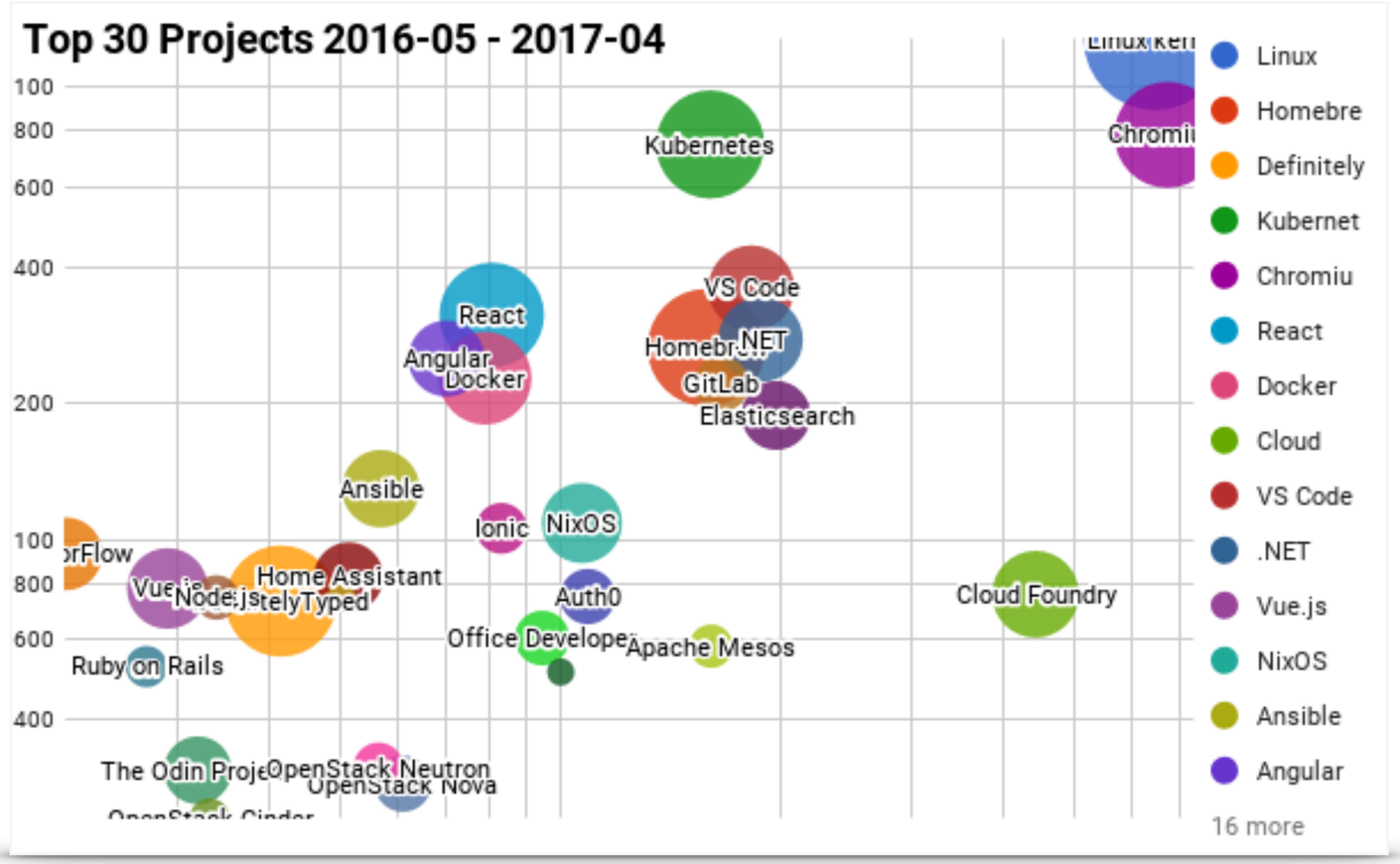
Kubernetes Walk-through

by Harry Zhang @resouer

Kubernetes

- Created by Google Borg/Omega team
- Founded and operated by CNCF (Linux Foundation)
- Container orchestration, scheduling and management
- One of the most popular open source project in the world

Project State

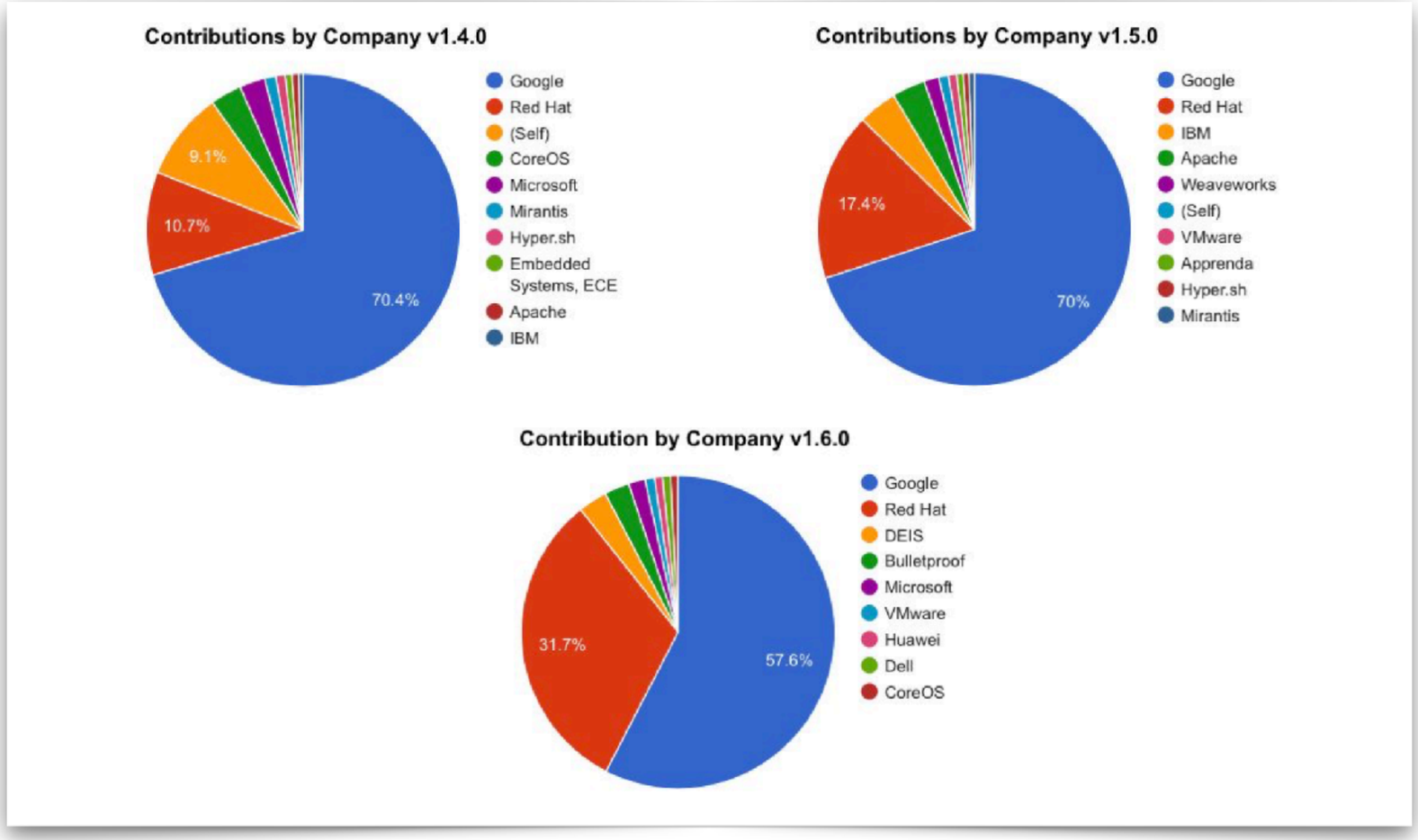


Rank	Labe
1	Linux kernel
2	Homebrew
3	DefinitelyTyped
4	Kubernetes
5	Chromium
6	React
7	Docker
8	Cloud Foundry
9	VS Code
10	.NET
11	Vue.js
12	NixOS
13	Ansible
14	Angular
15	TensorFlow
16	Home Assistant
17	Elasticsearch
18	The Odin Project
19	OpenStack Nova
20	Auth0
21	GitLab
22	Office Developer
23	Ionic
24	OpenStack Neutron
25	Node.js
26	Apache Mesos
27	Ruby on Rails
28	OpenStack Cinder
29	Terraform
30	Chef

