Kubernetes from Basic to Advanced

kubernetes

Session #04
Security

kubernetes

Session Contents

- RBAC
- Service Accounts
- Quotas

kubernetes

RBAC

Motivation

- As any system, you want to have a control about who access your resources and which type of action can perform
- Your cluster can be reached by human users and services
- Human users are the operator who access the cluster using API Server (through kubectl)
- Services are your pods running inside your cluster
- Role-based Access Control (RBAC) is the implementation of authn/authz in Kubernetes

RBAC

- Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization
- RBAC authorization uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing you to dynamically configure policies through the Kubernetes API
- By default, RBAC is not enabled on a Kubernetes cluster. You need to explicitly enable it

kubernetes

RBAC Objects

- RBAC API declares 4 objects: Role, ClusterRole, RoleBinding and ClusterRoleBinding
- Role and ClusterRole contains rules that represent a set of permissions that are always additive (there are no "deny" rules)
- RoleBinding and ClusterRoleBinding grants the permissions defined in a role to a user or set of users

kubernetes

Role and ClusterRole

- Role and ClusterRole contains rules that represent a set of permissions that are always additive (there are no "deny" rules)
- A Role always sets permissions within a particular namespace; ClusterRole, by contrast, is a non-namespaced resource
- You can use a ClusterRole to:
  - Define permissions on namespaced resources and be granted access within individual namespace(s)
  - Define permissions on namespaced resources and be granted access across all namespaces
  - Define permissions on cluster-scoped resources

kubernetes

Role and ClusterRole Manifest

- When creating Role or ClusterRole manifest you create a list of rules
- For each rules you need to define
  - `apiGroups`
  - `resources`
  - `verbs`
  - Optionally, `resourceNames`
- To get a list of available values use
  - `kubectl api-resources -o wide`

