running containers in a consistent and maintainable way.
- Sometimes called:
  - kube
  - k8s (that's 'k' + 8 letters + 's')

# WHAT IS KUBERNETES

- Kubernetes is a container cluster manager
- Manages containerized applications in a clustered environment
- It provides discovery across the cluster




- Kubelet (daemon on the worker node)
- Pods (collection of containers)
- Services (discovery mechanism between pods)
- Replication Controllers (replicated and monitored pods)

# What is Kubernetes?

- Particularly suited for horizontally scaleable, stateless, or 'microservices' application architectures.
  - Does not mean others will not work or are ignored
- Additional functionality to make containers easier to use in a cluster (reachability and discovery).
- Kubernetes does NOT and will not expose all of the 'features' of the docker command line.

# Kubernetes Key Words

- Master
- Node/Minion

- Pod
- Replication Controller
- Service

- Label
- Namespace

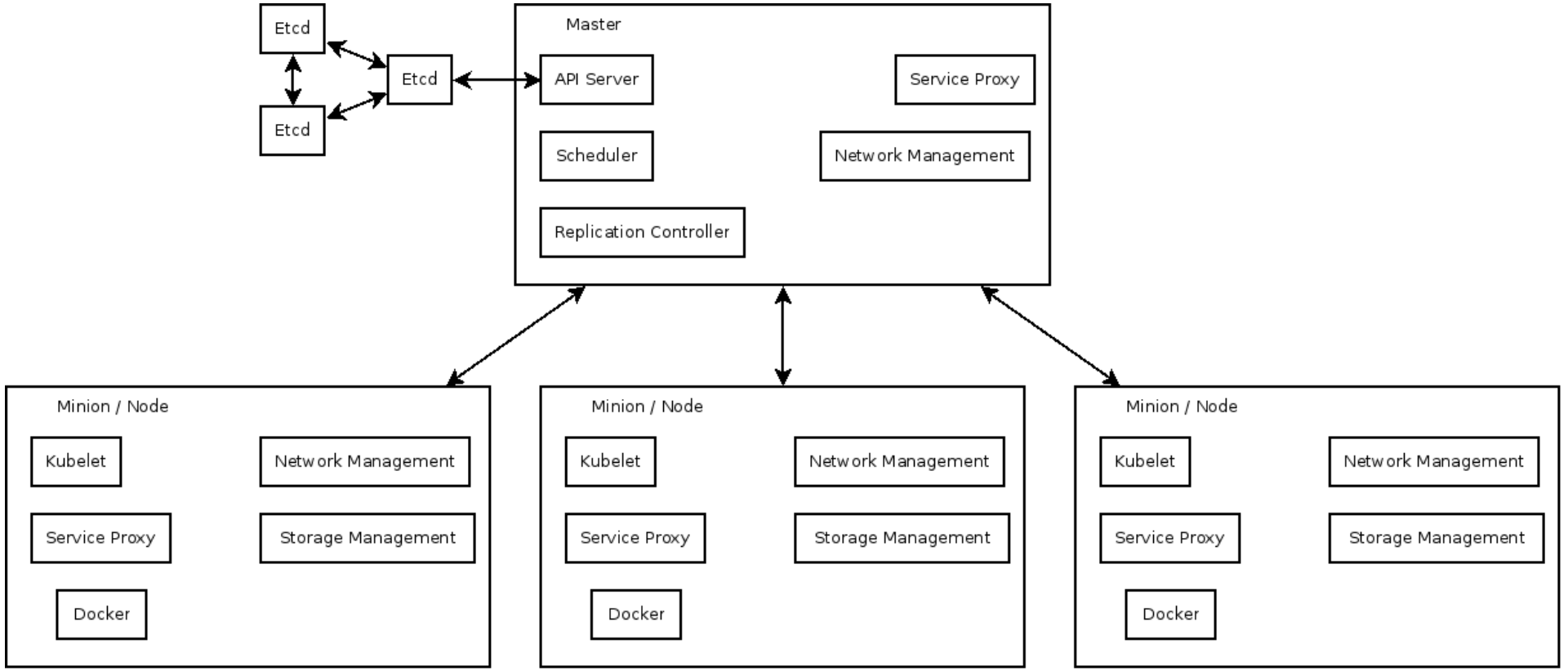
# Master

- Typically consists of:
  - kube-apiserver
  - kube-scheduler
  - kube-controller-manager
  - etcd
- Might contain:
  - a network management utility