Data source: CNCF blog

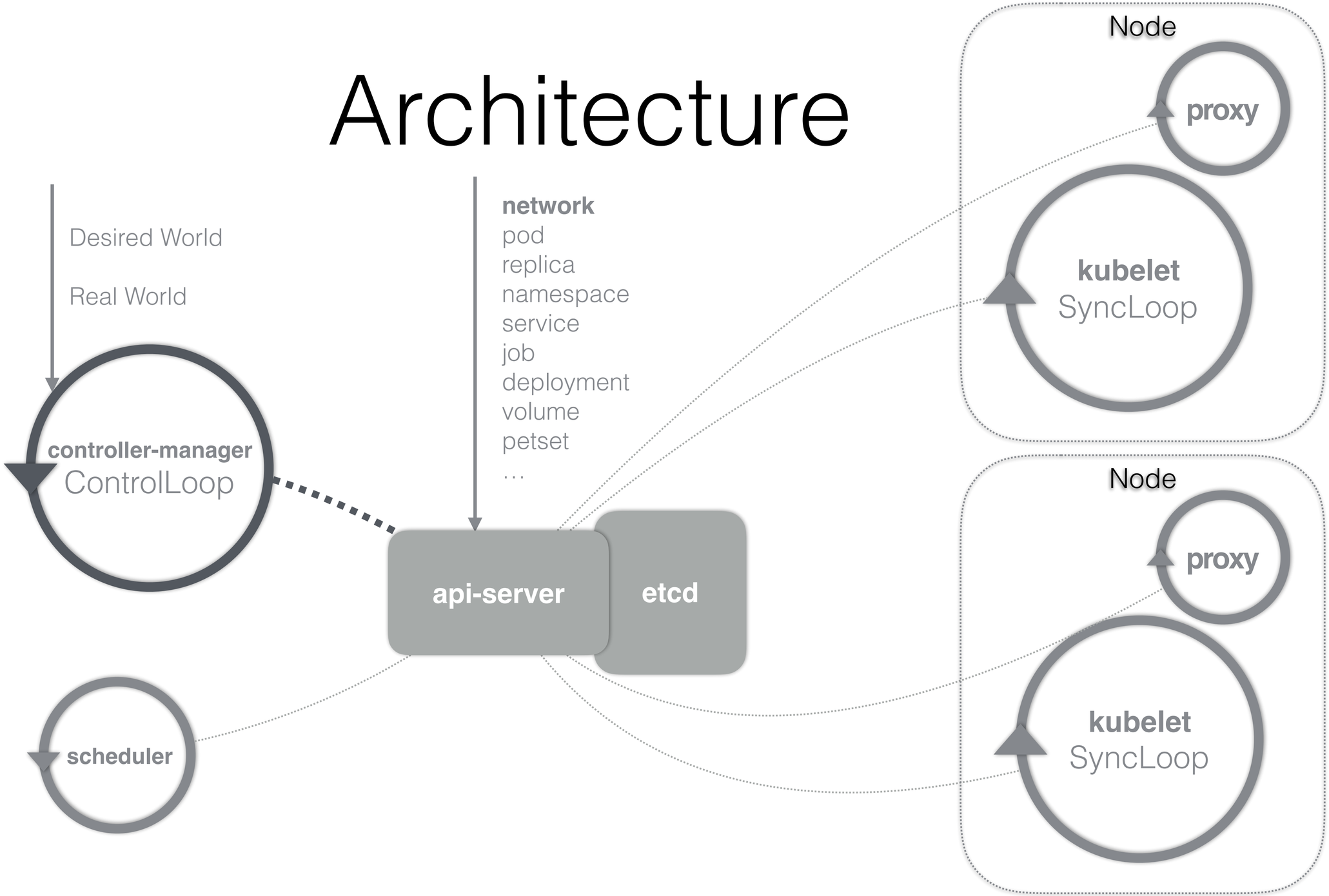
Growing Contributors

- 1728+ authors

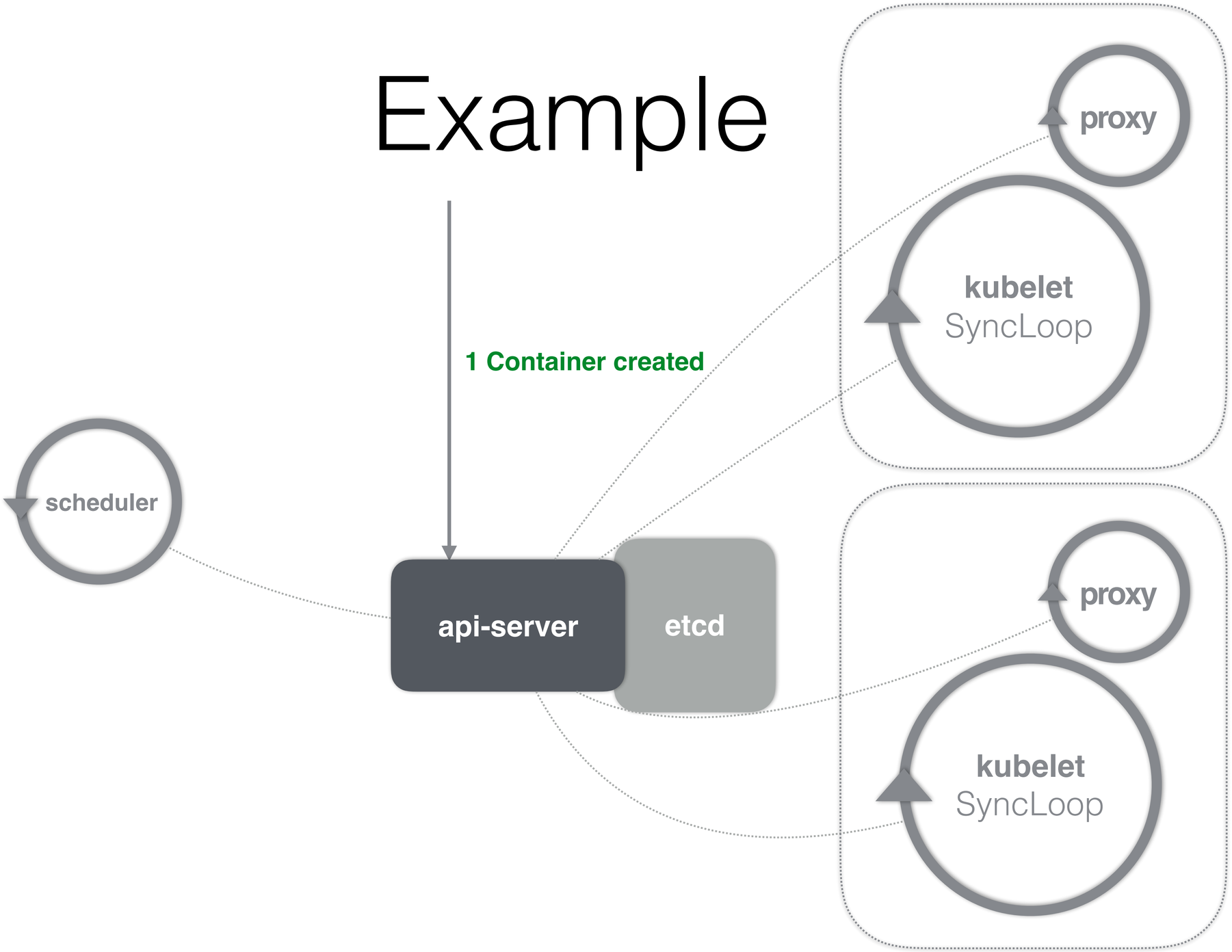


Data source: Kubernetes Leadership Summit

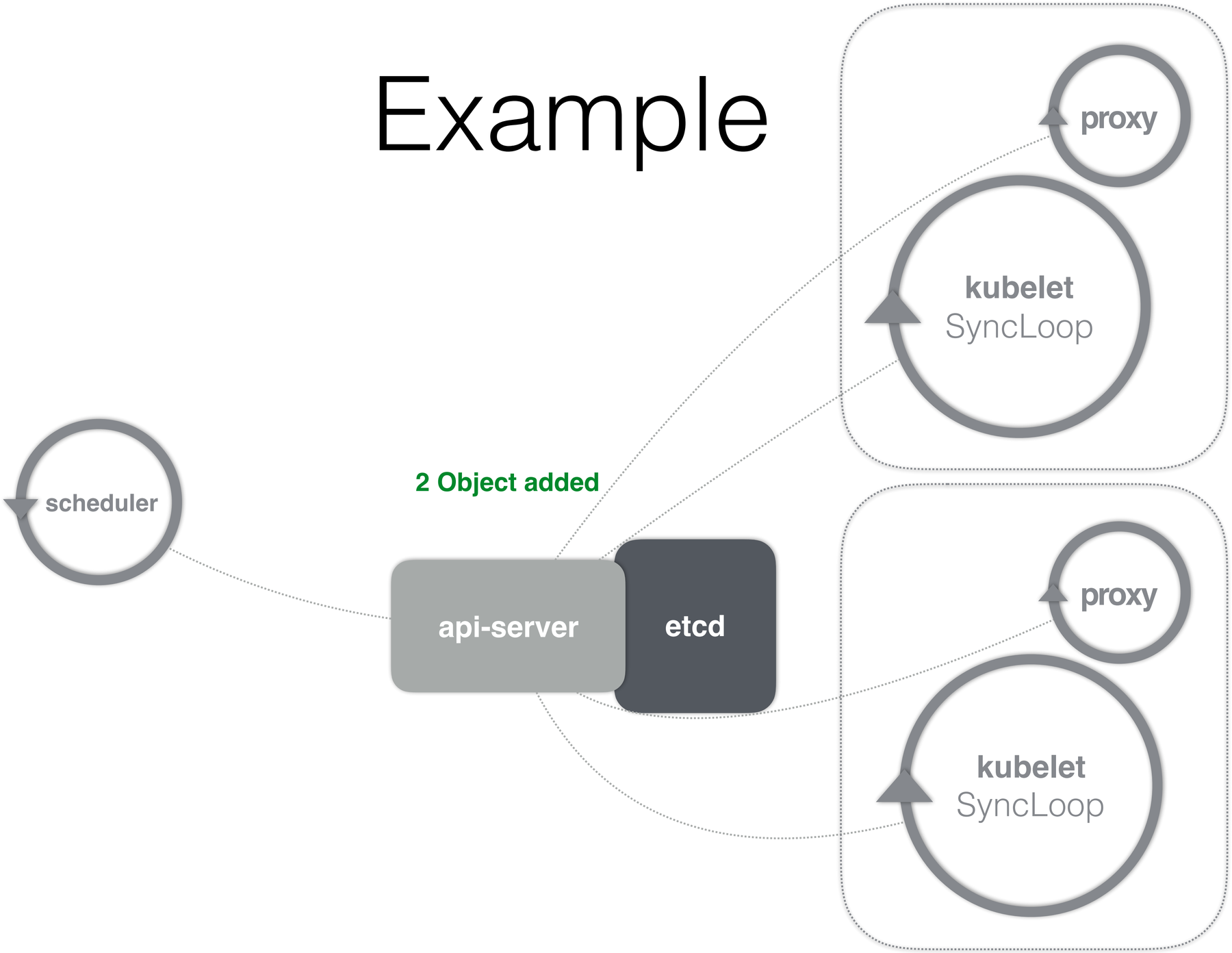
Architecture



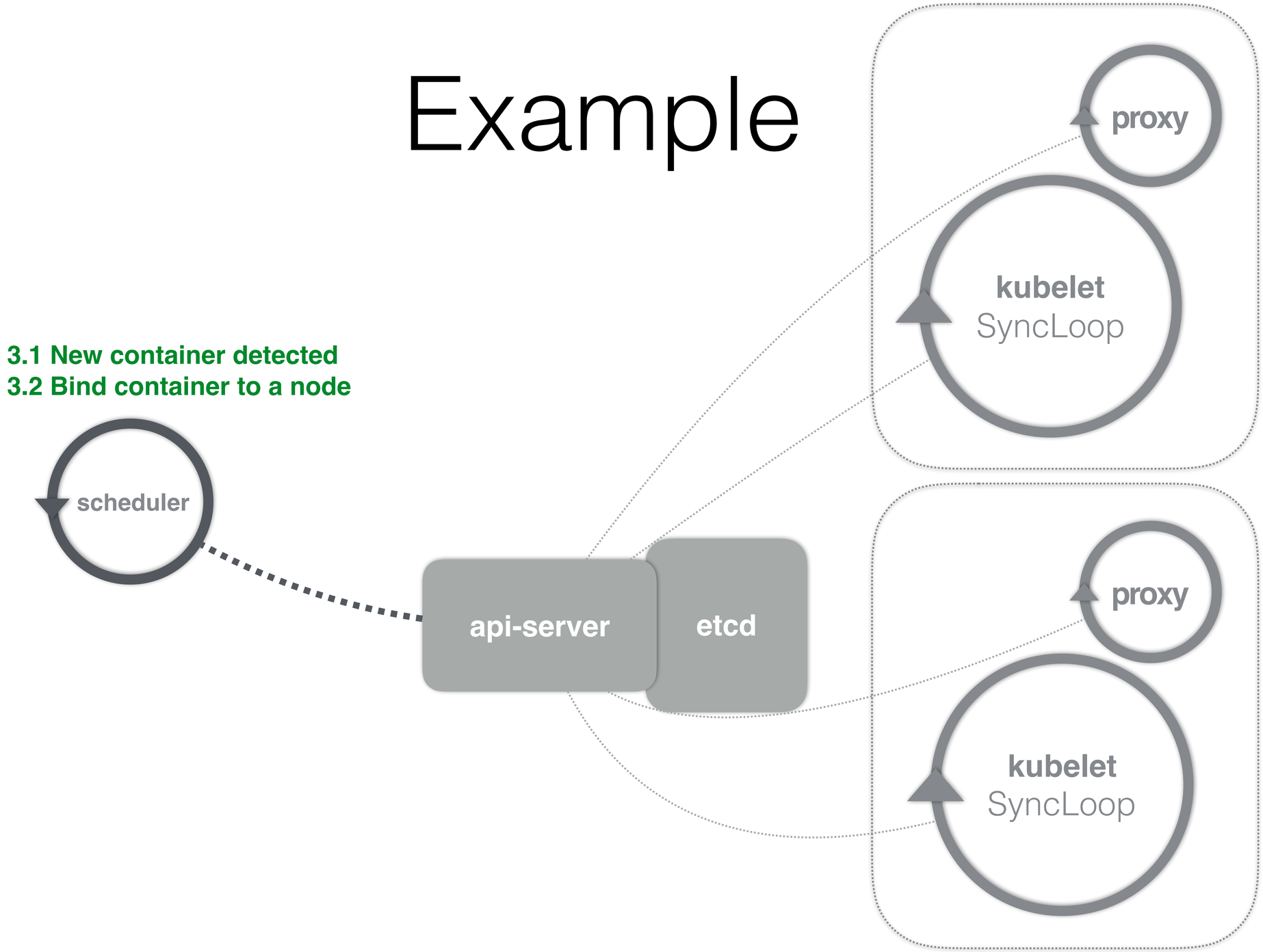
Example


1 Container created
scheduler
api-server
etcd
proxy
kubelet
SyncLoop
proxy
kubelet
SyncLoop

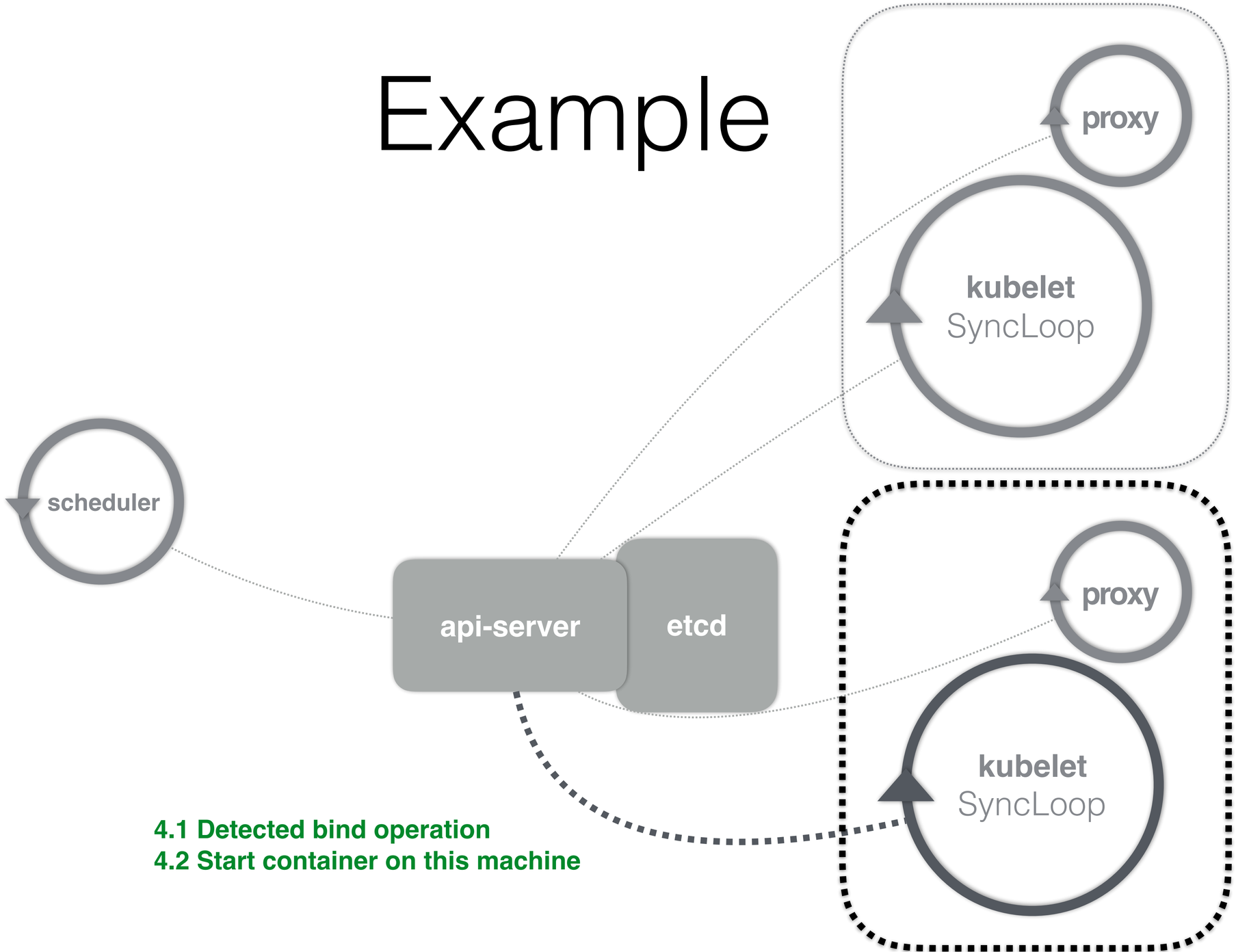
Example


proxy
kubelet
SyncLoop
scheduler
2 Object added
api-server
etcd
proxy
kubelet
SyncLoop

Example


3.1 New container detected
3.2 Bind container to a node
scheduler
api-server
etcd
proxy
kubelet
SyncLoop
proxy
kubelet
SyncLoop

Example



4.1 Detected bind operation
4.2 Start container on this machine

Take Aways

- Independent control loops
  - loosely coupled
  - high performance
  - **easy to customize and extend**
- “Watch” object change
