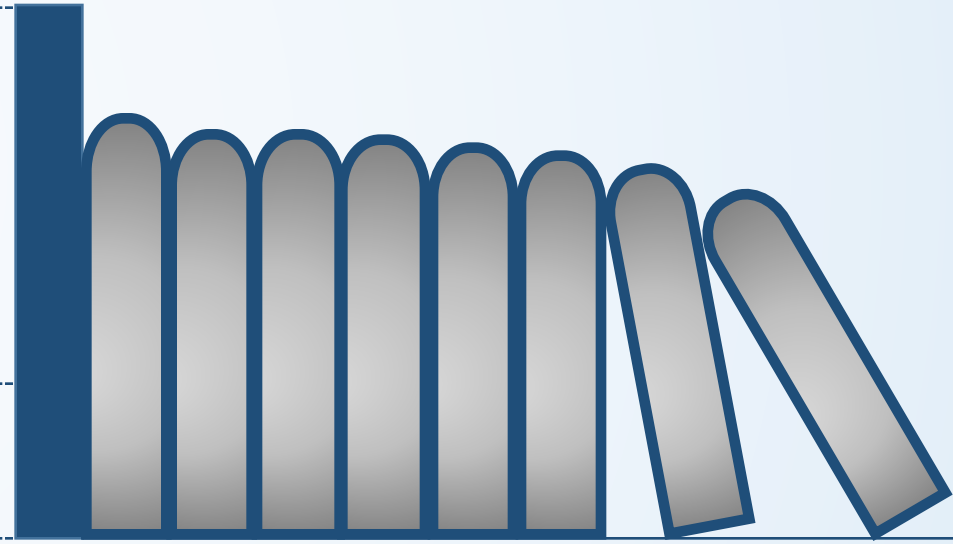
Kubernetes安全

主讲人：宋小金

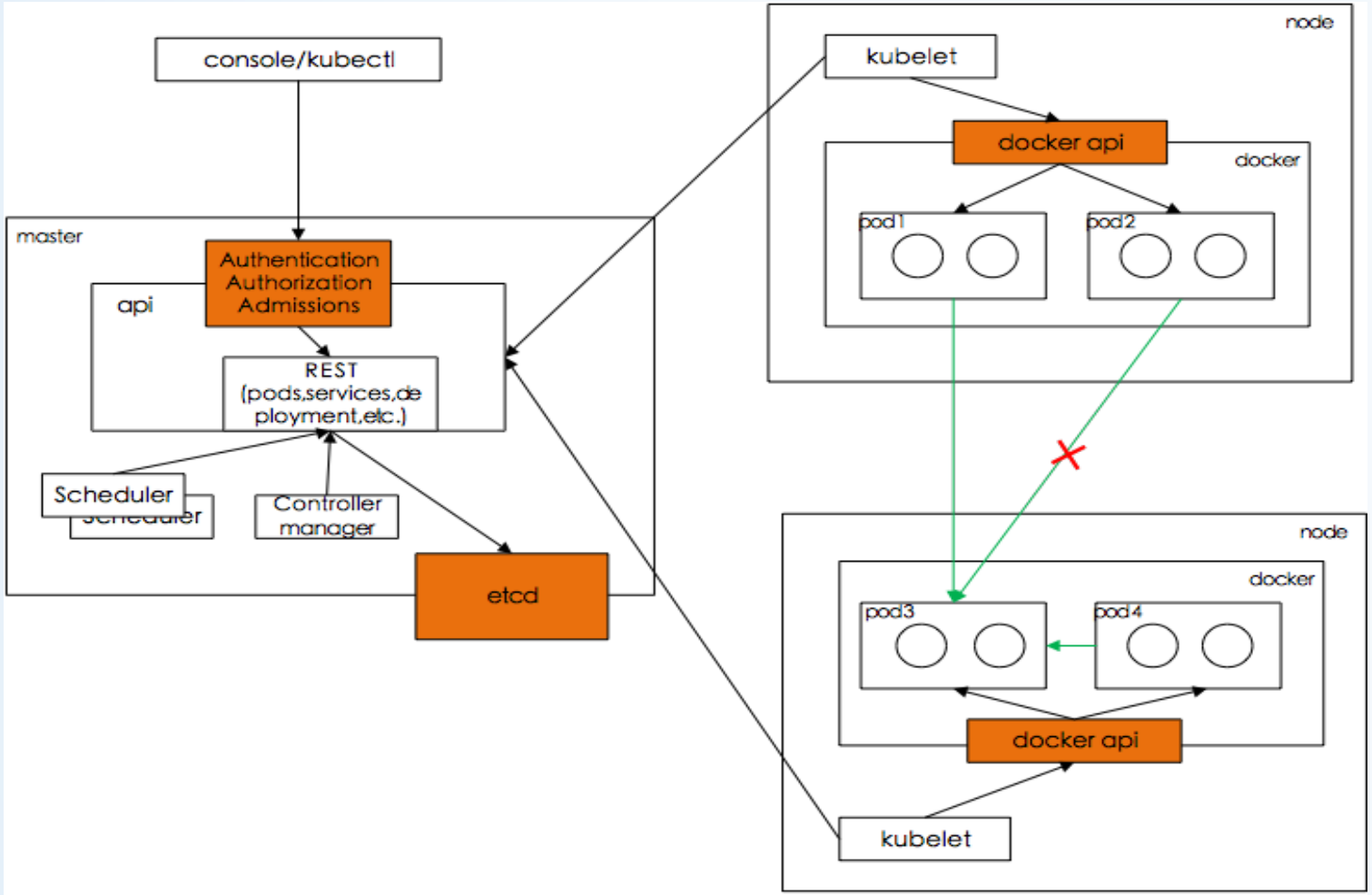


目录

预期收获

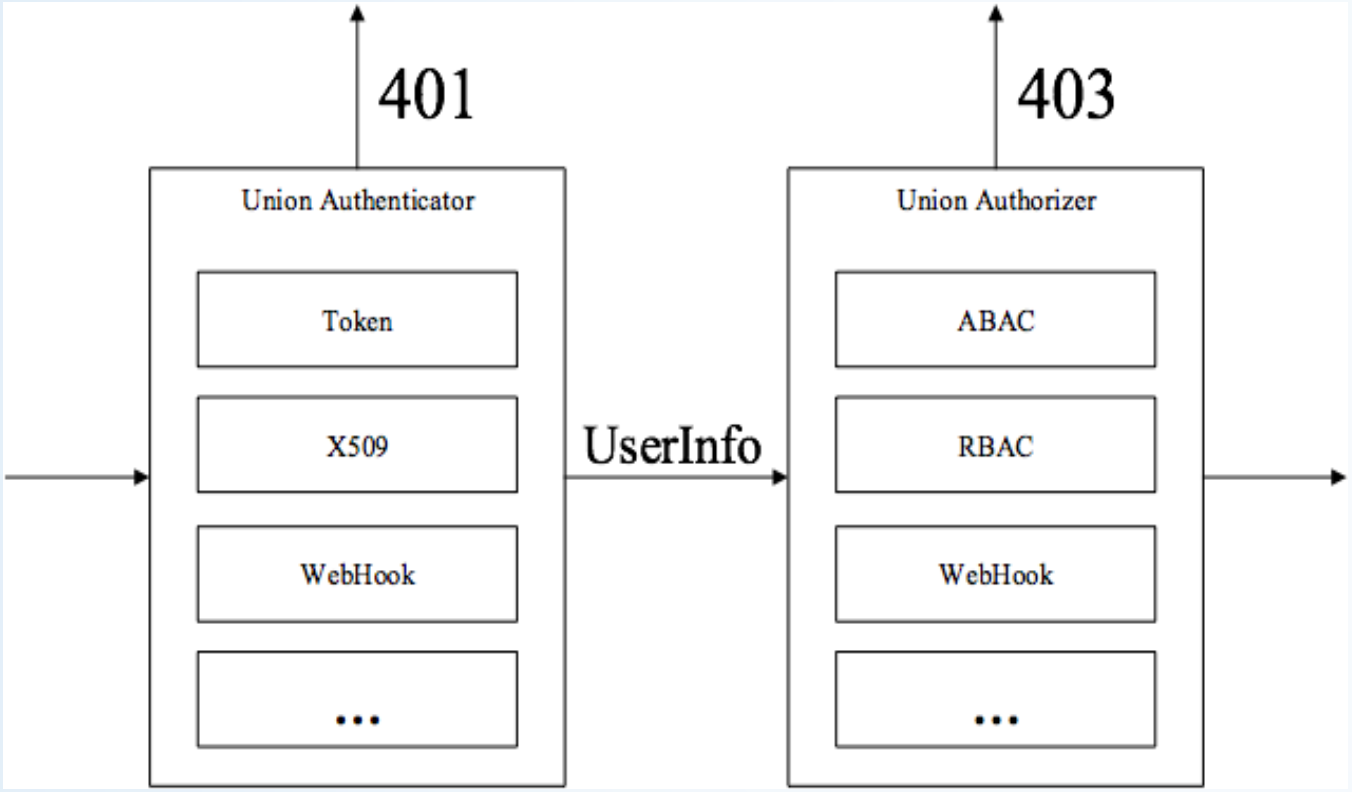
- *了解Kubernetes安全规则*
- 了解如何使用安全规则

安全概览



- 部署态的安全控制
  - 认证
  - 鉴权
  - Admission（准入控制）
  - Pod SecurityContext
- 运行态的安全控制
  - Network policy

认证(Authentication)和鉴权(Authorization)




```
// Info describes a user that has been authenticated to the system.
type Info interface {
  // GetName returns the name that uniquely identifies this user among
all
  // other active users.
