Luxeria Talks

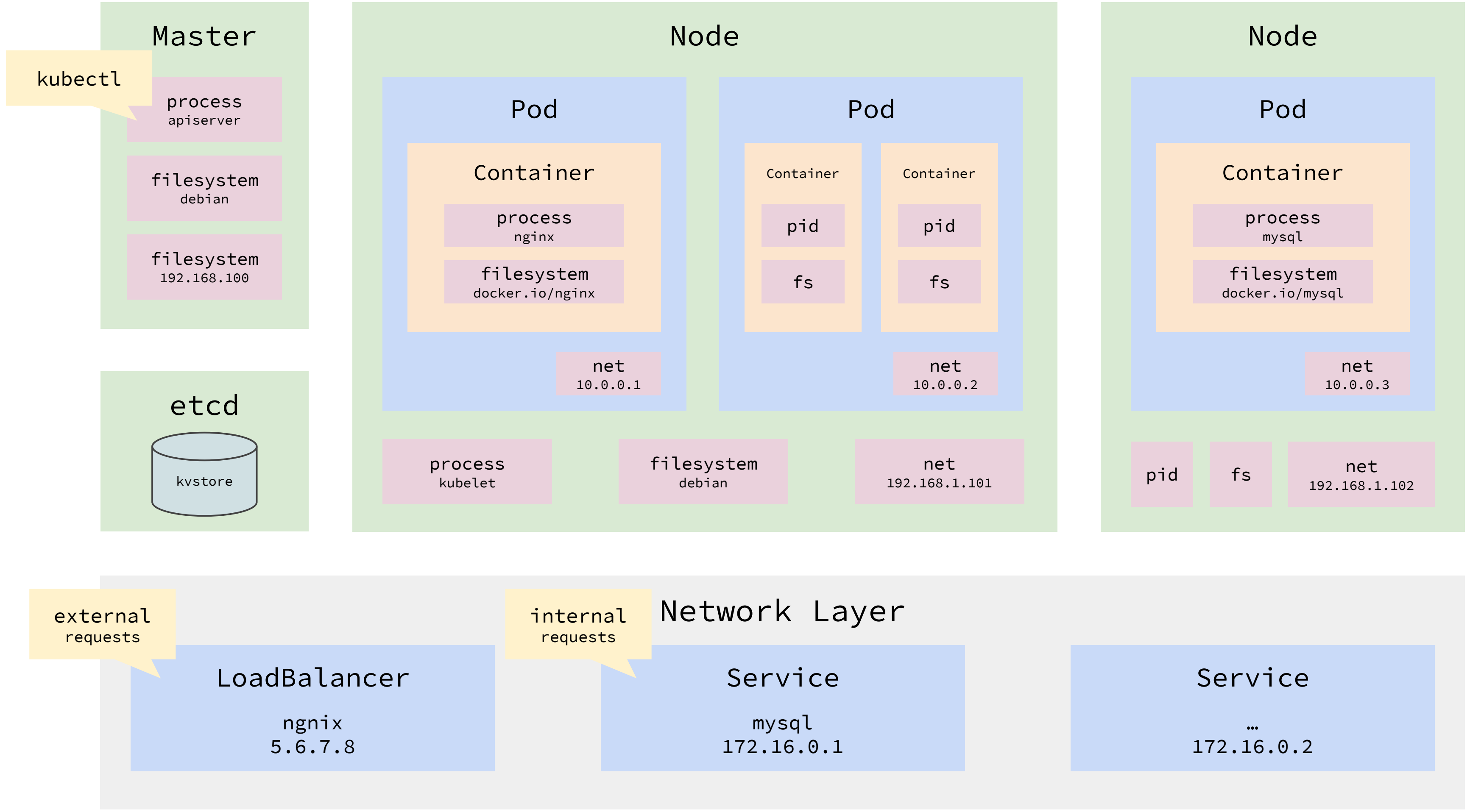
Kubernetes Security

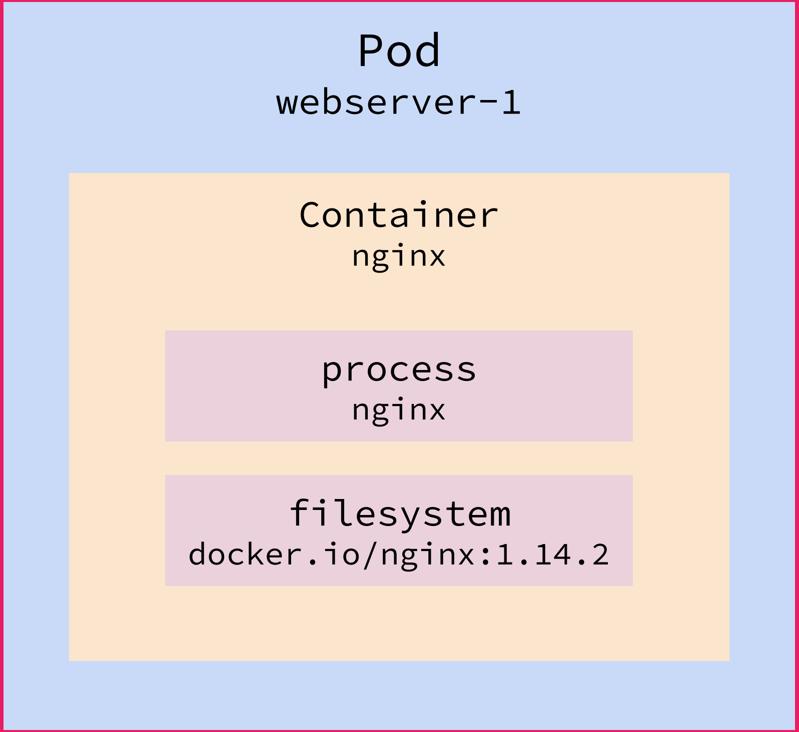
Sebastian Wicki

1. April 2020

Kubernetes 1×1

drinking from the firehose


Master
kubectl
process apiserver
filesystem debian
filesystem 192.168.100
etcd
kvstore
Node
Pod
Container
process nginx
filesystem docker.io/nginx
net 10.0.0.1
Pod
Container
pid
fs
Container
pid
fs
net 10.0.0.2
process kubelet
filesystem debian
net 192.168.1.101
Node
Pod
Container
process mysql
filesystem docker.io/mysql
net 10.0.0.3
pid
fs
net 192.168.1.102
Network Layer
external requests
LoadBalancer
ngnix
5.6.7.8
internal requests
Service
mysql
172.16.0.1
Service
…
172.16.0.2



```
kubectl create -f
webserver-1.yaml
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webserver-1
  labels:
   environment: production
   app: nginx
   tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Labels and Selectors

```
kubectl get pods
webserver-1
```

```
NAME          READY   STATUS    …   IP         NODE
webserver-1   1/1     Running   …   10.0.0.1   worker0
```

```
apiVersion: v1
kind: Pod
metadata:
  name: webserver-1
  labels:
   environment: production
   app: nginx
   tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Labels and Selectors

```
kubectl get pods
-l 'environment=production,
app=frontend'
```

```
NAME          READY   STATUS    …   IP         NODE
webserver-1   1/1     Running   …   10.0.0.1   worker0
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webserver-1
  labels:
   environment: production
   app: nginx
   tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Labels and Selectors