- Decide next step based on state change
  - not edge driven (event), level driven (state)

{Pod} = a group of containers

Co-scheduling

- Tow containers:
  - **App:** generate log files
  - **LogCollector:** read and redirect logs to storage
- Request MEM:
  - **App**: 1G
  - **LogCollector**: 0.5G
- Available MEM:
  - Node_A: 1.25G
  - Node_B: 2G
- What happens if **App** is scheduled to Node_A first?

Pod

- Deeply coupled containers
- Atomic scheduling/placement unit
- Shared namespace
  - network, IPC etc
- Shared volume
- Process group in container cloud

Why co-scheduling?

- It’s about using container in **right** way:
  - Lesson learnt from Borg: “workloads tend to have tight relationship”

Ensure Container Order

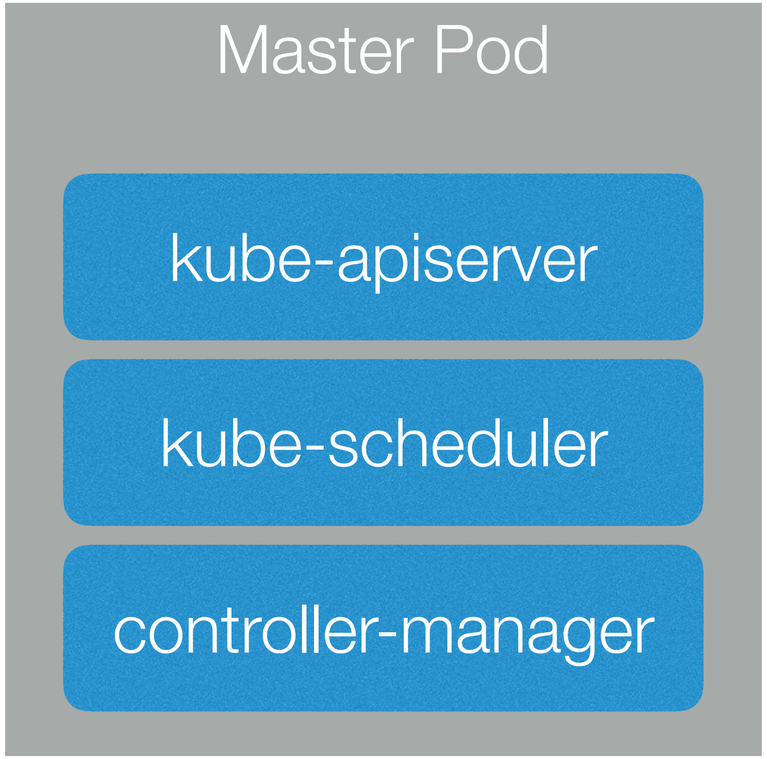
- Decouple web server and application
  - war file container
  - tomcat container

