RED HAT
FORUM
Europe, Middle East & Africa

redhat.

Shifting to containers - Docker and Kubernetes on Red Hat Atomic

Yoel Jacobsen
CTO / Emet Computing

About

PREMIER
BUSINESS PARTNER

- Red Hat Premier Business Partner
- UNIX and Linux specialists for over 30 years
- Full-stack expertise – From API and analytic down to servers, racks and PDUs

Containers vs. Virtualization

- Virtualization
  - Hardware abstraction
  - To the host, the entire guest is a process
- Containers
  - Kernel abstraction
  - Each container process is a separate “host” process
  - Essentially – a chroot on steroids

Containers

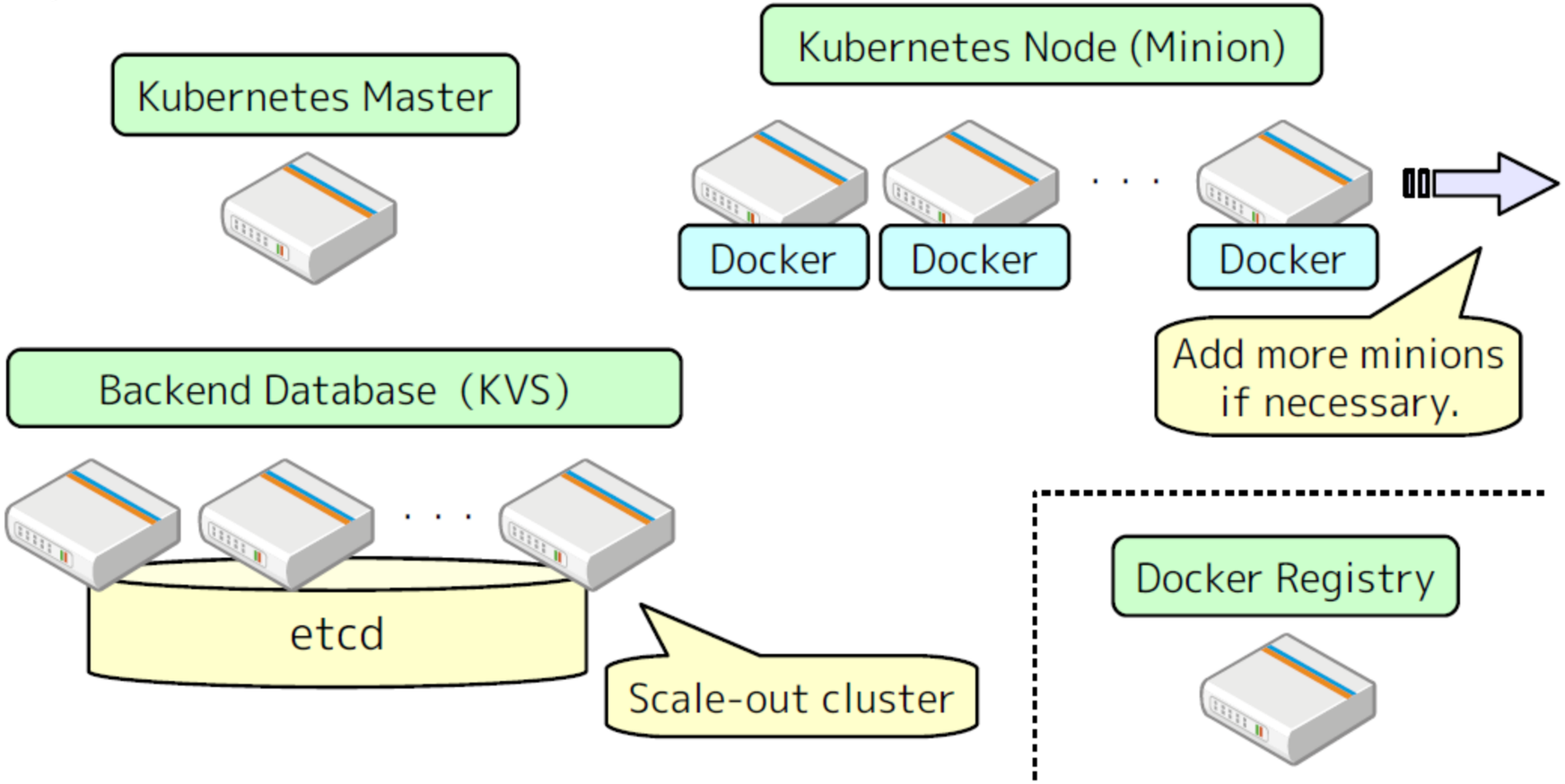
- Look like a VM
  - One can SSH into a container
  - With root access
  - Install applications
  - Modify iptables
- But
  - It boots in milliseconds
  - Provides bare metal performance
  - May share resources with the “host”