```
kubectl get pods
-l 'environment in (production, qa)'
```

```
NAME          READY   STATUS    …   IP         NODE
webserver-1   1/1     Running   …   10.0.0.1   worker0
```

```
apiVersion: v1
kind: Pod
metadata:
  name: webserver-1
  labels:
   environment: production
   app: nginx
   tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Labels and Selectors

```yaml
apiVersion: v1
kind: Service
metadata:
  name: www-prod
spec:
  selector:
    app: nginx
    environment: production
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webserver-1
  labels:
   environment: production
   app: nginx
   tier: frontend
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Built-in Security

Role-Based Access Control

https://kubernetes.io/docs/reference/access-authn-authz/rbac/

Role-Based Access Control

- Manage access control to API objects
  - subjects: User, ServiceAccount
  - objects: Logs, NetworkPolicies, Pods
- Examples:
  - User “alice” may have only read-access to logs
  - Pod “drone-ci” may only deploy pods in “development” namespace

Roles & Role Bindings

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: allow-read-pod-logs
rules:
- apiGroups: [""] # built-in types
  resources: ["pods", "pods/log"]
  verbs: ["get", "list"]
```

```
kubectl logs webserver-1 ✅
kubectl delete pod webserver-1 □
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: alice-may-read-logs
  namespace: default
subjects:
- kind: User
  name: alice
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: allow-read-pod-logs
  apiGroup: rbac.authorization.k8s.io
```