# Node - Minion

- Typically consists of:
  - kubelet
  - kube-proxy
- Might contain:
  - a network management utility

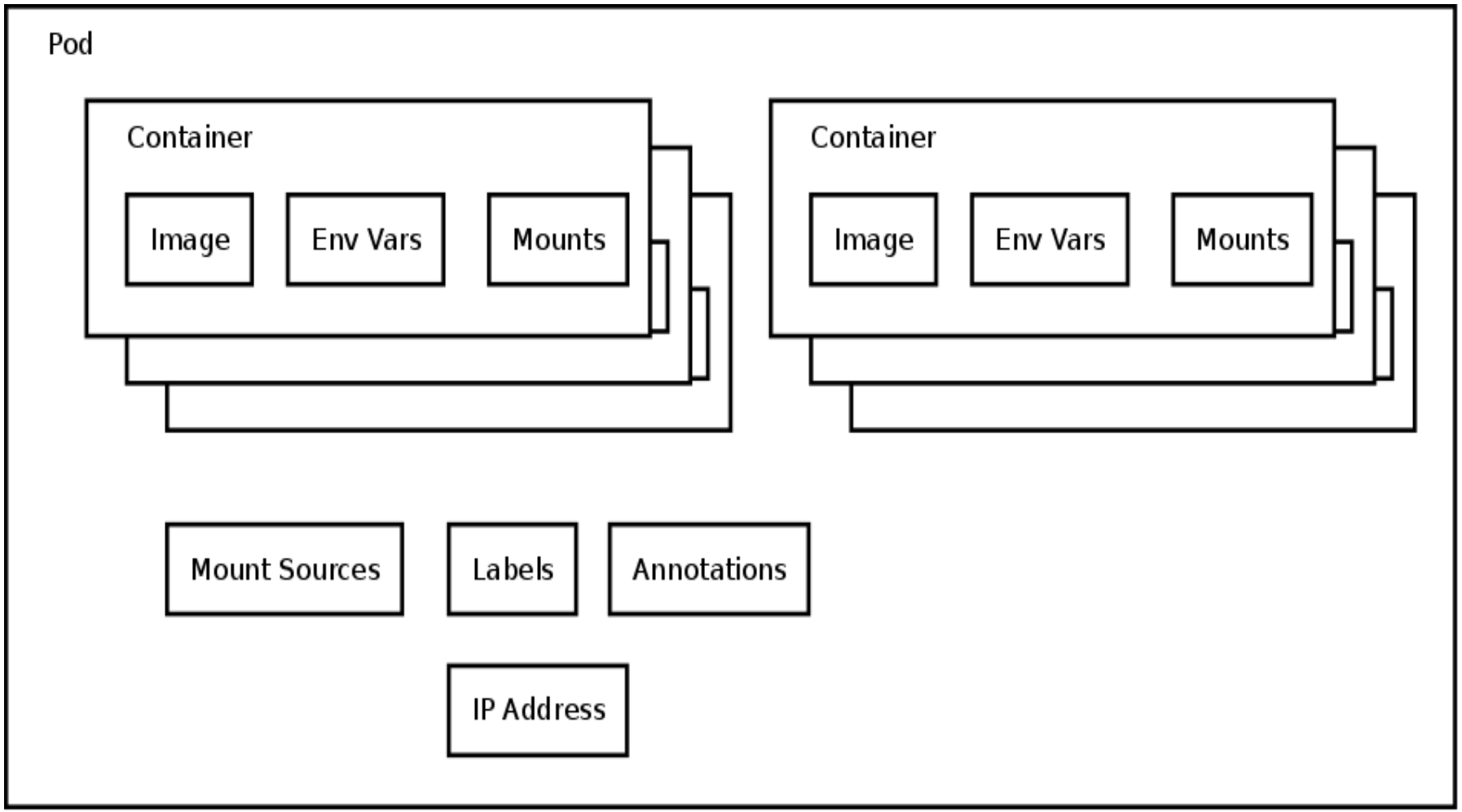
# Architecture



# Pod

- Single schedulable unit of work
  - Can not move between machines
- One or more containers
  - Shared network namespace
- Every pod gets an unique IP
  - Assigned by the container engine, not kube!
- Metadata about the container(s)
- Env vars – configuration for the container