What's new?

- Docker made LXC usable
- But not for production load
- For production you want
  - Availability management
  - Dev-Ops relationships
  - Auto scaling
  - Better security

Kubernetes

- Container cluster management



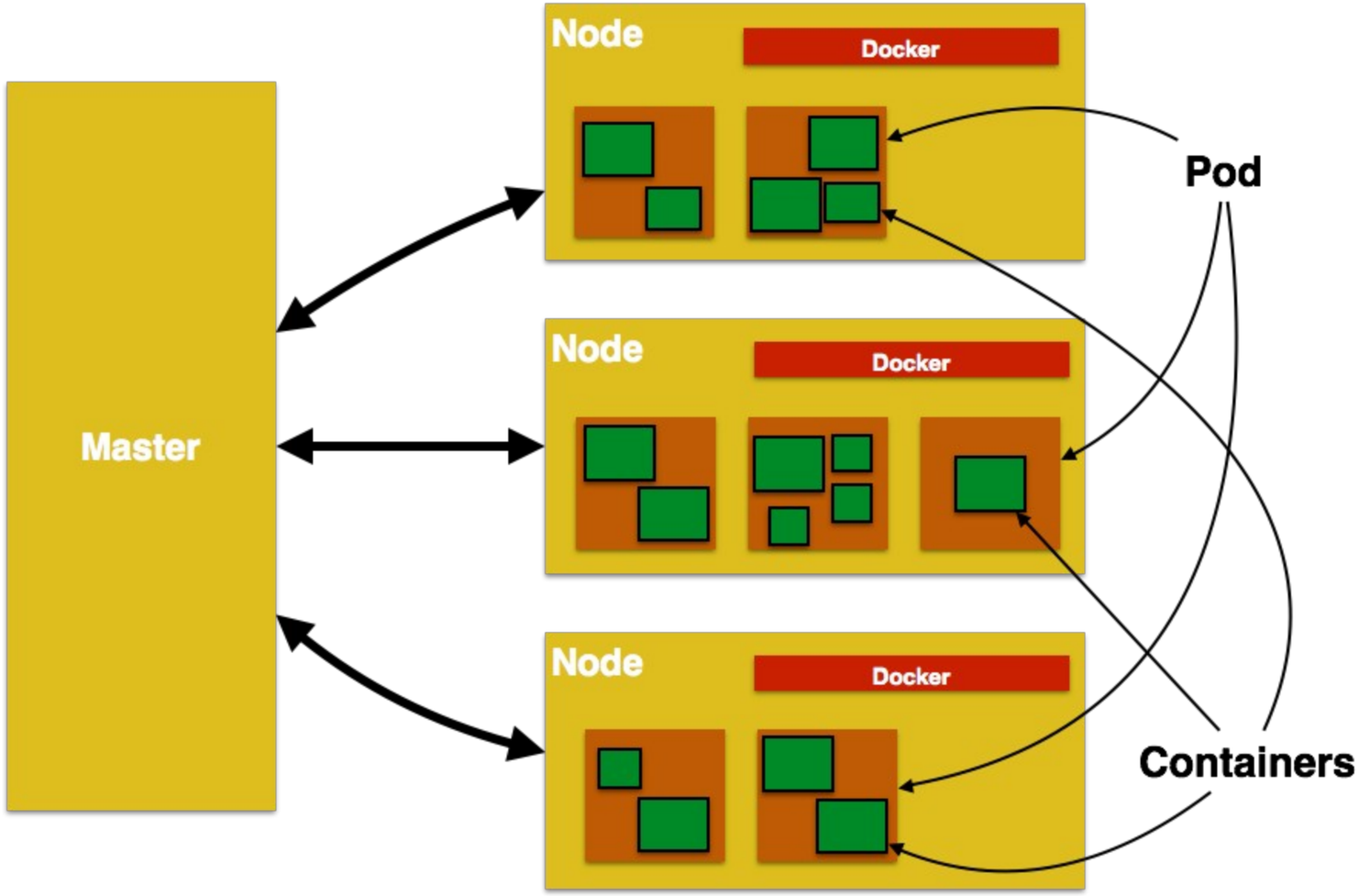
Kubernetes Terminology

- Minion – A Kubernetes node
- Pod
  - A group of containers running on the same host
  - Sharing resources (NIC, Private IP, File system)