  GetName() string
  // GetUID returns a unique value for a particular user that will
change
  // if the user is removed from the system and another user is added
with
  // the same name.
  GetUID() string
  // GetGroups returns the names of the groups the user is a member of
  GetGroups() []string
  // GetExtra can contain any additional information that the
authenticator
  // thought was interesting. One example would be scopes on a token.
  // Keys in this map should be namespaced to the authenticator or
  // authenticator/authorizer pair making use of them.
  // For instance: "example.org/foo" instead of "foo"
  // This is a map[string][]string because it needs to be serializeable
into
  // a SubjectAccessReviewSpec.authorization.k8s.io for proper
authorization
  // delegation flows
  // In order to faithfully round-trip through an impersonation flow,
these keys
  // MUST be lowercase.
  GetExtra() map[string][]string
}
```

- 认证支持多种方式，其中一种认证方式认证通过即通过，输出userinfo
- 基于认证输出的userinfo进行鉴权，鉴权也支持多种方式，常用方式为RBAC

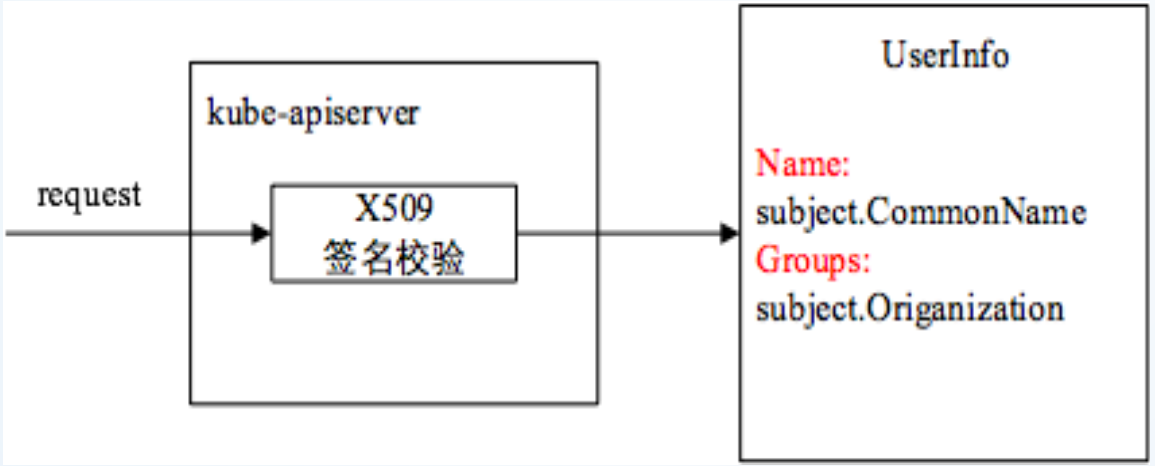
认证(Authentication)

