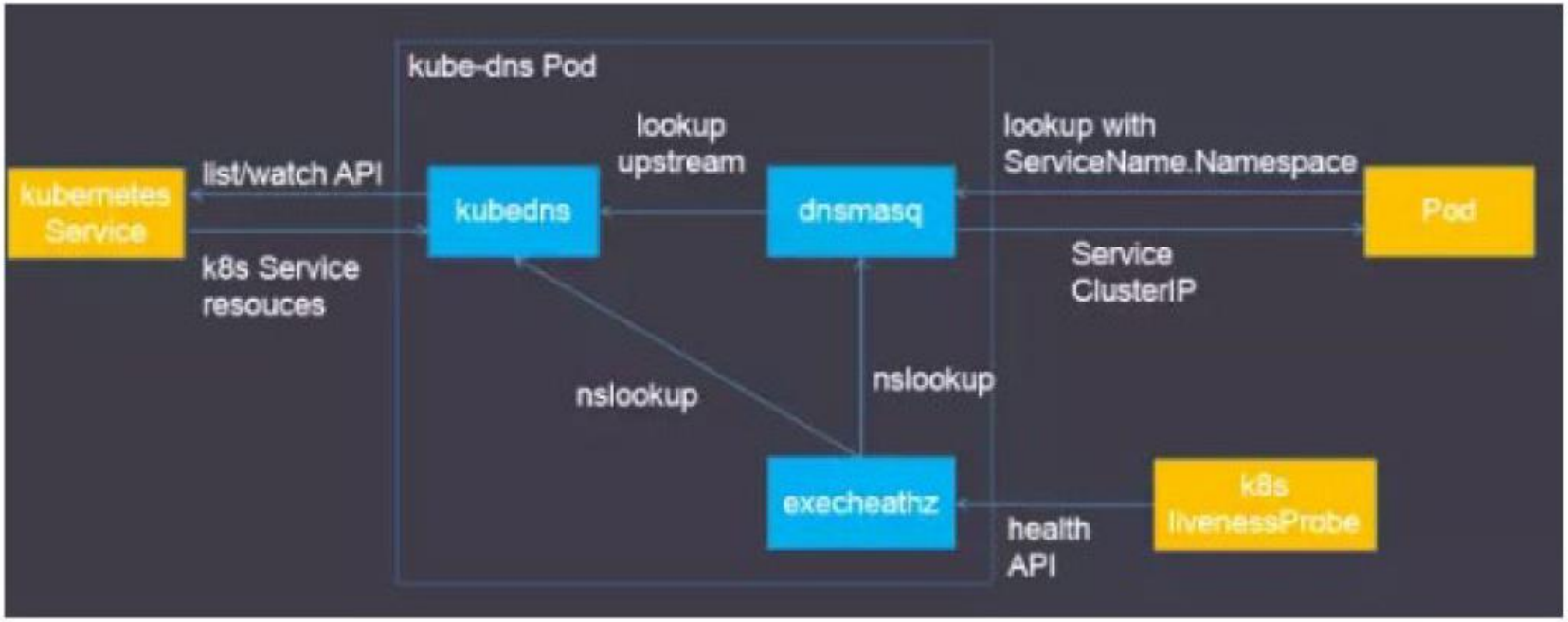
实现原理


kube-dns Pod
lookup
upstream
lookup with
ServiceName.Namespace
kubernetes
Service
list/watch API
kubedns
dnsmasq
Pod
k8s Service
resouces
Service
ClusterIP
nslookup
nslookup
execheathz
k8s
livenessProbe
health
API

- kubedns容器的功能：
    - 接入SkyDNS，为dnsmasq提供查询服务
    - 替换etcd容器，使用树形结构在内存中保存DNS记录
    - 通过K8S API监视Service资源变化并更新DNS记录
    - 服务10053端口
    - 会检查两个容器的健康状态。
- dnsmasq容器的功能：
    - Dnsmasq是一款小巧的DNS配置工具
    - 在kube-dns插件中的作用
    - 通过kubedns容器获取DNS规则，在集群中提供DNS查询服务
    - 提供DNS缓存，提高查询性能
    - 降低kubedns容器的压力、提高稳定性
    - 在kube-dns插件的编排文件中可以看到，dnsmasq通过参数–server=127.0.0.1#10053指定upstream为kubedns。
- exec-healthz容器的功能：
    - 在kube-dns插件中提供健康检查功能
    - 会对两个容器都进行健康检查，更加完善。