kubernetes

Role and ClusterRole

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: deployments-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["apps"]
  resources: ["replicasets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["apps"]
  resources: ["deployments"
  verbs: ["get", "watch", "list"]
```

kubernetes

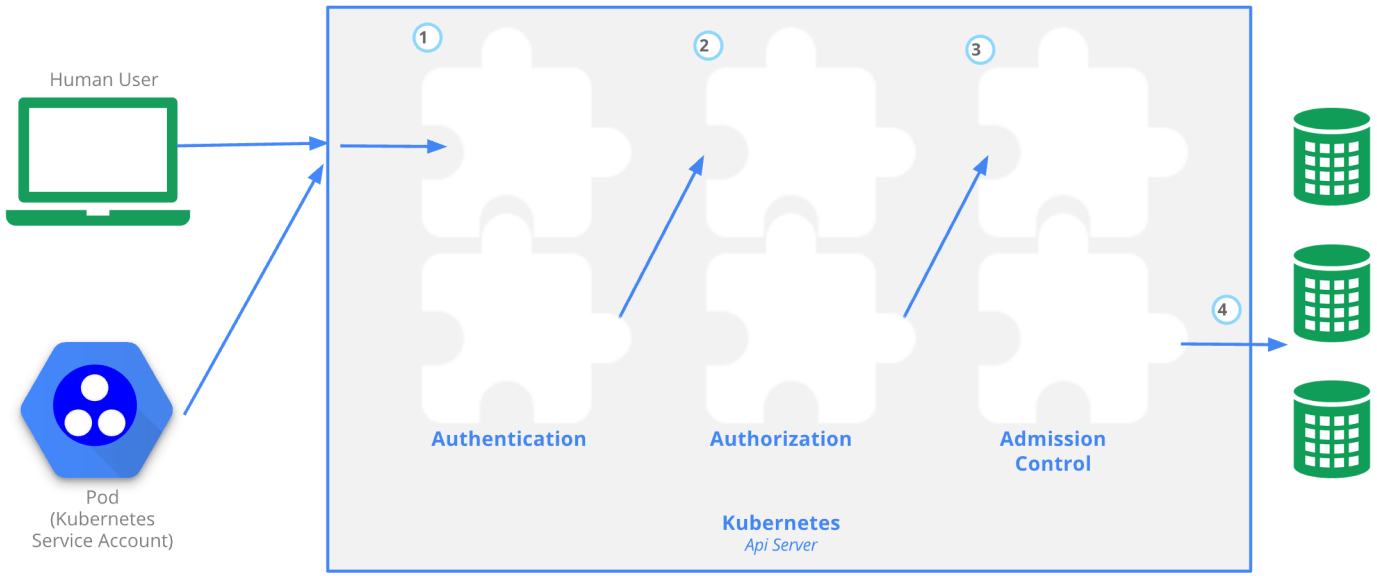
RoleBinding and ClusterRoleBinding

- RoleBinding and ClusterRoleBinding grants the permissions defined in a role to a user or set of users
- It holds a list of subjects (users, groups, or service accounts), and a reference to the role being granted
- A RoleBinding grants permissions within a specific namespace whereas a ClusterRoleBinding grants that access cluster-wide
- A RoleBinding may reference any Role in the same namespace or reference a ClusterRole and bind that ClusterRole to the namespace of the RoleBinding
- If you want to bind a ClusterRole to all the namespaces in your cluster, you use a ClusterRoleBinding

kubernetes

Binding Subjects

- When creating a binding, you can specify 3 different subjects: users, groups and ServiceAccounts



Users and Groups

- Kubernetes don't have a traditional user and groups definition
- To add a user, you need to use one of the available authentication methods
  - X509 Client Certs
  - Static Token File
  - Bootstrap Tokens
  - Service Account Tokens
  - OpenID Connect Tokens
  - Webhook Token Authentication
  - Authenticating Proxy
- X509 Client Certs is the preferable way to create local users

kubernetes

Users and Groups

- X509 Client Certs is the preferable way to create local users
- OpenID Connect allow you to connect your cluster with an identity external provider
- Groups are not defined directly on cluster configuration
- On X509 strategy, you can define the group(s) on key definition

kubernetes

Service Accounts

- A Service Account is a type of non-human account that provides a distinct identity in a Kubernetes cluster
- Pods can use a specific ServiceAccount's credentials to identify as that ServiceAccount
- This identity is useful in various situations, including authenticating to the API server or implementing identity-based security policies
- Service Accounts are a namespaced resource
- Every namespace have a default Service Account that is automatically used when any is configured on your resources

kubernetes

Service Accounts Use Cases

- Your pod needs to use API Server and you want to control resources and verbs to be used
- Access a private registry to fetch container images
- Third-party software installed on your cluster needs a Service Account to run

kubernetes

Demo | RBAC

Resource Quotas

Motivation

- Using a Cluster you are sharing all resources by everyone using the nodes
- This is a concern that one team could use more than its fair share of resources.
- How to explicitly make boundaries on the resources each team can use?
- Resource quotas are a tool for administrators to address this concern.

kubernetes

Resource Quotas

- Resource quotas are a tool for administrators to address this concern
- A resource quota, defined by a `ResourceQuota` object, provides constraints that limit aggregate resource consumption per namespace
- It can limit the quantity of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed by resources in that namespace

How it works

- Different teams work in different namespaces. This can be enforced with RBAC.
- The administrator creates one ResourceQuota for each namespace.
- Users create resources (pods, services, etc.) in the namespace, and the quota system tracks usage to ensure it does not exceed hard resource limits defined in a ResourceQuota.
- If creating or updating a resource violates a quota constraint, the request will fail with HTTP status code 403 FORBIDDEN with a message explaining the constraint that would have been violated.
- If quota is enabled in a namespace for compute resources like cpu and memory, users must specify requests or limits for those values; otherwise, the quota system may reject pod creation.

kubernetes

Sample Scenarios

- In a cluster with a capacity of 32 GiB RAM, and 16 cores, let team A use 20 GiB and 10 cores, let B use 10GiB and 4 cores, and hold 2GiB and 2 cores in reserve for future allocation.
- Limit the "testing" namespace to using 1 core and 1GiB RAM. Let the "production" namespace use any amount.

kubernetes

Constraints

- In the case where the total capacity of the cluster is less than the sum of the quotas of the namespaces, there may be contention for resources. This is handled on a first-come-first-served basis
- When a namespace has a ResourceQuota object defined creating a restriction on resource usage, all your pods should have a request and limit explicitly defined for memory and cpu

kubernetes

Resources Quota Object

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

kubernetes

Resources Quota Object

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-demo
spec:
  hard:
    pods: "2"
```

kubernetes

Demo | Quotas

Questions?

kubernetes