79Kubernetes 系列（七十二）揭开 RBAC 的神秘面纱：深入了解基于角色的访问控制

介绍

在容器编排领域，Kubernetes 已成为大规模管理和部署容器化应用程序的事实标准。凭借其灵活的架构和强大的功能集，Kubernetes 提供了强大的工具来管理访问和保护集群。Kubernetes 中的一个关键安全功能是基于角色的访问控制（RBAC）。在这篇博文中，我们将探讨 Kubernetes RBAC 及其组件的来龙去脉，以及如何利用它来在集群中实施细粒度访问控制。

了解 RBAC

基于角色的访问控制 （RBAC） 是一种安全机制，它提供了一种精细的方法来管理对系统中资源的访问。RBAC 允许管理员定义具有特定权限的角色，并将这些角色分配给用户或组。在 Kubernetes 的上下文中，RBAC 使群集管理员能够控制谁可以对群集中的各种资源执行某些操作。

Kubernetes RBAC 的组件

1. **角色**：角色是一组规则，用于定义特定命名空间中的一组权限。它可用于授予或限制对 API 对象（如 Pod、服务、部署等）的访问权限。角色特定于命名空间，不能跨命名空间使用。

2. **群集角色（ClusterRole）**：与角色类似，群集角色是一组定义权限的规则，但它们不是特定于命名空间的。群集角色可用于授予或限制对跨多个命名空间或整个群集的资源的访问权限。

3. **角色绑定(RoleBinding)**：角色绑定将角色或群集角色绑定到一个或多个用户、组或服务帐户。它在角色/群集角色中定义的一组权限与应具有这些权限的实体之间建立连接。

4. **ClusterRoleBindings**：就像 RoleBindings 一样，ClusterRoleBindings 将 ClusterRoles 绑定到用户、组或服务帐户。但是，群集角色绑定将权限应用于整个群集，而不是局限于特定的命名空间。

实施访问控制

通过利用 RBAC，可以定义适合组织安全要求的细粒度访问控制策略。下面是可以使用 RBAC 的一些常见方案：

1. **角色隔离**：RBAC 允许您通过将不同的角色分配给不同的团队或个人来分离职责。例如，可以将“开发人员”角色分配给仅有权创建和管理部署的团队，而“操作员”角色可能具有管理底层基础结构的权限。

2. **有限的命名空间访问**：RBAC 使你能够限制对特定命名空间的访问。当多个团队或项目共享一个群集，并且您希望隔离其资源并限制其指定命名空间中的特权时，这可能很有用。

3. **服务帐户**：RBAC 可用于控制服务帐户的权限，这些权限由在 Pod 中运行的应用程序用于与 Kubernetes API 交互。通过为服务帐户分配适当的角色，您可以控制他们可以执行的操作。

RBAC 的最佳实践

为了充分利用 Kubernetes RBAC，请考虑以下最佳实践：

1. **最小特权原则**：定义角色时遵循最小特权原则。仅授予执行特定任务所需的最低权限，以最大程度地降低未经授权操作的风险。

2. **定期审核**：定期查看和审核 RBAC 配置，以确保它们符合组织的安全策略。删除不必要或过多的权限，并根据需要更新角色。

3. **有效使用命名空间**：利用命名空间在逻辑上隔离不同的团队或项目，并在命名空间级别强制实施 RBAC 策略。

4. **测试 RBAC 策略**：在非生产环境中全面测试 RBAC 策略，以确保它们在应用于生产群集之前按预期运行。

下面是一个说明如何实现 Kubernetes RBAC 的示例：

假设我们有一个具有多个命名空间的 Kubernetes 集群，我们希望配置 RBAC，以便为组织内的不同团队提供不同级别的访问权限。

1. **定义角色**：我们首先定义指定每个团队所需权限的角色。例如，我们可以创建两个角色：“开发人员”和“操作员”。

“开发人员”角色可能有权在特定命名空间中创建和管理部署、服务和 Pod。“开发人员”角色的 YAML 定义可能如下所示：

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: my-namespace
  name: developer-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "services", "pods"]
  verbs: ["create", "get", "update", "delete", "list", "watch"]
```

同样，“操作员”角色可能有权管理同一命名空间中的持久卷、机密和配置映射：

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: my-namespace
  name: operator-role
rules:
- apiGroups: [""]
  resources: ["persistentvolumes", "secrets", "configmaps"]
  verbs: ["create", "get", "update", "delete", "list", "watch"]
```

2. 创建角色绑定：接下来，我们需要创建角色绑定以将这些角色与相应的用户、组或服务帐户相关联。例如，让我们创建一个 RoleBinding，将“开发人员”角色分配给“my-namespace”命名空间中的“开发团队”组：

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: my-namespace
  name: developer-rolebinding
subjects:
- kind: Group
  name: development-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer-role
  apiGroup: rbac.authorization.k8s.io
```

类似地，我们可以创建一个 RoleBinding，将“operator”角色分配给同一命名空间中名为“operator-user”的特定用户：

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: my-namespace
  name: operator-rolebinding
subjects:
- kind: User
  name: operator-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: operator-role
  apiGroup: rbac.authorization.k8s.io
```

3. **应用 RBAC 配置**：定义角色和角色绑定后，可以使用 kubectl apply 命令应用 RBAC 配置：

```
kubectl apply -f developer-role.yaml
kubectl apply -f operator-role.yaml
kubectl apply -f developer-rolebinding.yaml
kubectl apply -f operator-rolebinding.yaml
```

应用这些配置后，“开发团队”组将具有“开发人员”角色中定义的权限，“操作员用户”将具有“my-namespace”命名空间内的“操作员”角色中定义的权限。

此示例展示了如何使用 RBAC 向 Kubernetes 群集中的不同团队或个人授予特定权限。通过组合角色和角色绑定，可以定义和实施符合组织要求的访问控制策略。请记住，随着群集的发展和新的访问要求的出现，定期查看和更新 RBAC 配置。

总结

基于角色的访问控制 （RBAC） 是保护 Kubernetes 集群的重要组成部分。通过实现 RBAC，可以控制对各种资源的访问并强制实施细化权限。了解 RBAC 的组件并遵循最佳做法将有助于设计符合组织安全要求的可靠访问控制策略。借助 RBAC，您可以在授予必要的权限和维护 Kubernetes 环境的完整性和安全性之间取得平衡。