# Pod


Pod
Container
Image
Env Vars
Mounts
Container
Image
Env Vars
Mounts
Mount Sources
Labels
Annotations
IP Address

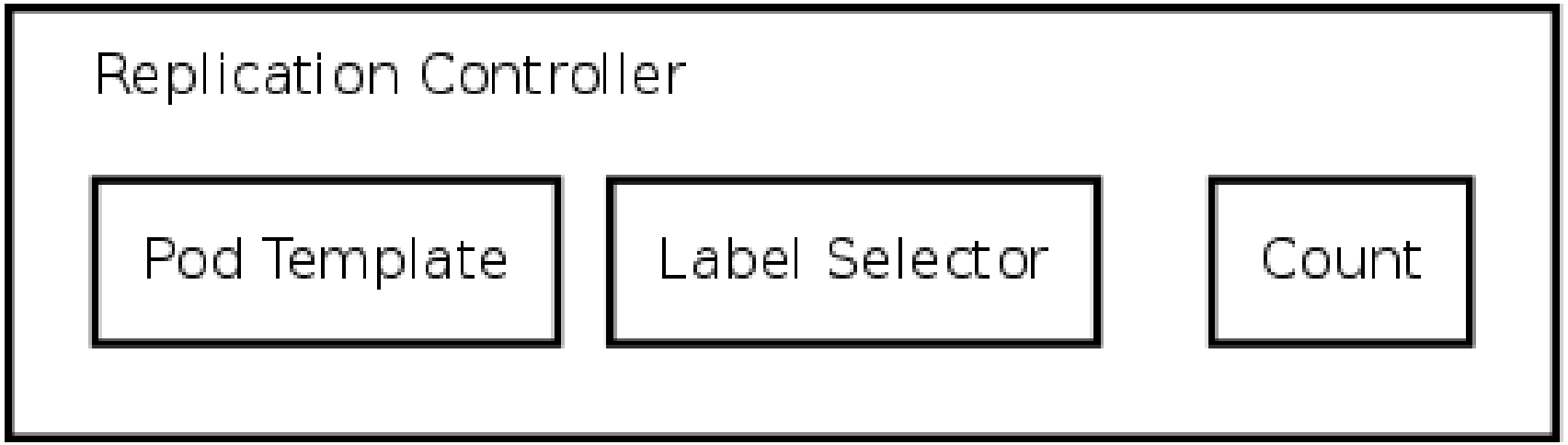
# Pod – Example JSON

```
{ "apiVersion": "v1beta1", "id": "apache", "kind": "Pod", "namespace": "default",
    "labels": { "name": "apache" },
    "desiredState": { "manifest": {
        "version": "v1beta1", "id": "apache", "volumes": null,
        "containers": [{ "name": "my-fedora-apache", "image": "fedora/apache",
            "ports": [{ "containerPort": 80, "hostPort": 80, "protocol": "TCP" }],
        }],
        "restartPolicy": {" always": {} }
    }, },
}
```

# Replication Controller

- Consists of
  - Pod template
  - Count
  - Label Selector
- Kube will try to keep $count copies of pods matching the label selector running
- If too few copies are running the replication controller will start a new pod somewhere in the cluster