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: javaweb-2
spec:
  initContainers:
  - name: war
    image: resouer/sample:v2
    command: ["cp", "/sample.war", "/app"]
    volumeMounts:
    - mountPath: /app
      name: app-volume
  containers:
  - name: tomcat
    image: resouer/mytomcat:7.0
    command: ["sh","-c","/root/apache-tomcat-7.0.42-v2/bin/start.sh"]
    volumeMounts:
    - mountPath: /root/apache-tomcat-7.0.42-v2/webapps
      name: app-volume
    ports:
    - containerPort: 8080
      hostPort: 8001
  volumes:
  - name: app-volume
    emptyDir: {}
```

Multiple Apps in One Container?

- Wrong!

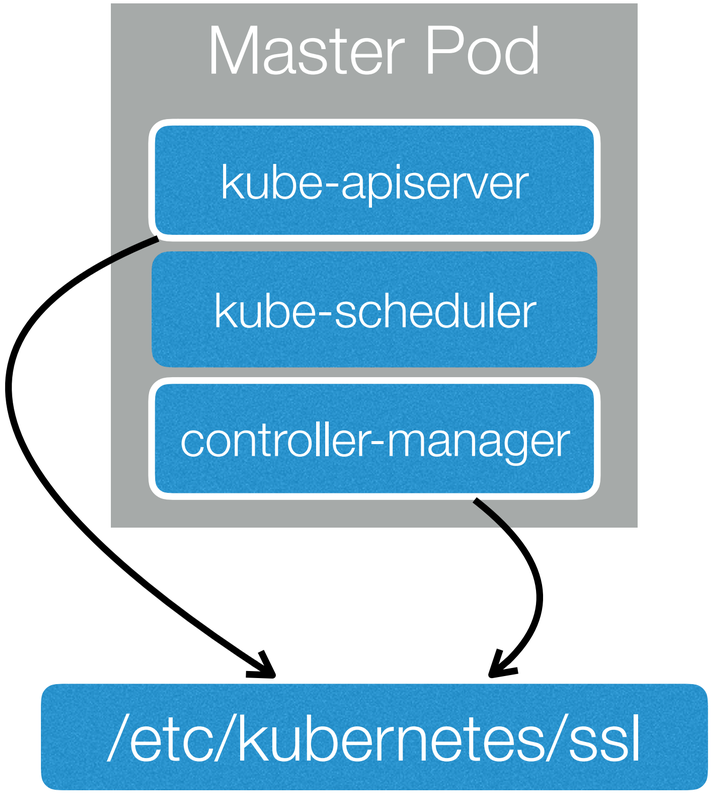
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
spec:
  containers:
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
  - name: "kube-apiserver"
    image: kube-apiserver:v1.6.0
```



Copy Files from One to Another?

- Wrong!