  - The basic unit of management
- Replication controller – activates several similar pods
  - Supports auto scaling
- Service
  - The interface to a pod's containers: IP:PORT, private and (optionally) public
  - Round robin via a proxy daemon (per minion)

Supporting technologies

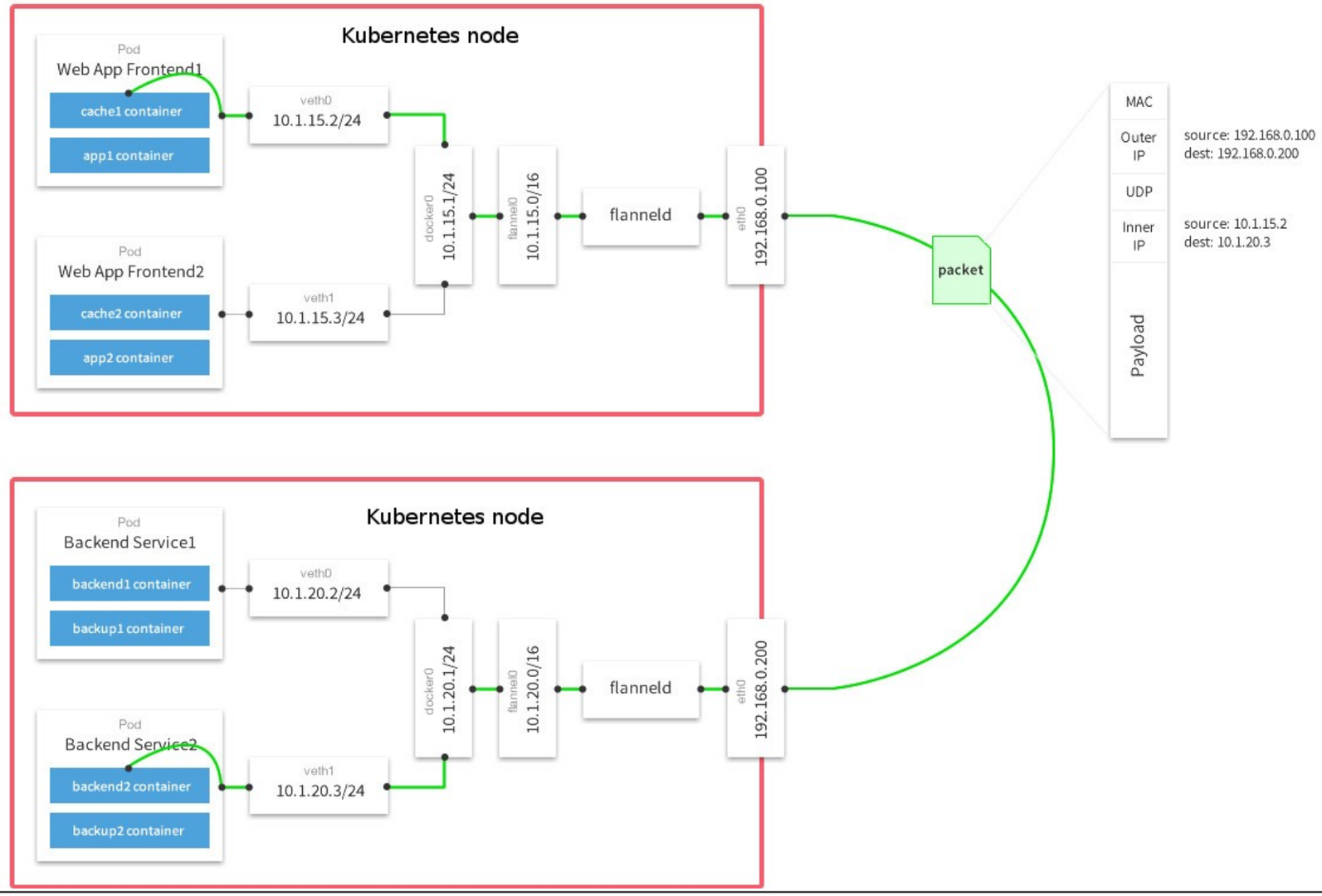
- etcd – distributed, consistent Key-Value store
  - Configuration database
- Flannel – virtual network management