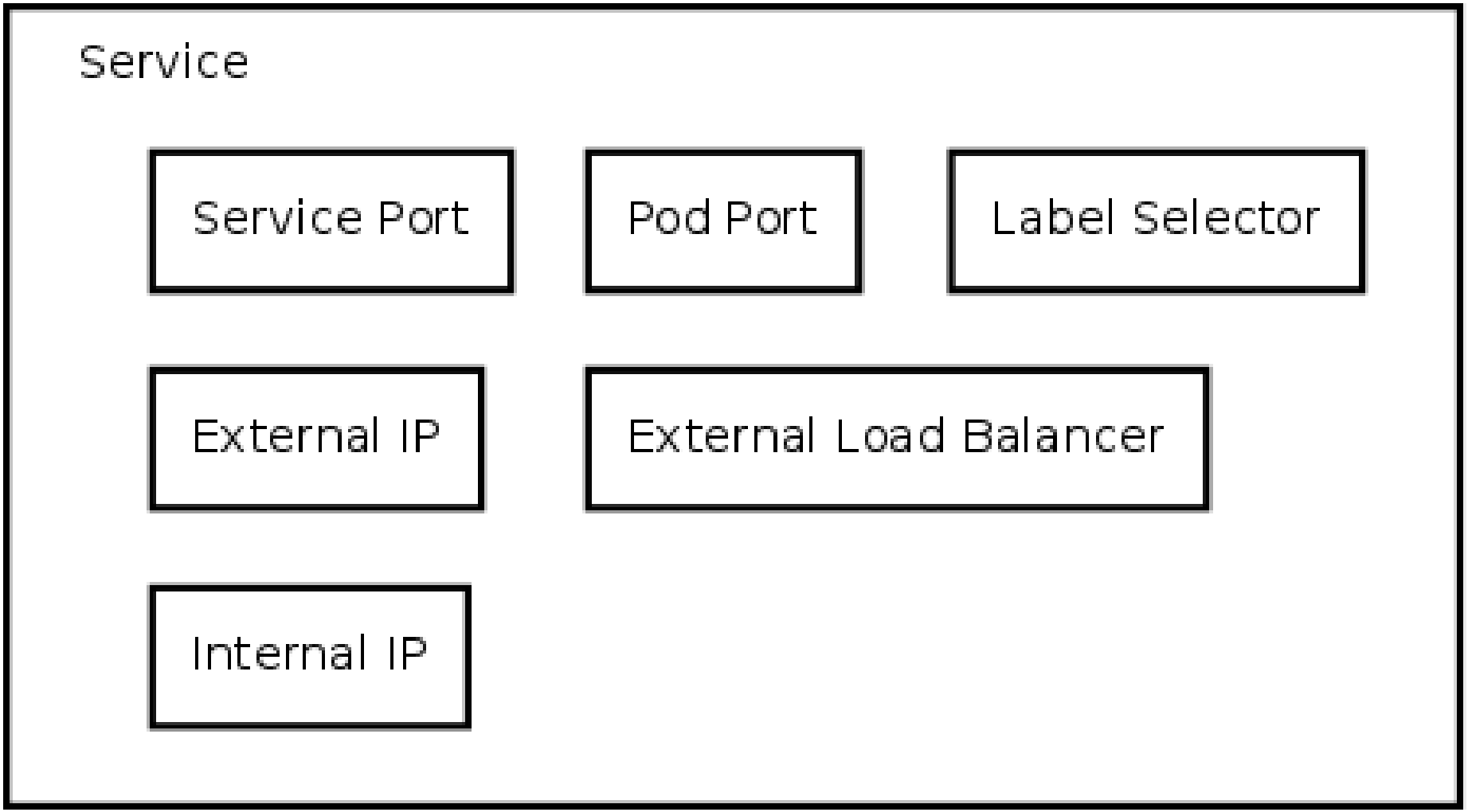
# Replication Controller



# Services

- Most every pod/replication controller will need a service.  What's the point of a pod that doesn't provide some sort of service/useful work?
- How 'stuff' finds pods which could be anywhere?
  - Containers are started and stopped dynamically by kube, thus always changing IP addresses
- Define:
  - What port in the container
  - Labels on pods which should respond to this type of request