备注：kube-dns组件 github 下载地址

创建kubedns-cm.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
```

对比 kubedns-dns-controller 配置文件修改

```
diff kubedns-controller.yaml.sed /root/kubernetes/k8s-deploy/mainifest/dns/kubedns-controller.yaml
58c58
<         image: gcr.io/google_containers/k8s-dns-kube-dns-amd64:1.14.5
---
>         image: gcr.io/google_containers/k8s-dns-kube-dns-amd64:1.14.4
88c88
<         - --domain=$DNS_DOMAIN.
---
>         - --domain=cluster.local.
109c109
<         image: gcr.io/google_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.5
---
>         image: gcr.io/google_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.4
128c128
<         - --server=/$DNS_DOMAIN/127.0.0.1#10053
---
>         - --server=/cluster.local/127.0.0.1#10053
147c147
<         image: gcr.io/google_containers/k8s-dns-sidecar-amd64:1.14.5
---
>         image: gcr.io/google_containers/k8s-dns-sidecar-amd64:1.14.4
160,161c160,161
<         - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.$DNS_DOMAIN,5,A
<         - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.$DNS_DOMAIN,5,A
---
>         - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local,5,A
>         - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local,5,A
```

创建kubedns-dns-controller

```
# Should keep target in cluster/addons/dns-horizontal-autoscaler/dns-horizontal-autoscaler.yaml
# in sync with this file.

# Warning: This is a file generated from the base underscore template file: kubedns-controller.yaml.base

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  # replicas: not specified here:
  # 1. In order to make Addon Manager do not reconcile this replicas parameter.
  # 2. Default is 1.
  # 3. Will be tuned in real time if DNS horizontal auto-scaling is turned on.
  strategy:
    rollingUpdate:
      maxSurge: 10%
      maxUnavailable: 0
```

```
  selector:
    matchLabels:
      k8s-app: kube-dns
  template:
    metadata:
      labels:
        k8s-app: kube-dns
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      volumes:
      - name: kube-dns-config
        configMap:
          name: kube-dns
          optional: true
      containers:
      - name: kubedns
        image: gcr.io/google_containers/k8s-dns-kube-dns-amd64:1.14.4
        resources:
          # TODO: Set memory limits when we've profiled the container for large
          # clusters, then set request = limit to keep this container in
          # guaranteed class. Currently, this container falls into the
          # "burstable" category so the kubelet doesn't backoff from restarting it.
          limits:
            memory: 170Mi
```

```
    requests:
      cpu: 100m
      memory: 70Mi
  livenessProbe:
    httpGet:
      path: /healthcheck/kubedns
      port: 10054
      scheme: HTTP
    initialDelaySeconds: 60
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 5
  readinessProbe:
    httpGet:
      path: /readiness
      port: 8081
      scheme: HTTP
    # we poll on pod startup for the Kubernetes master service and
    # only setup the /readiness HTTP server once that's available.
    initialDelaySeconds: 3
    timeoutSeconds: 5
  args:
  - --domain=cluster.local.
  - --dns-port=10053
  - --config-dir=/kube-dns-config
  - --v=2
  env:
  - name: PROMETHEUS_PORT
    value: "10055"
```

```
        ports:
        - containerPort: 10053
          name: dns-local
          protocol: UDP
        - containerPort: 10053
          name: dns-tcp-local
          protocol: TCP
        - containerPort: 10055
          name: metrics
          protocol: TCP
        volumeMounts:
        - name: kube-dns-config
          mountPath: /kube-dns-config
      - name: dnsmasq
        image: gcr.io/google_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.4
        livenessProbe:
          httpGet:
            path: /healthcheck/dnsmasq
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
```