Service Accounts

- Service accounts are user accounts for machines
- Each pod has a default service account that can be overwritten
- Kubernetes mounts auto-generated credentials in /var/run/secrets/
- kubectl from pod works automagically

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: i-am-robot
---

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: i-am-robot
  automountServiceAccountToken: true
  ...
```

Secrets

https://kubernetes.io/docs/concepts/configuration/secret/

Secrets

- Application secrets and credentials can also be managed by K8s
- Accessible from Pod via
  - Environment Variables
  - Filesystem Mounts
  - Kubernetes API
- Used to pull images from protected container registries

```
kubectl create secret generic my-ssh-keys
  --from-file=id_rsa=~/.ssh/id_rsa
  --from-file=id_rsa.pub=path~/.ssh/id_rsa.pub

kubectl create secret generic my-db-creds
  --from-literal=user=bob
  --from-literal=pass=guppy
  --from-literal=dbname=bobnet

kubectl create secret tls my-tls-secret
  --cert=path/to/tls.cert
  --key=path/to/tls.key

kubectl create secret docker-registry my-docker
  --from-file=~/.docker/config.json
```

Secrets are <u>not</u> encrypted by default

- Secrets can be protected via RBAC
- Stored in plain-text on filesystem of etcd/api server
  - Encryption is opt-in:
    - kube-apiserver --encryption-provider-config
    - https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/
- Third-Party Solutions
  - HashiCorp Vault
  - Commercial cloud provider solutions

Pod Security Policies

https://kubernetes.io/docs/concepts/policy/pod-security-policy/

Pod Security Policy

- Defines which host resources pods are allowed to access
- Must be assigned to a User or ServiceAccount using a RoleBinding or ClusterRoleBinding

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted-pods
spec:
  privileged: false
  volumes:
     - 'emptyDir'
     - 'configMap'
     - 'secret'
     # - 'hostPath'
  hostNetwork: false
  runAsUser:
     rule: 'MustRunAsNonRoot'
```

Honorable Mentions

- Limit Ranges
- Resource Quotas
- Metrics Server
- Audit Logger

Third-Party Solutions

Securing Application Behavior

Securing Application Behavior

Static Analysis

- Checks container images before deployment for known vulnerabilities
- **Quay Clair**
  - runs in CI/CD stage
  - available also for quay.io

Dynamic Analysis

- Checks container at run-time for suspicious behavior
- **docker-seccomp**
  - filter system calls
  - enabled by default, tunable
- **Sysdig Falco**
  - rule-based monitoring
  - e.g. emit a warning when shell is started inside container

Network Security Policies

https://kubernetes.io/docs/concepts/services-networking/network-policies/
https://docs.cilium.io/en/v1.7/policy/language/

Network Policies

NetworkPolicy

- Standardized format
- Enforced by the CNI
  - Calico
  - **Cilium**
  - Kube-router
  - Romana
  - Weave Net