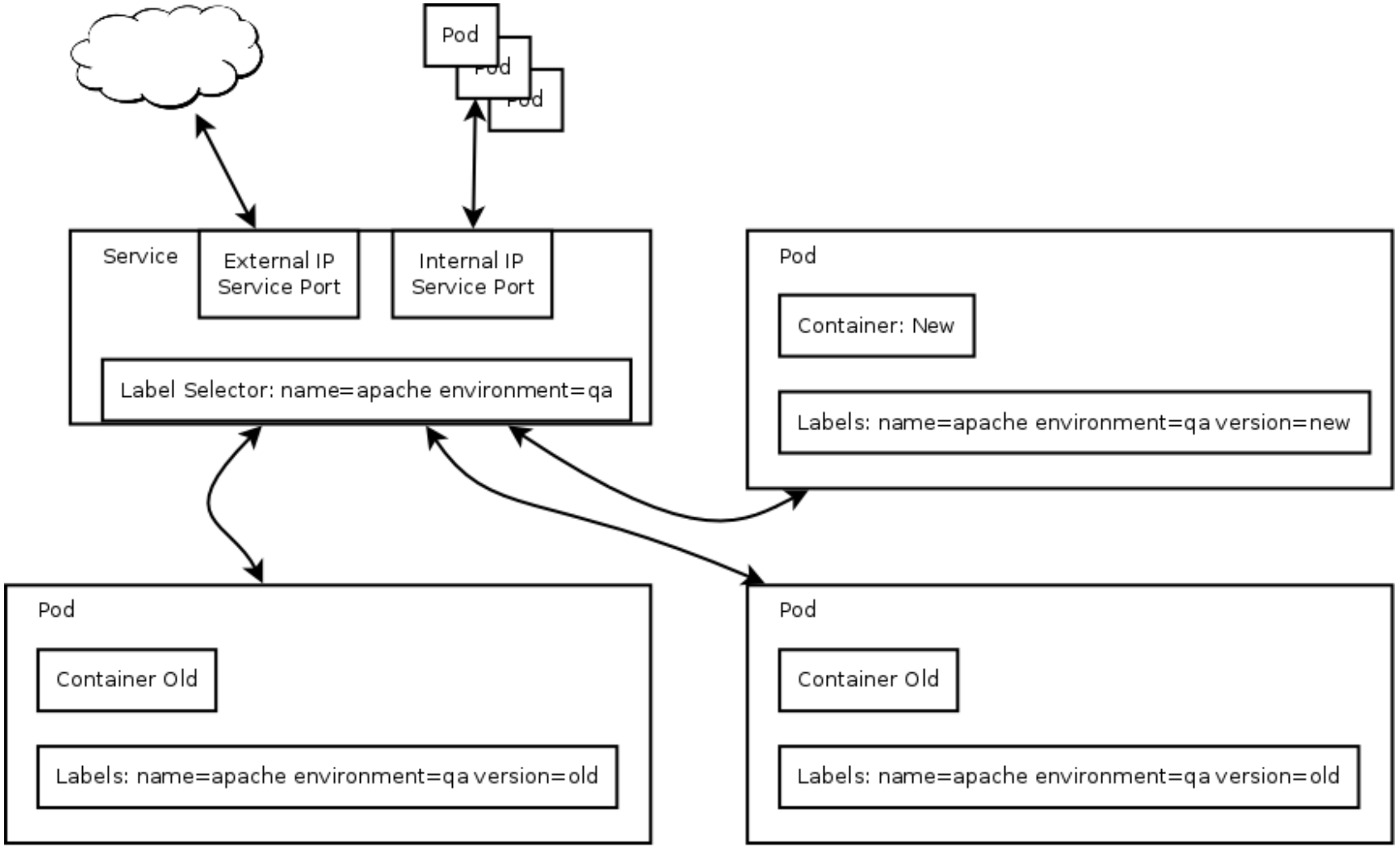
# Services



# Labels

- List of key=value pairs
- Attached to all objects
- Currently used in 2 main places
  - Matching pods to replication controllers
  - Matching pods to services
- Objects can be queried from the API server by label