```
        args:
        - -v=2
        - -logtostderr
        - -configDir=/etc/k8s/dns/dnsmasq-nanny
        - -restartDnsmasq=true
        - --
        - -k
        - --cache-size=1000
        - --log-facility=-
        - --server=/cluster.local/127.0.0.1#10053
        - --server=/in-addr.arpa/127.0.0.1#10053
        - --server=/ip6.arpa/127.0.0.1#10053
        ports:
        - containerPort: 53
          name: dns
          protocol: UDP
        - containerPort: 53
          name: dns-tcp
          protocol: TCP
        # see: https://github.com/kubernetes/kubernetes/issues/29055 for details
        resources:
          requests:
            cpu: 150m
            memory: 20Mi
```

```
        volumeMounts:
        - name: kube-dns-config
          mountPath: /etc/k8s/dns/dnsmasq-nanny
      - name: sidecar
        image: gcr.io/google_containers/k8s-dns-sidecar-amd64:1.14.4
        livenessProbe:
          httpGet:
            path: /metrics
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        args:
        - --v=2
        - --logtostderr
        - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local,5,A
        - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local,5,A
        ports:
        - containerPort: 10054
          name: metrics
          protocol: TCP
        resources:
          requests:
            memory: 20Mi
            cpu: 10m
      dnsPolicy: Default  # Don't use cluster DNS.
      serviceAccountName: kube-dns
```

创建 kubedns-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "KubeDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 10.254.0.2
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP
```

创建 kubedns-sa.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
```

部署dns-horizontal-autoscaler

创建 dns-horizontal-autoscaler-rbac.yaml

```
kind: ServiceAccount
apiVersion: v1
metadata:
  name: kube-dns-autoscaler
  namespace: kube-system
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: system:kube-dns-autoscaler
```

```
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["list"]
  - apiGroups: [""]
    resources: ["replicationcontrollers/scale"]
    verbs: ["get", "update"]
  - apiGroups: ["extensions"]
    resources: ["deployments/scale", "replicasets/scale"]
    verbs: ["get", "update"]
# Remove the configmaps rule once below issue is fixed:
# kubernetes-incubator/cluster-proportional-autoscaler#16
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "create"]
---
```

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: system:kube-dns-autoscaler
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
subjects:
  - kind: ServiceAccount
    name: kube-dns-autoscaler
    namespace: kube-system
roleRef:
  kind: ClusterRole
  name: system:kube-dns-autoscaler
  apiGroup: rbac.authorization.k8s.io
```

创建 dns-horizontal-autoscaler.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kube-dns-autoscaler
  namespace: kube-system
  labels:
    k8s-app: kube-dns-autoscaler
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  template:
    metadata:
      labels:
        k8s-app: kube-dns-autoscaler
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
```

```
spec:
  containers:
  - name: autoscaler
    image: gcr.io/google_containers/cluster-proportional-autoscaler-amd64:1.1.2-r2
    resources:
        requests:
            cpu: "20m"
            memory: "10Mi"
    command:
      - /cluster-proportional-autoscaler
      - --namespace=kube-system
      - --configmap=kube-dns-autoscaler
      # Should keep target in sync with cluster/addons/dns/kubedns-controller.yaml.base
      - --target=Deployment/kube-dns
      # When cluster is using large nodes(with more cores), "coresPerReplica" should dominate.
      # If using small nodes, "nodesPerReplica" should dominate.
      - --default-params={"linear":{"coresPerReplica":256,"nodesPerReplica":16,"preventSinglePointFailure":true}}
      - --logtostderr=true
      - --v=2
  tolerations:
  - key: "CriticalAddonsOnly"
    operator: "Exists"
  serviceAccountName: kube-dns-autoscaler
```

创建 dashboard-rbac

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dashboard
  namespace: kube-system

---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: dashboard
subjects:
  - kind: ServiceAccount
    name: dashboard
    namespace: kube-system
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

创建 dashboard-controller

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
```

```
  spec:
    containers:
    - name: kubernetes-dashboard
      image: gcr.io/google_containers/kubernetes-dashboard-amd64:v1.6.1
      resources:
        # keep request = limit to keep this container in guaranteed class
        limits:
          cpu: 100m
          memory: 300Mi
        requests:
          cpu: 100m
          memory: 100Mi
      ports:
      - containerPort: 9090
      args:
      - --apiserver-host=http://172.16.200.100:8080
      livenessProbe:
        httpGet:
          path: /
          port: 9090
        initialDelaySeconds: 30
        timeoutSeconds: 30
    tolerations:
    - key: "CriticalAddonsOnly"
      operator: "Exists"
```