Kubernetes high level



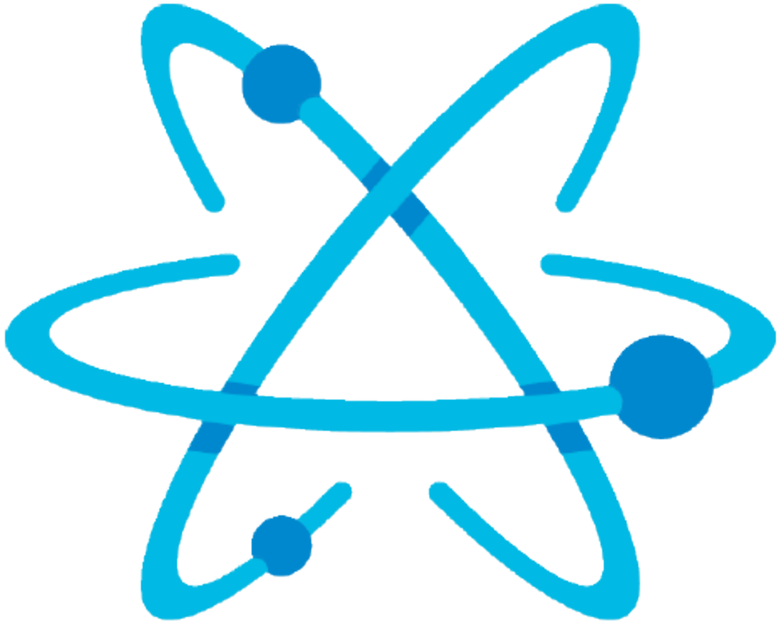
Kubernetes Networking

- Each Pod have a routable address.
- Each node runs flanneld – an agent to the Flannel virtual network
  - Flanneld allocates address inside a node
  - Subnet and address allocation are managed in etcd
  - Flannel0 is the GW for minion ↔ minion



RHEL Atomic

- Variation of RHEL 7
- Components
  - Systemd
  - Kubernetes
  - Docker
  - SELinux
- Manged like a firmware
  - OSTree vs. rpm
- Upstream project
  - http://www.projectatomic.io/