# Services and Labels



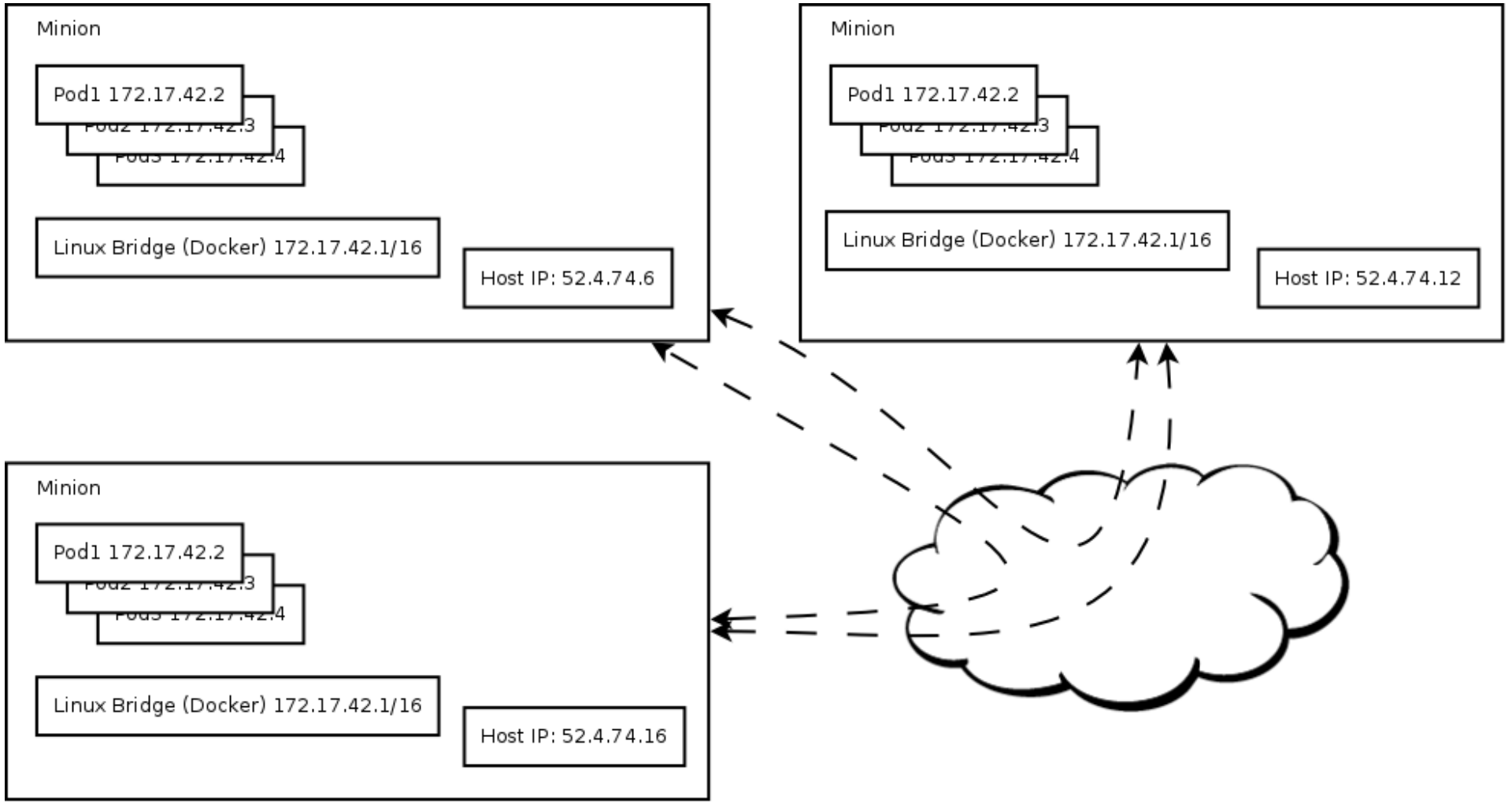
# Namespace

- Attached to every object
- Pods in ns1 will not get service variable from ns2
- Users with permission to CRUD (create, read, update, delete) objects in ns1 may not have permissions to CRUD object in ns2
- The network is not segregated.
- Some people consider using a namespace per application.  Some say a namespace per team or location.

# Networking Setup

- Networking is a docker problem – not kube
  - Kube makes those problems apparent!
  - If any two docker containers on any two hosts can talk over IP, kube will just work.
- Docker looks so easy
  - 2 containers on one host can easily talk.
  - How to get to those containers from outside?
  - How to get to from one container on one host to a container on another?
- Networking is really hard!