认证方式有：X509、service account、Authenticating Proxy、WebHook、username/password…

常用认证方式介绍：

X509：

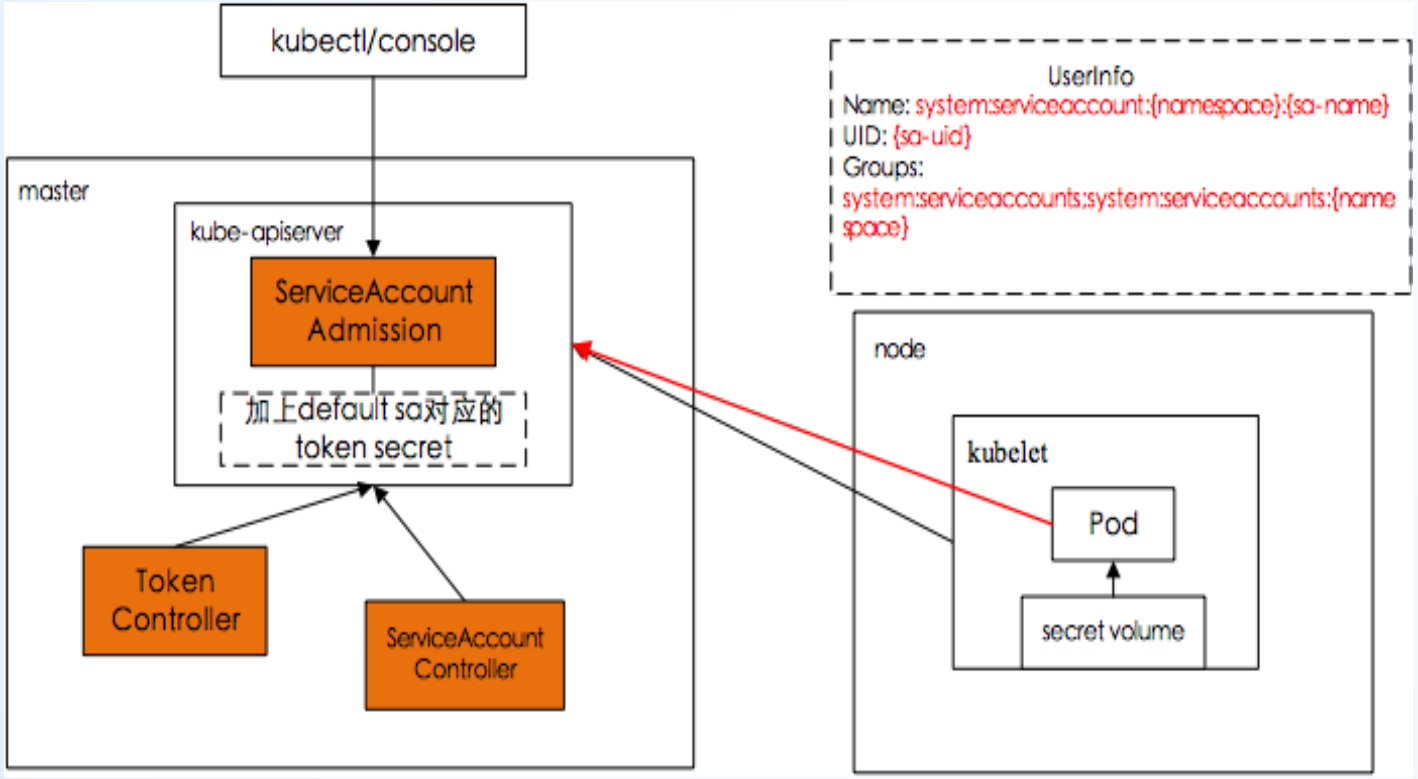
- Kube-apiserver的启动参数’—client-ca-file=ca.crt’指定X509根证书，请求中需带有由该根证书签名的证书，才能认证通过
- 客户端签署的证书里包含user、group信息，具体为证书的subject.CommonName（user name）以及subject.Organization（group）