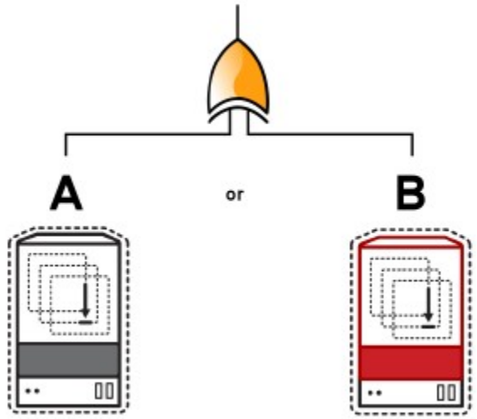
OSTree

- The key to “Atomic” upgrades / rollbacks
- Atomic should be treat like “firmware”

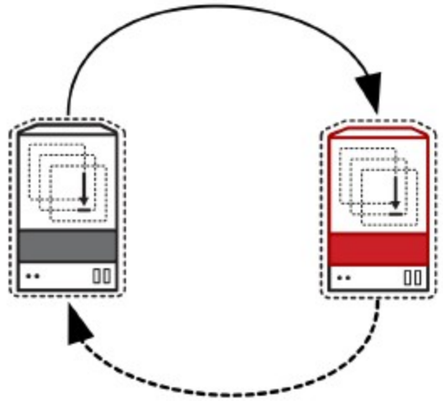
RED HAT ENTERPRISE LINUX ATOMIC HOST with OSTree