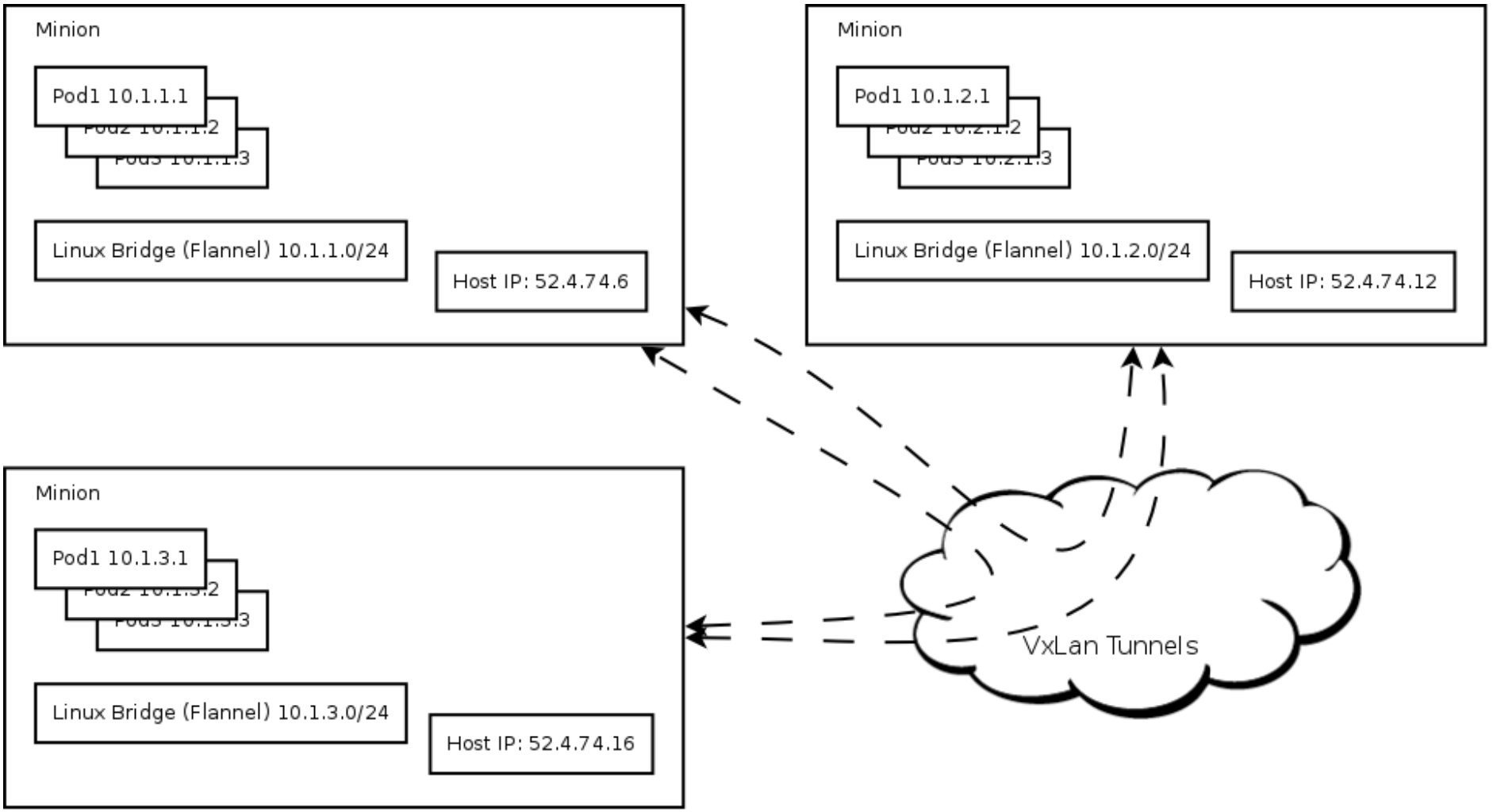
# Networking Docker Out Of The Box



# Networking Setup – Flannel

- Flannel
  - Super super easy configuration
  - Can create a vxlan overlay network
  - Can configure docker to launch pods in this overlay
  - Pods just work!
- There are many other solutions.
  - This one is easy.

# Networking with an overlay network



# Ressources

- Kubernetes

  http://kubernetes.io/
- Project Atomic

  http://www.projectatomic.io/

For questions please email

Thorsten Scherf <tscherf@redhat.com>

Credits go to Eric Paris for his excellent slidedeck.