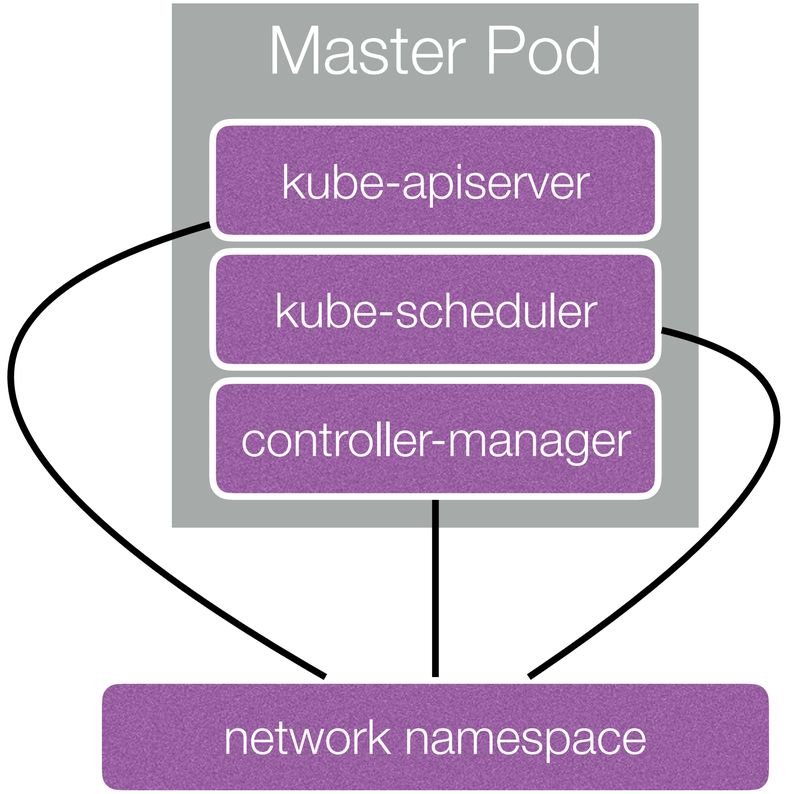
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
spec:
  containers:
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
    volumeMounts:
    - mountPath: /etc/kubernetes/ssl
      name: ssl-certs-kubernetes
      readOnly: true
  - name: "kube-apiserver"
    image: kube-apiserver:v1.6.0
    volumeMounts:
      - mountPath: /etc/kubernetes/ssl
        name: ssl-certs-kubernetes
        readOnly: true
  volumes:
  - hostPath:
      path: /etc/kubernetes/ssl
    name: ssl-certs-kubernetes
```



Connect to Peer Container thru IP?

- Wrong!

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: k8s-master-pod
  labels:
    app: k8s-master-pod
spec:
  containers:
  - name: "kube-apiserver"
    image: kube-apiserver:v1.6.0
    command:
    - apiserver
    - --bind-address=0.0.0.0
    - --insecure-port=8080
  - name: "kube-scheduler"
    image: kube-scheduler:v1.6.0
    command:
    - scheduler
    - --master=http://127.0.0.1:8080
  - name: "kube-controller-manager"
    image: kube-controller-manager:v1.6.0
    command:
    - controller-manager
    - --master=http://127.0.0.1:8080
```



So this is Pod

- Design pattern in container world
  - decoupling
  - reuse & refactoring
- Describe more real-world workloads by container
  - e.g. ML
    - Parameter server and trainer in same Pod

Kubernetes **Control Panel**

1. How Kubernetes schedule workloads?

Resource Model

- **Compressible resources**
  - Hold no state
  - Can be taken away very quickly
  - “Merely” cause slowness when revoked
    - e.g. CPU
- **Non-compressible resources**
  - Hold state
  - Are slower to be taken away
  - Can fail to be revoked
    - e.g. Memory, disk space

Kubernetes (and Docker) can only handle CPU & Memory

Don’t handle things like memory bandwidth, disk time, cache, network bandwidth, ... (yet)

Resource Model

- **Request**: amount of a resource allowed to be used, with a strong guarantee of availability
  - CPU (seconds/second), RAM (bytes)
  - Scheduler will not over-commit requests
- **Limit**: max amount of a resource that can be used, regardless of guarantees
  - scheduler **ignores** limits

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

- Mapping to Docker
  - —cpu-shares=requests.cpu
  - —cpu-quota=limits.cpu
  - —cpu-period=100ms
  - —memory=limits.memory

QoS Tiers and Eviction

- **Guaranteed**
  - limits is set for all resources, all containers
  - limits == requests (if set)
  - Be killed **until they exceed their limits**
    - or if the system is under memory pressure and there are no lower priority containers that can be killed.
- **Burstable**
  - requests is set for one or more resources, one or more containers
  - limits (if set) != requests
  - killed once they **exceed** their requests and **no Best-Effort pods exist** when system under memory pressure
- **Best-Effort**
  - requests and limits are not set for all of the resources, all containers
  - **First to get killed** if the system runs out of memory

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Scheduler

- Predicates
  - **NoDiskConflict**
  - NoVolumeZoneConflict
  - **PodFitsResources**
  - **PodFitsHostPorts**
  - **MatchNodeSelector**
  - MaxEBSVolumeCount
  - MaxGCEPDVolumeCount
  - **CheckNodeMemoryPressure**
    - eviction, QoS tiers
  - **CheckNodeDiskPressure**

- Priorities
  - **LeastRequestedPriority**
  - BalancedResourceAllocation
  - **SelectorSpreadPriority**
  - CalculateAntiAffinityPriority
  - **ImageLocalityPriority**
  - NodeAffinityPriority

- **Design tips:**
  - watch and sync podQueue
  - schedule based on cached info
  - optimistically bind
  - predicates is paralleled between nodes
  - priorities are paralleled between functions in Map-Reduce way

Multi-Scheduler

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    component: scheduler
    tier: control-plane
  name: my-scheduler
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        component: scheduler
        tier: control-plane
        version: second
    spec:
      containers:
      - command: [/usr/local/bin/kube-scheduler, --address=0.0.0.0,
                  --scheduler-name=my-scheduler]
        image: gcr.io/my-gcp-project/my-kube-scheduler:1.0
        livenessProbe:
          httpGet:
            path: /healthz
            port: 10251
          initialDelaySeconds: 15
```

The 2nd scheduler

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: annotation-second-scheduler
  annotations:
    scheduler.alpha.kubernetes.io/name: my-scheduler
  labels:
    name: multischeduler-example
spec:
  containers:
  - name: pod-with-second-annotation-container
    image: gcr.io/google_containers/pause:2.0
```

- Tips: **annotation**: system usage labels
  - Do **NOT** abuse labels

2. Workload management?

Deployment

- **Replicas with control**
  - **Bring up** a Replica Set and Pods.
  - **Check** the status of a Deployment.
  - **Update** that Deployment (e.g. **new image, labels**).
  - **Rollback** to an earlier Deployment revision.
  - **Pause** and **resume** a Deployment.

Create

```
$ kubectl create -f docs/user-guide/nginx-deployment.yaml --record
deployment "nginx-deployment" created
```

- **ReplicaSet**
  - Next generation of ReplicaController
    - —record: record command in the annotation of ‘nginx-deployment’

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Check

```
$ kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           18s
```

- **DESIRED**: *.spec.replicas*
- **CURRENT**: *.status.replicas*
- **UP-TO-DATE**: contains the latest pod template
- **AVAILABLE:** pod status is ready (running)

Update

- **kubectl set image**
  - will change container image
- **kubectl edit**
  - open an editor and modify your deployment yaml

```
$ kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
deployment "nginx-deployment" image updated
```

```
$ kubectl edit deployment/nginx-deployment
deployment "nginx-deployment" edited
```

trigger

```
$ kubectl rollout status deployment/nginx-deployment
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
deployment nginx-deployment successfully rolled out
```

```
$ kubectl get rs
NAME                          DESIRED   CURRENT   AGE
nginx-deployment-1564180365   3         3         6s
nginx-deployment-2035384211   0         0         36s
```

- RollingUpdateStrategy
  - 1 max unavailable
  - 1 max surge
  - can also be percentage
- Does not kill old Pods until a sufficient number of new Pods have come up
- Does not create new Pods until a sufficient number of old Pods have been killed.

Update Process

- The update process is coordinated by Deployment Controller

```
NewReplicaSet:      nginx-deployment-1564180365 (3/3 replicas created)
Events:
  FirstSeen LastSeen    Count   From                    SubobjectPath   Type        Reason              Message
  --------- --------    -----   ----                    -------------   --------    ------              -------
  36s       36s         1       {deployment-controller }                Normal      ScalingReplicaSet   Scaled up replica set nginx-deployment-2035384211 to 3
  23s       23s         1       {deployment-controller }                Normal      ScalingReplicaSet   Scaled up replica set nginx-deployment-1564180365 to 1
  23s       23s         1       {deployment-controller }                Normal      ScalingReplicaSet   Scaled down replica set nginx-deployment-2035384211 to 2
  23s       23s         1       {deployment-controller }                Normal      ScalingReplicaSet   Scaled up replica set nginx-deployment-1564180365 to 2
  21s       21s         1       {deployment-controller }                Normal      ScalingReplicaSet   Scaled down replica set nginx-deployment-2035384211 to 0
  21s       21s         1       {deployment-controller }                Normal      ScalingReplicaSet   Scaled up replica set nginx-deployment-1564180365 to 3
```

- **Create**: Replica Set (nginx-deployment-2035384211) and scaled it up to 3 replicas directly.
- **Update**:
  - created a new Replica Set (nginx-deployment-1564180365) and scaled it up to 1
  - scaled down the old Replica Set to 2
    - continued scaling up and down the new and the old Replica Set, with the same rolling update strategy.
  - Finally, 3 available replicas in the new Replica Set, and the old Replica Set is scaled down to 0.