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: frontend-can-access-db
spec:
  podSelector:
     matchLabels:
       role: db
  ingress:
  - from:
     - podSelector:
         matchLabels:
           role: frontend
```

Cilium Network Policy

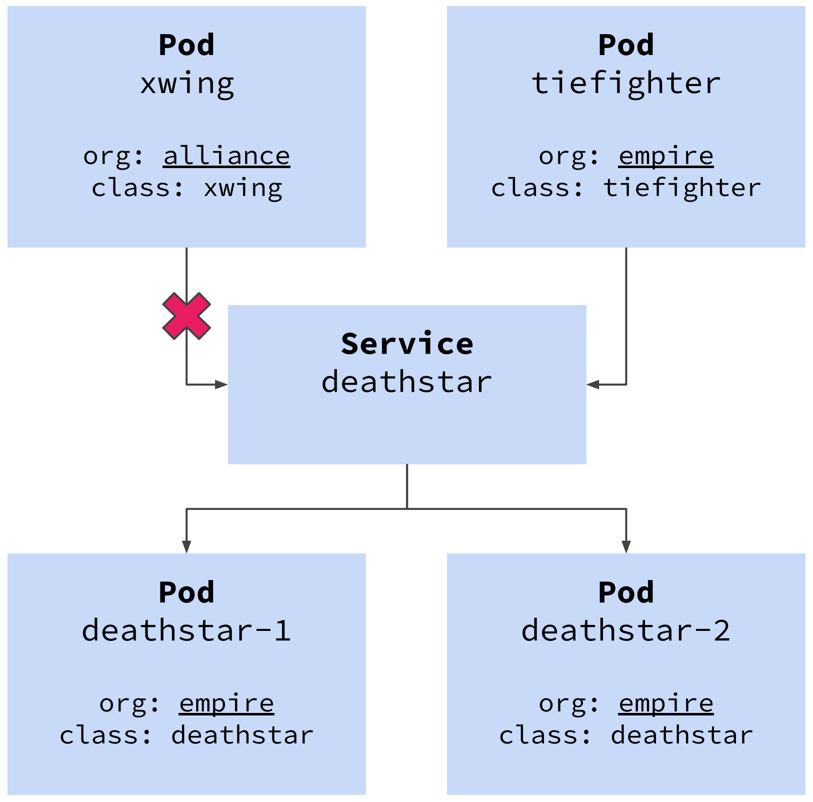
Cilium Policy Enforcement Modes

- **default**
  - if any ingress rule applies to the pod, it goes into default deny at ingress
  - if any egress rule applies to the pod, it goes into default deny at egress
- **always** (default deny)
- **never** (allow everything)

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: policy-name
spec:
  endpointSelector:
    matchLabels:
      app: nginx
  ingress:
    - ...
  egress:
    - ...
```

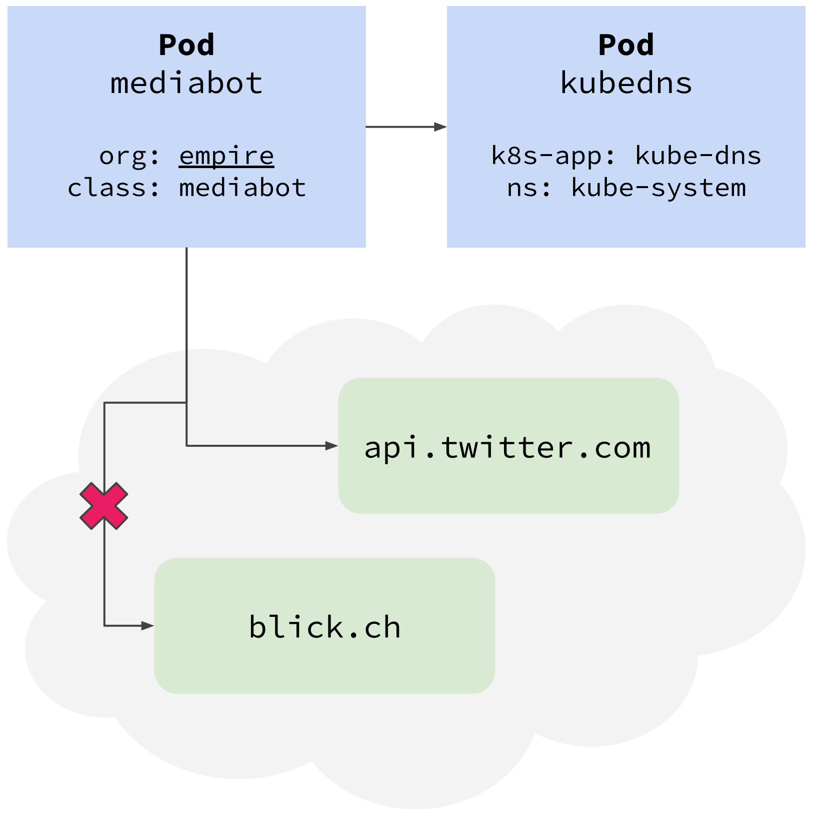
Demo

Ingress Policy



Demo

Egress Policy



Fin