创建 dashboard-service

```
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  type: NodePort
  selector:
    k8s-app: kubernetes-dashboard
  ports:
  - port: 80
    targetPort: 9090
```

RBAC Support in Kubernetes

Kubernetes 中的 RBAC 支持

> PS：在Kubernetes1.6版本中新增角色访问控制机制（Role-Based Access，RBAC）让集群管理员可以针对特定使用者或服务账号的角色，进行更精确的资源访问控制。在RBAC中，权限与角色相关联，用户通过成为适当角色的成员而得到这些角色的权限。这就极大地简化了权限的管理。在一个组织中，角色是为了完成各种工作而创造，用户则依据它的责任和资格来被指派相应的角色，用户可以很容易地从一个角色被指派到另一个角色。

RBAC vs ABAC

鉴权的作用是，决定一个用户是否有权使用 Kubernetes API 做某些事情。它除了会影响 kubectl 等组件之外，还会对一些运行在集群内部并对集群进行操作的软件产生作用，例如使用了 Kubernetes 插件的 Jenkins，或者是利用 Kubernetes API 进行软件部署的 Helm。ABAC 和 RBAC 都能够对访问策略进行配置。

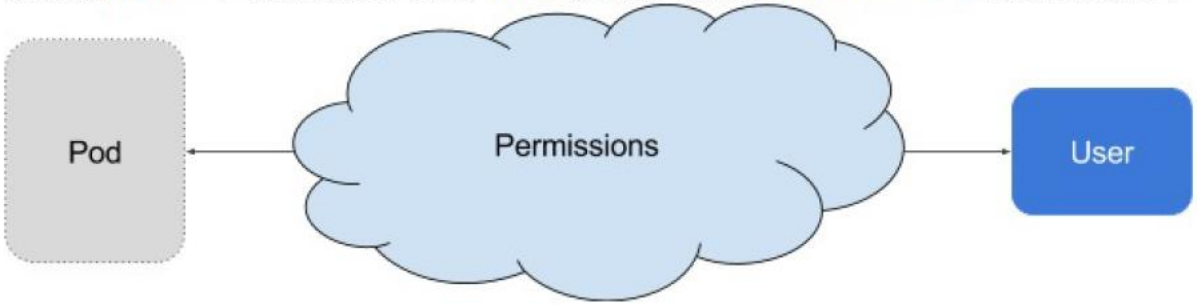
ABAC（Attribute Based Access Control）本来是不错的概念，但是在 Kubernetes 中的实现比较难于管理和理解（怪我咯），而且需要对 Master 所在节点的 SSH 和文件系统权限，而且要使得对授权的变更成功生效，还需要重新启动 API Server。

而 RBAC 的授权策略可以利用 kubectl 或者 Kubernetes API 直接进行配置。RBAC 可以授权给用户，让用户有权进行授权管理，这样就可以无需接触节点，直接进行授权管理。RBAC 在 Kubernetes 中被映射为 API 资源和操作。

因为 Kubernetes 社区的投入和偏好，相对于 ABAC 而言，RBAC 是更好的选择。

基础概念

需要理解 RBAC 一些基础的概念和思路，RBAC 是让用户能够访问 Kubernetes API 资源的授权方式。



在 RBAC 中定义了两个对象，用于描述在用户和资源之间的连接权限。

Role and ClusterRole

在 RBAC API 中，Role 表示一组规则权限，权限只会增加(累加权限)，不存在一个资源一开始就有很多权限而通过 RBAC 对其进行减少的操作；Role 可以定义在一个 namespace 中，如果想要跨 namespace 则可以创建 ClusterRole。