Rolling Back

- Check reversions

```
$ kubectl rollout history deployment/nginx-deployment
deployments "nginx-deployment":
REVISION    CHANGE-CAUSE
1           kubectl create -f docs/user-guide/nginx-deployment.yaml --record
2           kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
3           kubectl set image deployment/nginx-deployment nginx=nginx:1.91
```

```
$ kubectl rollout history deployment/nginx-deployment --revision=2
deployments "nginx-deployment" revision 2
  Labels:       app=nginx
          pod-template-hash=1159850644
  Annotations:  kubernetes.io/change-cause=kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
  Containers:
   nginx:
    Image:      nginx:1.9.1
    Port:       80/TCP
     QoS Tier:
        cpu:      BestEffort
        memory:   BestEffort
    Environment Variables:      <none>
  No volumes.
```

- Roll back to reversion

```
$ kubectl rollout undo deployment/nginx-deployment --to-revision=2
deployment "nginx-deployment" rolled back
```

Pausing & Resuming (Canary)

```
$ kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1; kubectl rollout pause deployment/nginx-deployment
deployment "nginx-deployment" image updated
deployment "nginx-deployment" paused
```

```
$ kubectl rollout status deployment/nginx-deployment
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
```

```
$ kubectl rollout resume deployment/nginx-deployment
deployment "nginx-deployment" resumed
```

- Tips
  - blue-green deployment: duplicated infrastructure
  - canary release: share same infrastructure
  - rollback resumed deployment is WIP
  - old way: kubectl rolling-update rc-1 rc-2

3. Deploy Daemon workload to every Node?

DaemonSet

- Spread daemon pod to every node
- DaemonSet Controller
  - bypass default scheduler
    - even on *unschedulable* nodes
      - e.g. bootstrap

```yaml
kind: DaemonSet
apiVersion: extensions/v1beta1
metadata:
  name: calico-node
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    k8s-app: calico-node
 spec:
    hostNetwork: true
    containers:
      - name: calico-node
        image: quay.io/calico/node:v0.21.0
        env:
          - name: ETCD_ENDPOINTS
            value: "http://10.0.0.17:6666"
          - name: CALICO_NETWORKING
            value: "false"
        securityContext:
          privileged: true
```

4. Automatically scale?

Horizontal Pod Autoscaling

```
$ kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
deployment "php-apache" autoscaled
```

```
$ kubectl get hpa
NAME         REFERENCE                     TARGET    CURRENT   MINPODS   MAXPODS   AGE
php-apache   Deployment/php-apache/scale   50%       305%      1         10        3m
```

```
$ kubectl get deployment php-apache
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
php-apache   7         7         7            7           19m
```

```yaml
apiVersion: extensions/v1beta1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleRef:
    kind: Deployment
    name: php-apache
    subresource: scale
  minReplicas: 1
  maxReplicas: 10
  cpuUtilization:
    targetPercentage: 50
```

- Tips
  - Scale out/in
  - TriggeredScaleUp (GCE, AWS, will add more)
  - Support for **custom metrics**

```yaml
annotations:
  alpha/target.custom-metrics.podautoscaler.kubernetes.io: '{"items":[{"name":"qps", "value": "10"}]}'
```

Custom Metrics

- Endpoint (Location to collect metrics from)
- Name of metric
- Type (Counter, Gauge, ...)
- Data Type (int, float)
- Units (kbps, seconds, count)
- Polling Frequency
- Regexps (Regular expressions to specify which metrics to collect and how to parse them)
- The metric will be added to pod as **ConfigMap** volume

```
{
  "endpoint" : "http://localhost:9100/metrics"
}
```

Prometheus

```
{
  "endpoint" : "http://localhost:8000/nginx_status",
  "metrics_config" : [
    {
      "name" : "activeConnections",
      "metric_type" : "gauge",
      "units" : "number of active connections",
      "data_type" : "int",
      "polling_frequency" : 10,
      "regex" : "Active connections: ([0-9]+)"
    },
    {
      "name" : "reading",
      "metric_type" : "gauge",
      "units" : "number of reading connections",
      "data_type" : "int",
      "polling_frequency" : 10,
      "regex" : "Reading: ([0-9]+) .*"
    }
  ]
}
```

Nginx

5. Pass information to workloads?

ConfigMap

```
$ ls docs/user-guide/configmap/kubectl/
game.properties
ui.properties

$ kubectl create configmap game-config --from-file=docs/user-guide/configmap/kubectl
```

- Decouple configuration from image
  - configuration is a runtime attribute
- Can be consumed by pods thru:
  - env
  - volumes

```
$ kubectl get configmaps game-config -o yaml
```

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default
  resourceVersion: "407"-
  selfLink: /api/v1/namespaces/default/configmaps/game-config
  uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

ConfigMap Volume

```
$ kubectl create configmap example-redis-config --from-file=docs/user-guide/configmap/redis/redis-config

$ kubectl get configmap example-redis-config -o yaml
```

```
apiVersion: v1
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
kind: ConfigMap
metadata:
  creationTimestamp: 2016-03-30T18:14:41Z
  name: example-redis-config
  namespace: default
  resourceVersion: "24686"
  selfLink: /api/v1/namespaces/default/configmaps/example-redis-config
  uid: 460a2b6e-f6a3-11e5-8ae5-42010af00002
```

```
  volumeMounts:
  - mountPath: /redis-master-data
    name: data
  - mountPath: /redis-master
    name: config
volumes:
  - name: data
    emptyDir: {}
  - name: config
    configMap:
      name: example-redis-config
      items:
      - key: redis-config
        path: redis.conf
```

- **No need** to use Persistent Volume
- Think about Etcd

Secret

```
$ kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
secret "db-user-pass" created
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: MWYyZDF1MmU2N2Rm
  username: YWRtaW4=
```

- Tip: credentials for accessing the k8s API is automatically added to your pods as secret

```yaml
spec:
  containers:
    - name: mycontainer
      image: redis
      env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: password
```

```
"spec": {
  "containers": [{
    "name": "mypod",
    "image": "redis",
    "volumeMounts": [{
      "name": "foo",
      "mountPath": "/etc/foo",
      "readOnly": true
    }]
  }],
  "volumes": [{
    "name": "foo",
    "secret": {
      "secretName": "mysecret"
    }
```

6. Read information from system itself?

Downward Api

- Get these inside your pod as ENV or volume
  - The pod's name
  - The pod's namespace
  - The pod's IP
  - A container's cpu limit
  - A container's cpu request
  - A container's memory limit
  - A container's memory request

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: MY_POD_IP
          valueFrom:
            fieldRef:
              fieldPath: status.podIP
  restartPolicy: Never
```

7. Service discovery?

Service

```
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "my-service"
    },
    "spec": {
        "selector": {
            "app": "MyApp"
        },
        "ports": [
            {
                "protocol": "TCP",
                "port": 80,
                "targetPort": 9376
            }
        ]
    }
}
```

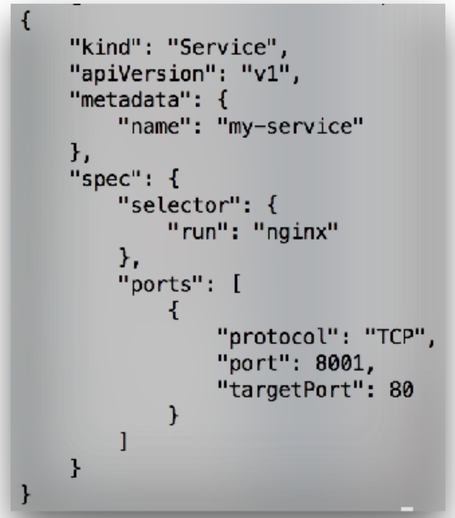
- The unified portal of replica Pods
- Portal IP:Port
- External load balancer
  - GCE
  - AWS
  - HAproxy
  - Nginx
  - OpenStack LB

Service Implementation

```
$ iptables-save | grep my-service
-A KUBE-SERVICES -d 10.0.0.116/32 -p tcp -m comment --comment "default/my-service: cluster IP" -m tcp --dport 8001 -j KUBE-SVC-KEAUNL7HVWWSEZA6

-A KUBE-SVC-KEAUNL7HVWWSEZA6 -m comment --comment "default/my-service:" --mode random -j KUBE-SEP-6XXFWO3KTRMPKCHZ
-A KUBE-SVC-KEAUNL7HVWWSEZA6 -m comment --comment "default/my-service:" --mode random -j KUBE-SEP-57KPRZ3JQVENLNBRZ

-A KUBE-SEP-6XXFWO3KTRMPKCHZ -p tcp -m comment --comment "default/my-service:" -m tcp -j DNAT --to-destination 172.17.0.2:80
-A KUBE-SEP-57KPRZ3JQVENLNBRZ -p tcp -m comment --comment "default/my-service:" -m tcp -j DNAT --to-destination 172.17.0.3:80
```