atomic upgrades and rollback for the operating system

OSTree: Atomic Operating System Changes

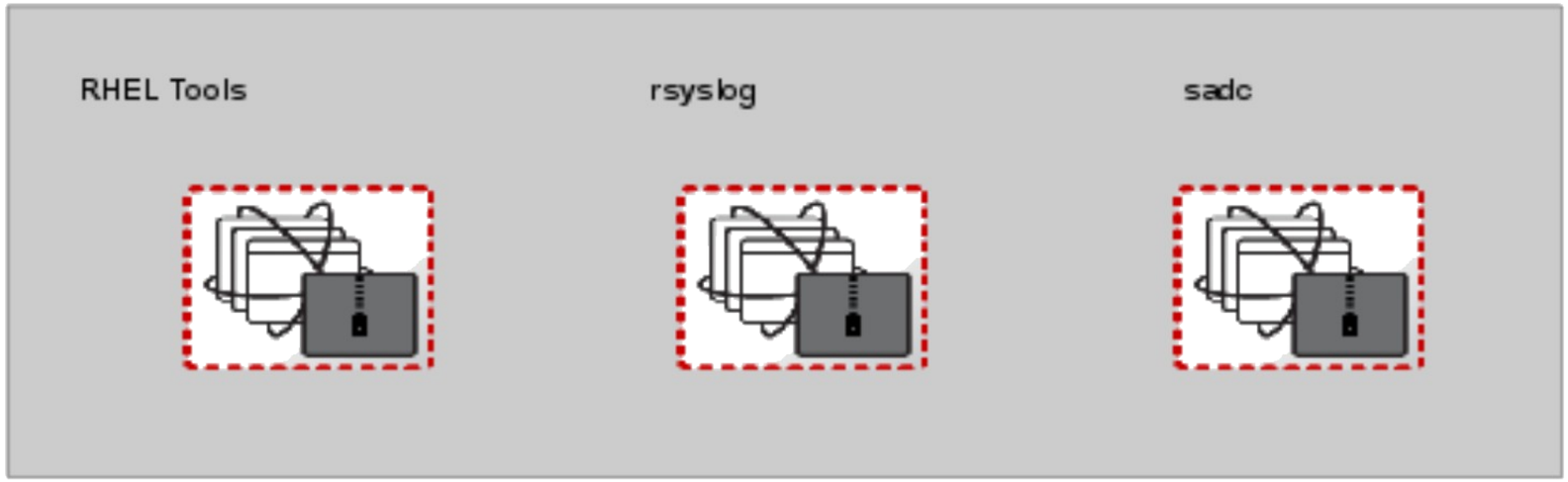


OSTree: Upgrade and Rollback



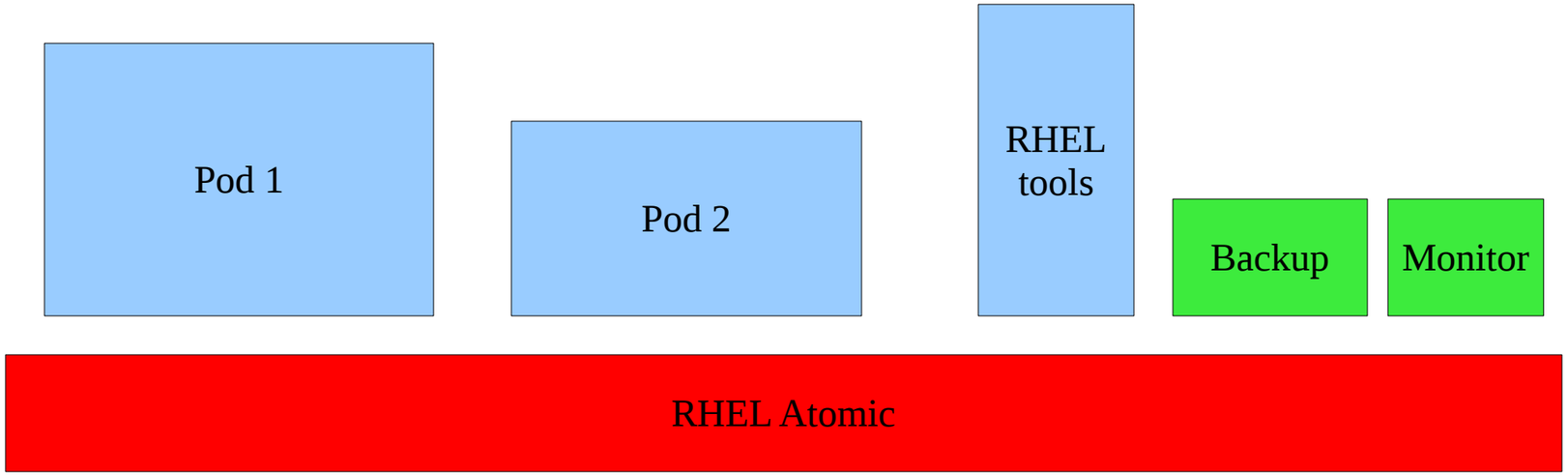
The new architecture

- Atomic is very small and “read only”
- Extra tools added with “privileged containers”



Holds for your toolchain as well

- Run operation tools in a privileged container
  - Backup agent / scripts
  - Monitoring and security tools