认证(Authentication)

Service Account（为k8s必选认证方式）：

- Kube-apiserver的启动参数’—service-account-key-file=key.pem’指定pem文件，用以生成bearer token；’—service-account-lookup=true/false’表示在删除service account后其token是否被吊销
- Serviceaccount Admission默认给Pod打上service account，当然用户也可以自行指定所需要的service account

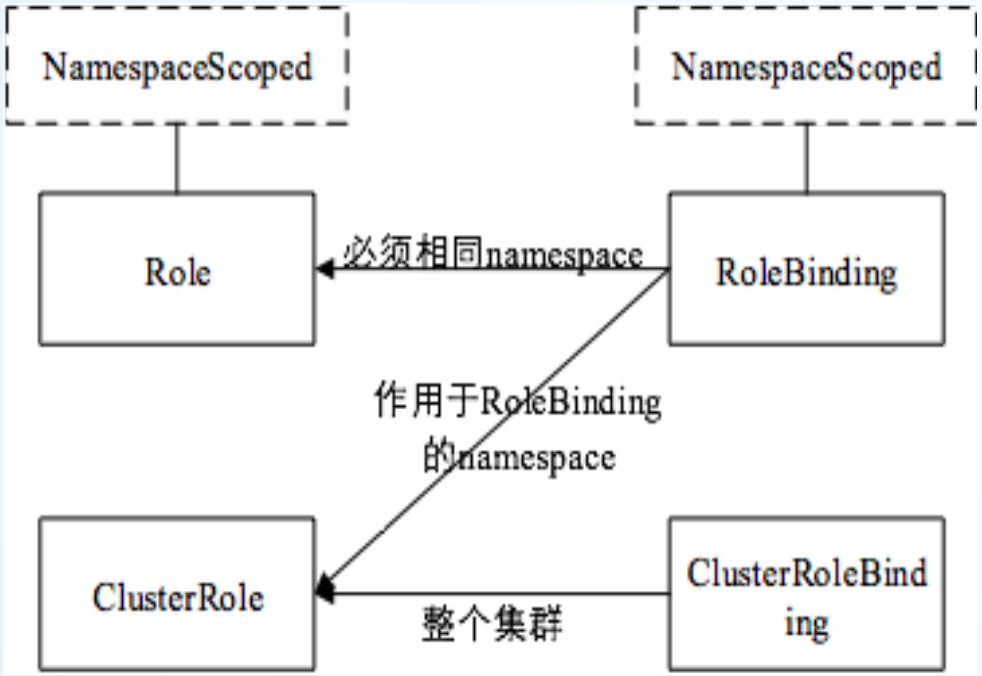



```
  spec:
    serviceAccountName: default
    containers:
    - image: nginx:latest
      imagePullPolicy: IfNotPresent
      name: container-0
      volumeMounts:
      - mountPath:
/var/run/secrets/kubernetes.io/serviceaccount
        name: default-token-rm7xw
        readOnly: true
    volumes:
    - name: default-token-rm7xw
      secret:
        defaultMode: 420
        secretName: default-token-rm7xw
```

鉴权(Authorization)

鉴权分为以下几种：RBAC、ABAC、Node以及Webhook

常用RBAC介绍：




```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: secret-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["secrets"]
```

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: read-pods
subjects:
- kind: User
  name: wangbo
  apiGroup: rbac.authorization.k8s.io/v1
roleRef:
  kind: Role #this can be Role or ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io/v1