```
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "my-service"
    },
    "spec": {
        "selector": {
            "run": "nginx"
        },
        "ports": [
            {
                "protocol": "TCP",
                "port": 8001,
                "targetPort": 80
            }
        ]
    }
}
```

Tip: ipvs solution works in nat mode which is the same with this iptables way

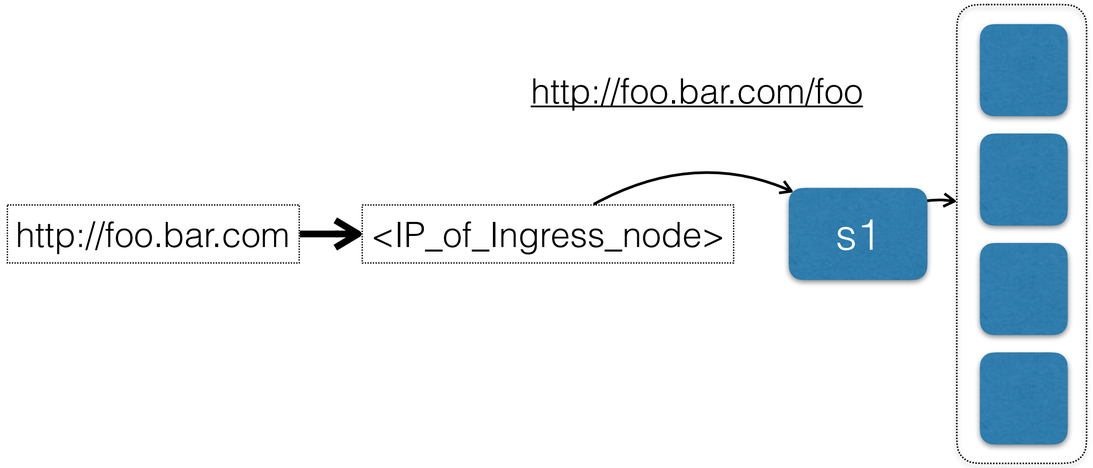
Publishing Services

- Use Service.Type=NodePort
  - <node_ip>:<node_port>
- External IP
  - IPs route to one or more cluster nodes (e.g. floating IP)
- Use external LoadBalancer
  - Require support from IaaS (GCE, AWS, OpenStack)
- Deploy a service-loadbalancer (e.g. HAproxy)
  - Official guide: https://github.com/kubernetes/contrib/tree/master/service-loadbalancer

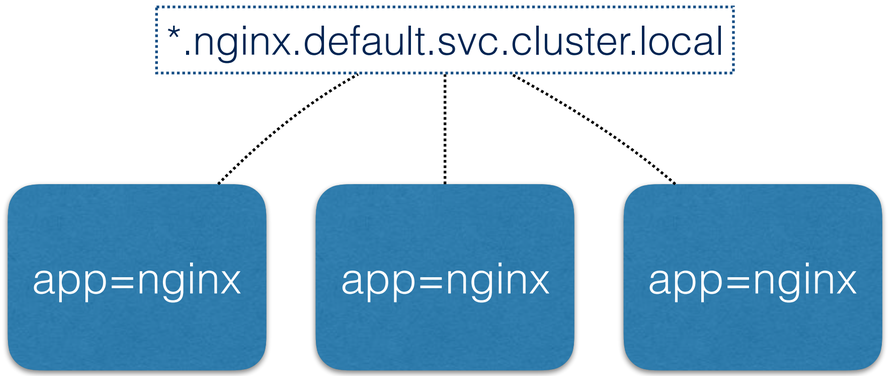
Ingress

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: s1
          servicePort: 80
```

- The next generation external Service load balancer
- Deployed as a Pod on dedicated Node (with external network)
- Implementation
  - Nginx, HAproxy, GCE L7
    - External access for service
    - SSL support for service
    - …



Headless Service



also: subdomain

```yaml
# A headless service to create DNS records
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  # *.nginx.default.svc.cluster.local
  clusterIP: None
  selector:
    app: nginx
```

8. Stateful applications?

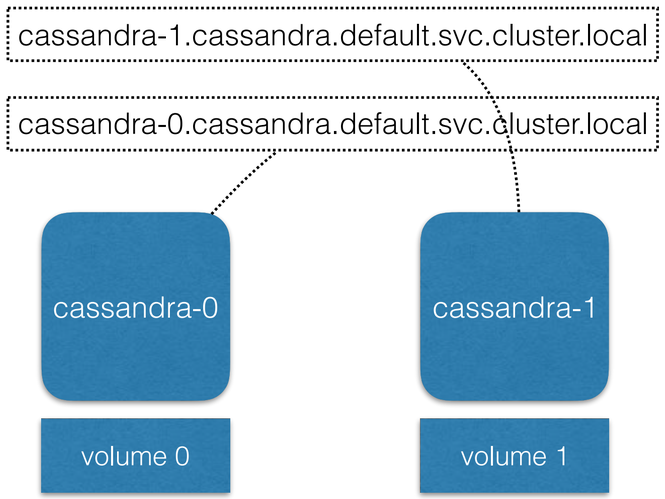
StatefulSet: “clustered applications”

- **Ordinal index**
  - startup/teardown ordering
- **Stable hostname**
- **Stable storage**
  - linked to the ordinal & hostname
- Databases like MySQL or PostgreSQL
  - single instance attached to a persistent volume at any time
- Clustered software like Zookeeper, Etcd, or Elasticsearch, Cassandra
  - stable membership.

Update StatefulSet:
Scale: create/delete one by one
Scale in: will not delete old persistent volume

StatefulSet Example

```
# Headless service to provide DNS lookup
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cassandra
  name: cassandra
spec:
  clusterIP: None
  ports:
    - port: 9042
  selector:
    app: cassandra-data
```




```
apiVersion: "apps/v1alpha1"
kind: StatefulSet
metadata:
  name: cassandra
spec:
  serviceName: cassandra
  # replicas are the same as used by Replication Controllers
  # except pets are deployed in order 0, 1, 2, 3, etc
  replicas: 5
```

$ kubectl patch petset cassandra -p '{"spec":{"replicas":10}}'

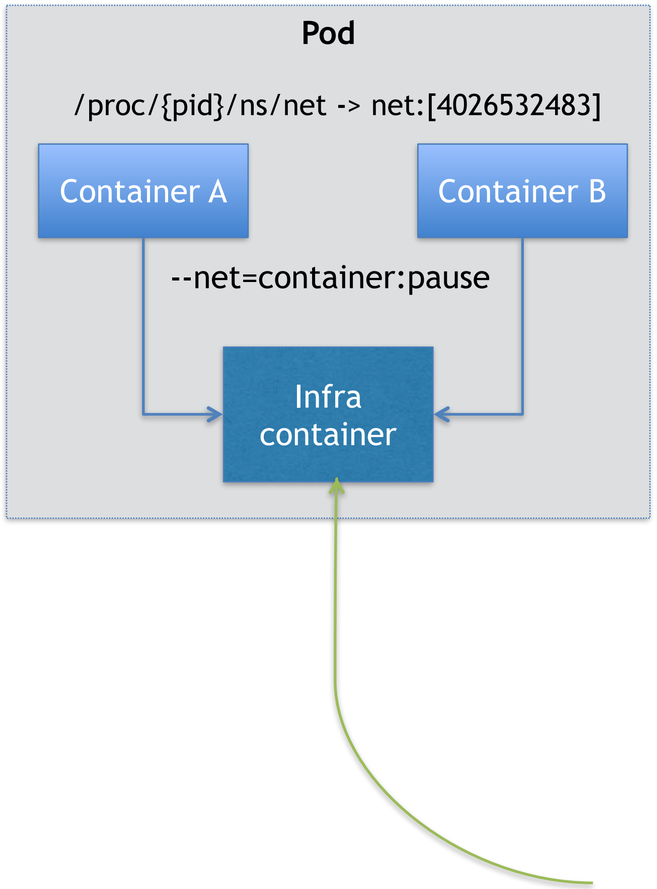
```
        env:
          - name: MAX_HEAP_SIZE
            value: 8192M
          - name: HEAP_NEWSIZE
            value: 2048M
          # this is relying on guaranteed network identity of Pet Sets, we
          # will know the name of the Pets / Pod before they are created
          - name: CASSANDRA_SEEDS
            value: "cassandra-
0.cassandra.default.svc.cluster.local,cassandra-
1.cassandra.default.svc.cluster.local"
          - name: CASSANDRA_CLUSTER_NAME
            value: "OneKDemo"
          - name: CASSANDRA_DC
            value: "DC1-Data"
```

```
        # These volume mounts are persistent. They are like inline claims,
        # but not exactly because the names need to match exactly one of
        # the pet volumes.
        volumeMounts:
        - name: cassandra-data
          mountPath: /cassandra_data
```

9. Container network?

One Pod One IP

- Network sharing is important for affiliate containers
- Not all containers need independent network
- Network implementation for pod is totally the same as for single container



Kubernetes uses CNI

- **CNI plugin**
  - e.g. Calico, Flannel etc
  - The kubelet cni flags:
    - --network-plugin=cni
    - --network-plugin-dir=/etc/cni/net.d