Deployment scenario

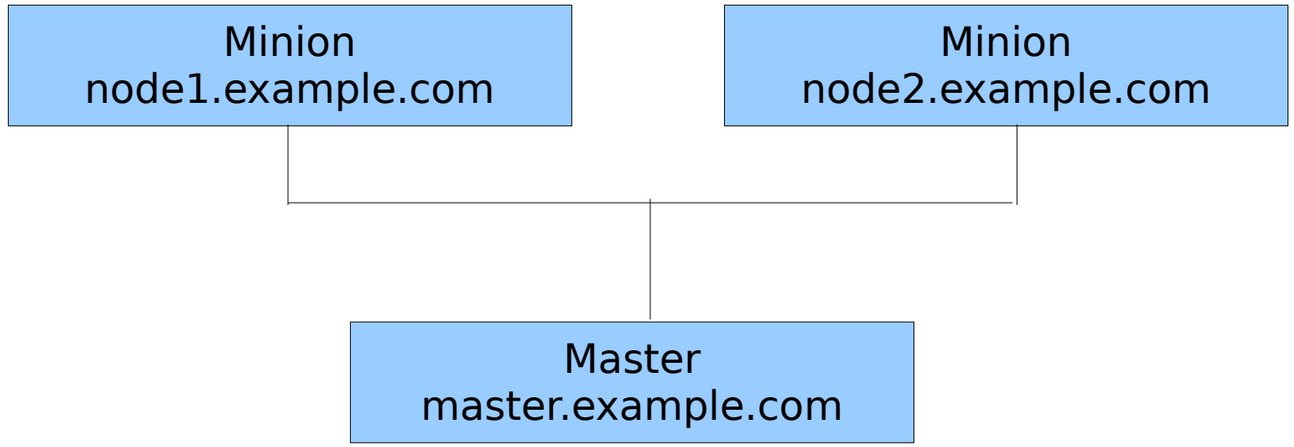
Deploy a simple Kubernetes Cluster

Notes

- This procedure assumes RHEL Atomic. For RHEL 7 there are some extra steps needed
- This technology is a rapidly evolving (A.K.A – a moving target)
- The presented configuration is correct today.

Topology

- If you try this at home your IP addresses may be somewhat difference
- **Master:**
  - flannel.1: 10.20.21.0/16
- **Node1:**
  - Flannel.1: 10.20.26.0/16
  - Docker0: 10.20.26.1/24
- **Node2:**
  - flannel.1: 10.20.37.0/16
  - docker0: 10.20.37.1/24



Prepare

- Kubernetes, etcd, flannel and docker are included with Atomic
- On master
  - Assign host name to master.example.com
  - Create an authentication key (required at the moment for v1):

    ```
    openssl genrsa -out /etc/kubernetes/serviceaccount.key 2048
    ```
- On nodes
  - Assign host names to node1.example.com and node2.example.com

Prepare and distribute docker images

- Assume a db and web server containers named `mydbforweb` and `mywebwithdb`
- On development machine (RHEL atomic is preinstalled with docker):

```
# docker export mydbforweb > dbforweb.tar
# docker export mywebwithdb > webwithdb.tar
# scp dbforweb.tar webwithdb.tar node1:/tmp
# scp dbforweb.tar webwithdb.tar node2:/tmp
```

- On nodes:

```
# cat /tmp/webwithdb.tar | docker import – webwithdb
# cat /tmp/dbforweb.tar | docker import – dbforweb

# docker images | grep web
dbforweb  latest  2e48be49fec4  About a minute ago  568.7 MB
webwithdb latest  5f2daa2e04e9  2 minutes ago       408.6 MB
```

Configure etcd and Kubernetes on master

- Edit /etc/etcd/etcd.conf

```
ETCD_NAME=default
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://localhost:2380,http://localhost:7001"
ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:4001,http://0.0.0.0:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://0.0.0.0:4001,http://0.0.0.0:2379"
```

- Edit /etc/kubernetes/apiserver

```
KUBE_ETCD_SERVERS="--etcd_servers=http://master.example.com:4001"
KUBE_API_ADDRESS="--address=0.0.0.0"
KUBE_SERVICE_ADDRESSES="--portal_net=10.254.0.0/16"
KUBE_API_ARGS="--service_account_key_file=/etc/kubernetes/serviceaccount.key"
```

- Edit /etc/kubernetes/controller-manger

```
KUBE_CONTROLLER_MANAGER_ARGS="--service_account_private_key_file=/etc/kubernetes/serviceaccou
nt.key"
```

- Edit /etc/kubernetes/config

```
KUBE_MASTER="--master=http://master.example.com:8080"
```

Configure etcd and Kubernetes on nodes

- Edit /etc/kubernetes/kubelet

```
KUBELET_ADDRESS="—address=0.0.0.0"
KUBELET_ARGS="--register-node=true"
KUBELET_API_SERVER="--api_servers=http://master.example.com:8080"
# Optionally: KUBELET_HOSTNAME="--hostname_override=node2.example.com"
```

- Edit /etc/kubernetes/config

```
KUBE_MASTER="--master=http://master.example.com:8080"
```

Start services

- On master:

```
# for SERVICES in etcd kube-apiserver kube-controller-manager
kube-scheduler; do
   systemctl restart $SERVICES
   systemctl enable $SERVICES
   systemctl status $SERVICES
 done
```

- On nodes:

```
# for SERVICES in docker kube-proxy.service kubelet.service; do
   systemctl restart $SERVICES
   systemctl enable $SERVICES
   systemctl status $SERVICES
 done
```