角色

角色是一系列的权限的集合，例如一个角色可以包含读取 Pod 的权限和列出 Pod 的权限， ClusterRole 跟 Role 类似，但是可以在集群中到处使用. **Role 只能用于授予对单个命名空间中的资源访问权限**. 以下是一个对默认命名空间中 Pods 具有访问权限的样例:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

ClusterRole 具有与 Role 相同的权限角色控制能力，不同的是 ClusterRole 是集群级别的，ClusterRole 可以用于:

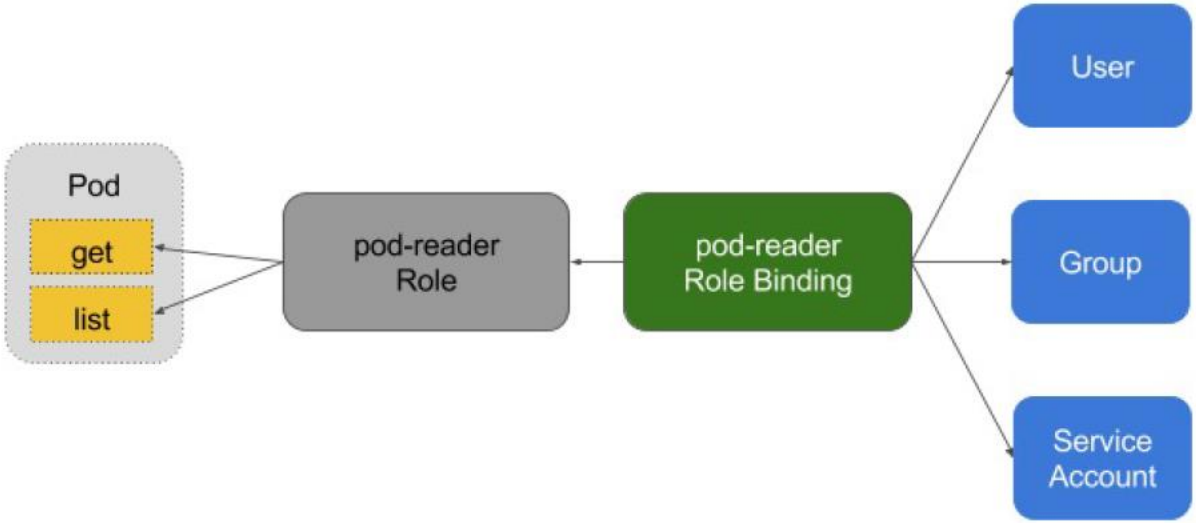
- 集群级别的资源控制(例如 node 访问权限)
- 非资源型 endpoints(例如 /healthz 访问)
- 所有命名空间资源控制(例如 pods)

以下是 ClusterRole 授权某个特定命名空间或全部命名空间(取决于绑定方式)访问 secrets 的样例

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

RoleBinding and ClusterRoleBinding

RoloBinding 可以将角色中定义的权限授予用户或用户组，RoleBinding 包含一组权限列表(subjects)，**权限列表中包含有不同形式的待授予权限资源类型(users, groups, or service accounts)**；RoloBinding 同样包含对被 Bind 的 Role 引用；RoleBinding 适用于某个命名空间内授权，而 ClusterRoleBinding 适用于集群范围内的授权。



RoleBinding 可以在同一命名空间中引用对应的 Role，以下 RoleBinding 样例将 default 命名空间的 pod-reader Role 授予 jane 用户，此后 jane 用户在 default 命名空间中将具有 pod-reader 的权限

```
# This role binding allows "jane" to read pods in the "default" namespace.
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

RoleBinding 同样可以引用 ClusterRole 来对当前 namespace 内用户、用户组或 ServiceAccount 进行授权，这种操作允许集群管理员在整个集群内定义一些通用的 ClusterRole，然后在不同的 namespace 中使用 RoleBinding 来引用

例如，以下 RoleBinding 引用了一个 ClusterRole，这个 ClusterRole 具有整个集群内对 secrets 的访问权限；但是其授权用户 dave 只能访问 development 空间中的 secrets(因为 RoleBinding 定义在 development 命名空间)