---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager
  apiGroup: rbac.authorization.k8s.io/v1
roleRef:
  kind: ClusterRole #this must be ClusterRole
```

Admission(PodSecurityPolicy)

- Kube-apiserver的启动参数’—admission-control=PodSecurityPolicy’新增PodSecurityPolicy admission
- Admin用户创建PodSecurityPolicy策略，决定能创建什么样的Pod
- 创建Pod的用户也必须赋予它能使用PodSecurityPolicy策略的权限

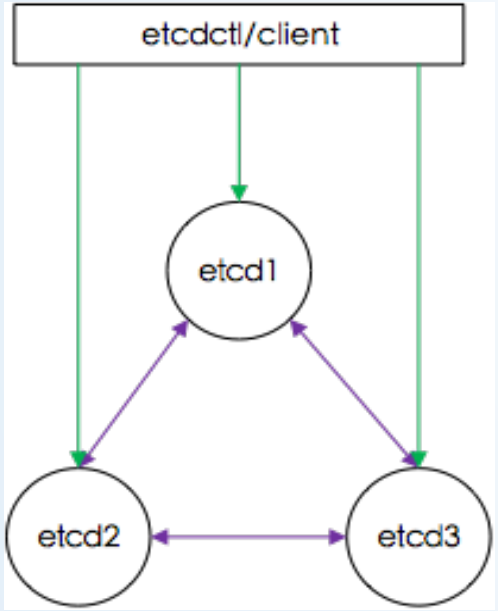
```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName:  'docker/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName:  'runtime/default'
spec:
  privileged: false
  # Required to prevent escalations to root.
  allowPrivilegeEscalation: false
  # This is redundant with non-root + disallow privilege escalation,
  # but we can provide it for defense in depth.
  requiredDropCapabilities:
    - ALL
  # Allow core volume types.
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that persistentVolumes set up by the cluster admin are safe to use.
    - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    # Require the container to run without root privileges.
    rule: 'MustRunAsNonRoot'
  seLinux:
    # This policy assumes the nodes are using AppArmor rather than SELinux.
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max: 65535
  fsGroup:
    rule: 'MustRunAs'
    ranges:
      # Forbid adding the root group.
      - min: 1
        max: 65535
  readOnlyRootFilesystem: fals
```

安全的持久化保存键值（etcd）

- etcd支持备份恢复机制，防止数据被误删导致数据丢失
- 用户的敏感信息建议存放在secret类型的资源中，该类型资源是加密存储在etcd中

- etcd支持https，kube-apiserver访问etcd使用https协议

具体配置方式：



Client->Server:
--cert-file=<path>
--key-file=<path>
通道以tls协议加密

--client-cert-auth
--trusted-ca-file=<path>
服务端会认证客户端证书是否是受信任CA签发

--auto-tls
是否系统自动生成证书

Server->Server:
--peer-cert-file=<path>
--peer-key-file=<path>
通道以tls协议加密

--peer-client-cert-auth
--peer-trusted-ca-file=<path>
服务端会认证客户端证书是否是受信任CA签发

--peer-auto-tls
是否系统自动生成证书

安全上下文（Pod SecurityContext）

- 分为Pod级别和容器级别，容器级别的会覆盖Pod级别的相同设置。
- 在有PodSecurityPolicy策略的情况下，两者需要配合使用

是否使用特权容器

指定容器启动UID

指定Pod中容器文件所属组GID

容器的文件系统是否是只读

容器系统调用能力配置

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: wangbo
spec:
  securityContext:
    privileged: false
    runAsUser: 1000
    fsGroup: 2000
  volumes:
  - name: test
    emptyDir: {}
  containers:
  - name: test
    image: gcr.io/google-samples/node-hello:1.0
    volumeMounts:
    - name: test
      mountPath: /data/test
    securityContext:
      readOnlyRootFilesystem: false
      runAsUser: 1001
      privileged: false
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
        drop: ["SYS_BOOT"]
```

Network Policy

分为Ingress和Egress策略控制，都为白名单

- Ingress为入口请求控制
- Egress为出口请求控制

规则匹配器，选择匹配的Pod

远端（访问端）IP白名单开放

远端（访问端）namespace白名单开放

远端（访问端）pod白名单开放

本端（被访问端）允许被访问的端口和协议

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

Network Policy

禁止所有入口请求

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

禁止所有出口请求

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

允许所有入口请求

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress:
  - {}
```

允许所有出口请求

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress :
  - {}
```

课程回顾

已学知识要点

了解Kubernetes安全规则以及如何使用