Sanity tests

- Check that etcd works

```
# curl -s -L http://master.example.com:4001/version
etcd 2.0.11
```

- Check the nodes

```
# kubectl get nodes
NAME                LABELS                                    STATUS
node1.example.com kubernetes.io/hostname=node1.example.com Ready
node2.example.com kubernetes.io/hostname=node2.example.com Ready
```

Set up flannel

- Create a configuration document – for instance in flannel.json:

```
{ "Network": "10.20.0.0/16",
  "SubnetLen": 24,
  "Backend": { "Type": "vxlan", "VNI": 1 }
}
```

- Upload to etcd

```
# etcdctl set coreos.com/network/config < flannel.json
```

- Configure flanneld – edit /etc/sysconfig/flanneld

```
FLANNEL_ETCD="http://master.example.com:4001"
FLANNEL_ETCD_KEY="/coreos.com/network"
FLANNEL_OPTIONS="-iface=enp0s8"
```

- Enable the flanneld service and reboot nodes

Set up pods and services

- db-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: db
  name: db-service
  namespace: default
spec:
  ports:
  - port: 3306
  selector:
    name: db
```

- db-rc.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    name: db-controller
  name: db-controller
  namespace: default
spec:
  replicas: 2
  selector:
    selectorname: db
  template:
   metadata:
      labels:
        name: db
        selectorname: db
   spec:
      containers:
      - image: dbforweb
        command:
["/usr/bin/mysqld_safe"]
        args: ["--basedir=/usr"]
        name: db
        ports:
        - containerPort: 3306
```

- ws-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: webserver
  name: webserve-service
  namespace: default
spec:
  ports:
  - port: 80
  publicIPs:
  - 192.168.122.15
  selector:
    name: webserver
```

- ws-rc.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    name: webserver-controller
  name: webserver-controller
  namespace: default
spec:
  replicas: 2
  selector:
    selectorname: webserver
  template:
    metadata:
      labels:
        name: webserver
        selectorname: webserver
        uses: db
    spec:
      containers:
      - image: webwithdb
        name: apache-frontend
        command: ["/usr/sbin/httpd"]
        args: ["-D", "FOREGROUND"]
        ports:
        - containerPort: 80
```

Deploy services and controllers

- Use the files from the previous step

```
# kubectl create -f db-service.yaml
# kubectl create -f db-rc.yaml
# kubectl create -f ws-service.yaml
# kubectl create -f ws-rc.yaml
```

- Check

```
# kubectl get services
NAME              LABELS                                    SELECTOR            IP(S)           PORT(S)
db-service        name=db                                   name=db             10.254.192.67   3306/TCP
kubernetes        component=apiserver,provider=kubernetes                       10.254.255.128  443/TCP
kubernetes-ro     component=apiserver,provider=kubernetes                       10.254.139.205  80/TCP
webserver-service name=webserver                            name=webserver      10.254.134.105  80/TCP
```

Access the service

- Find the IP and use curl to get a page

```
# kubectl get pod | grep webwithdb
47f0d628-edc2-11e4-8ee6-5254001aa4ee   10.20.11.3   apache-frontend
webwithdb   node1.example.com/
      name=webserver,selectorname=webserver,uses=db   Running   2 hours


# curl http://10.20.11.3/index.html
The Web Server is Running


# curl http://10.20.11.3/cgi-bin/action
<html>
<head>
<title>My Application</title>
</head>
<body>
<h2>Success</h2>
</body>
</html>
```

Complex?

A word about Openshift v3

OpenShift v3 is based on Kubernetes

- With the following extensions:
  - Internal networking with Open vSwitch
    - Better the flannel for high latency (WAN) communication
  - Transparent service access with service URL
    - Automates minion IP / port access to service. Simply use URLs
  - Multi tenant
  - Source to image automation
    - Push source to git
    - Run unit tests
    - Build image
    - Upload to the registry

redhat®

yoel@emet.co.il