```
# This role binding allows "dave" to read secrets in the "development" namespace.
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-secrets
  namespace: development # This only grants permissions within the "development" namespace.
subjects:
- kind: User
  name: dave
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

最后，使用 ClusterRoleBinding 可以对整个集群中的所有命名空间资源权限进行授权；以下 ClusterRoleBinding 样例展示了授权 manager 组内所有用户在全部命名空间中对 secrets 进行访问

```
# This cluster role binding allows anyone in the "manager" group to read secrets in any namespace.
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

Referring to Resources

Kubernetes 集群内一些资源一般以其名称字符串来表示，这些字符串一般会在 API 的 URL 地址中出现；同时某些资源也会包含子资源，例如 logs 资源就属于 pods 的子资源，API 中 URL 样例如下

```
GET /api/v1/namespaces/{namespace}/pods/{name}/log
```

如果要在 RBAC 授权模型中控制这些子资源的访问权限，可以通过 / 分隔符来实现，以下是一个定义 pods 资资源 logs 访问权限的 Role 定义样例

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: pod-and-pod-logs-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list"]
```

具体的资源引用可以通过 resourceNames 来定义，当指定 get、delete、update、patch 四个动词时，可以控制对其目标资源的相应动作；以下为限制一个 subject 对名称为 my-configmap 的 configmap 只能具有 get 和 update 权限的样例

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: configmap-updater
rules:
- apiGroups: [""]
  resources: ["configmap"]
  resourceNames: ["my-configmap"]
  verbs: ["update", "get"]
```

值得注意的是，当设定了 resourceNames 后，verbs 动词不能指定为 list、watch、create 和 deletecollection；因为这个具体的资源名称不在上面四个动词限定的请求 URL 地址中匹配到，最终会因为 URL 地址不匹配导致 Role 无法创建成功

Referring to Subjects

RoleBinding 和 ClusterRoleBinding 可以将 Role 绑定到 Subjects；Subjects 可以是 groups、users 或者 service accounts。

Subjects 中 Users 使用字符串表示，它可以是一个普通的名字字符串，如 “alice”；也可以是 email 格式的邮箱地址，如 “bob@example.com”；甚至是一组字符串形式的数字 ID。Users 的格式必须满足集群管理员配置的验证模块，RBAC 授权系统中没有对其做任何格式限定；但是 Users 的前缀 system: 是系统保留的，集群管理员应该确保普通用户不会使用这个前缀格式

Kubernetes 的 Group 信息目前由 Authenticator 模块提供，Groups 书写格式与 Users 相同，都为一个字符串，并且没有特定的格式要求；同样 system: 前缀为系统保留

具有 system:serviceaccount: 前缀的用户名和 system:serviceaccounts: 前缀的组为 Service Accounts

Role Binding Examples

以下示例仅展示 RoleBinding 的 subjects 部分

指定一个名字为 alice@example.com 的用户

```
subjects:
- kind: User
  name: "alice@example.com"
  apiGroup: rbac.authorization.k8s.io
```

指定一个名字为 frontend-admins 的组

```
subjects:
- kind: Group
  name: "frontend-admins"
  apiGroup: rbac.authorization.k8s.io
```

指定 kube-system namespace 中默认的 Service Account

```
subjects:
- kind: ServiceAccount
  name: default
  namespace: kube-system
```

指定在 qa namespace 中全部的 Service Account

```
subjects:
- kind: Group
  name: system:serviceaccounts:qa
  apiGroup: rbac.authorization.k8s.io
```

指定全部 namspace 中的全部 Service Account

```
subjects:
- kind: Group
  name: system:serviceaccounts
  apiGroup: rbac.authorization.k8s.io
```

指定全部的 authenticated 用户(1.5+)

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

指定全部的 unauthenticated 用户(1.5+)

```
subjects:
- kind: Group
  name: system:unauthenticated
  apiGroup: rbac.authorization.k8s.io
```

指定全部用户

```
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
- kind: Group
  name: system:unauthenticated
  apiGroup: rbac.authorization.k8s.io
```