- CNI is very simple

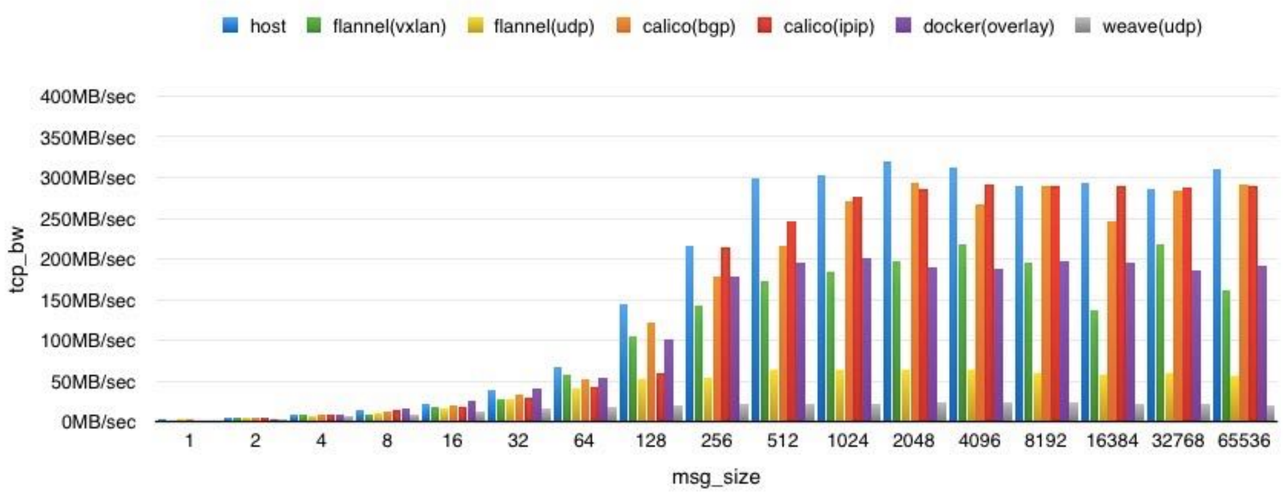
1. Kubelet creates a network namespace for Pod

2. Kubelet invokes CNI plugin to configure the NS (interface name, IP, MAC, gateway, bridge name …)

3. Infra container in Pod join this network namespace

Tips

	Calico	Flannel	Weave	Docker Overlay Network
Network Model	Pure Layer-3 Solution	VxLAN or UDP Channel	VxLAN or UDP Channel	VxLAN



- host < calico(bgp) < calico(ipip) = flannel(vxlan) = docker(vxlan) < flannel(udp) < weave(udp)

- Test graph comes from: http://cmgs.me/life/docker-network-cloud

Calico

- Step 1: Run calico-node image as DaemonSet

```yaml
kind: DaemonSet
apiVersion: extensions/v1beta1
metadata:
  name: calico-node
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    k8s-app: calico-node
spec:
  selector:
    matchLabels:
      kubernetes.io/cluster-service: "true"
      k8s-app: calico-node
  template:
    metadata:
      labels:
        kubernetes.io/cluster-service: "true"
        k8s-app: calico-node
    spec:
      hostNetwork: true
      containers:
        - name: calico-node
          image: quay.io/calico/node:v0.21.0
          env:
            - name: ETCD_ENDPOINTS
              value: "http://10.0.0.17:6666"
            - name: CALICO_NETWORKING
              value: "false"
          securityContext:
            privileged: true
```

Calico

- Step 2: Download and enable calico cni plugin

```
wget -N -P /opt/cni/bin https://github.com/projectcalico/calico-cni/releases/download/v1.3.1/calico
wget -N -P /opt/cni/bin https://github.com/projectcalico/calico-cni/releases/download/v1.3.1/calico-ipam
chmod +x /opt/cni/bin/calico /opt/cni/bin/calico-ipam
```

```
mkdir -p /etc/cni/net.d
$ cat >/etc/cni/net.d/10-calico.conf <<EOF
{
    "name": "calico-k8s-network",
    "type": "calico",
    "etcd_authority": "<ETCD_IP>:<ETCD_PORT>",
    "log_level": "info",
    "ipam": {
        "type": "calico-ipam"
    },
    "policy": {
        "type": "k8s"
    }
}
EOF
```

Calico

- Step 3: Add calico network controller

```
$ kubectl get pods --namespace=calico-system
NAME                                       READY     STATUS    RESTARTS   AGE
calico-policy-controller-172.18.18.101   2/2       Running   0          1m
```

- **Done!**

10. Persistent volume?

Persistent Volumes

- -v *host_path*:*container_path*

1. **Attach** networked storage to host path
    1. mounted to *host_path*
2. **Mount** host path as container volume
    1. bind mount *container_path* with *host_path*
3. Independent volume control loop

Officially Supported PVs

- GCEPersistentDisk
- AWSElasticBlockStore
- AzureFile
- FC (Fibre Channel)
- NFS
- iSCSI
- RBD (Ceph Block Device)
- CephFS
- Cinder (OpenStack block storage)
- Glusterfs
- VsphereVolume
- HostPath (single node testing only)
- **more than 20+**

- Write your own volume plugin: **FlexVolume**
  1. Implement 10 methods
  2. Put binary/shell in plugin directory
- example: LVM as k8s volume

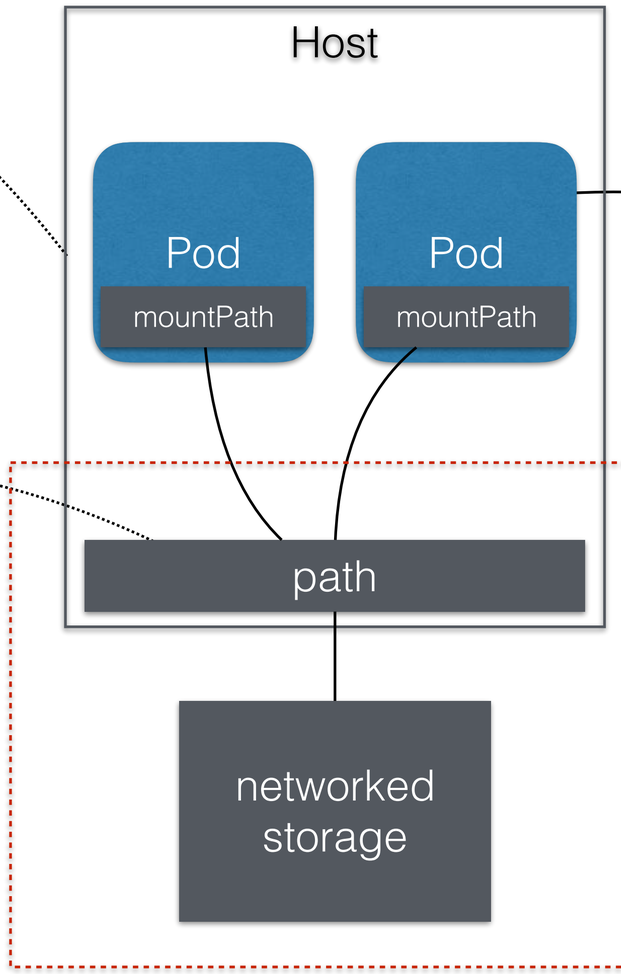
Production ENV Volume Model

PersistentVolumeClaims

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim-1
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Persistent Volumes

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv0001
  labels:
    type: local
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  nfs:
    path: /tmp
    server: 172.17.0.2
```



Pod

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: mypod
  labels:
    name: frontendhttp
spec:
  containers:
    - name: myfrontend
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
      - mountPath: "/usr/share/nginx/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
       claimName: myclaim-1
```

Key point: 职责分离

PV & PVC

- **System Admin:**
  - **$ kubectl create -f nfs-pv.yaml**
    - create a volume with **access mode, capacity, recycling mode**
- **Dev**:
  - **$ kubectl create -f pv-claim.yaml**
    - **request** a volume with **access mode, resource, selector**
  - **$ kubectl create -f pod.yaml**

More …

- **GC**
- **Health check**
- **Container lifecycle hook**
- **Jobs** (batch)
- Pod affinity and binding
- Dynamic provisioning
- Rescheduling
- **CronJob**
- **Logging** and monitoring
- **Network policy**
- Federation
- Container capabilities
- Resource quotas
- Security context
- Security polices
- **GPU scheduling**

Summary

- Q: Where are all these control panel ideas come from?
- A: Kubernetes = “Borg” + “Container”
- Kubernetes is a set of methodology for using containers based on past 10+ yr’s exp in Google Inc.
  - “不要摸着石头过河”
- Kubernetes is a container centric DevOps/Workload orchestration system
  - Not a “CI/CD”, “Micro-service” focused container cloud

Growing Adopters

- Public Cloud
  - AWS
  - Microsoft Azure (acquired Deis)
  - Google Cloud
  - 腾讯云
  - 百度AI
  - 阿里云

Enterprise Users

Data source: Kubernetes Leadership Summit (with CN adopters)

THE END

@resouer
harryzhang@zju.edu.cn