Kubernetes Security Fundamentals

Authorization

Authorization in Kubernetes is a critical mechanism that determines what actions authenticated users and applications can perform on cluster resources. It ensures that only authorized entities can access and modify cluster objects, thereby protecting the cluster from unauthorized operations.

Kubernetes supports several authorization mechanisms to control access to cluster resources. The most commonly used mechanism is Role-Based Access Control (RBAC), which defines permissions based on roles assigned to users and applications. Other mechanisms include Attribute-Based Access Control (ABAC) and webhook authorization.

RealLife Example:

Imagine a well-guarded city with a secured gatehouse and an organized system for granting access within the city itself. Authentication verifies a visitor's identity at the gate, but authorization determines where they can go and what they can do within the city limits. Similarly, in Kubernetes, authorization dictates what actions authenticated users are allowed to perform after they've proven their identity.

Why is Authorization Critical in Kubernetes?

- **Least Privilege Principle:** A core security principle that grants users only the minimum permissions necessary to complete their tasks. Imagine a delivery person needing access only to loading zones, not the city's power grid control center. Authorization ensures this by restricting access based on user roles.
- **Reduced Attack Surface:** By limiting user permissions, authorization minimizes the potential damage caused by accidental errors or malicious actors. Even if an attacker gains access, their ability to wreak havoc is restricted by their assigned permissions. Think of a lost visitor badge granting access only to public areas, not restricted government buildings.
- **Improved Security Posture:** A robust authorization system strengthens your cluster's overall security posture by preventing unauthorized access to critical resources and functionalities. Imagine a

well-defined permit system preventing unauthorized individuals from operating heavy machinery or accessing sensitive documents.

How Does Authorization Work in Kubernetes?

Kubernetes leverages Role-Based Access Control (RBAC) for authorization. Here's the breakdown:

- **Roles:** Define sets of permissions, like "admin", "developer", or "view-only". Think of roles as different job titles within the city, each with specific associated permissions.
- **ClusterRoles:** Similar to roles, but for cluster-wide access. Imagine a "city-wide maintenance worker" role allowing access to all public areas across the city.
- **RoleBindings/ClusterRoleBindings:** Assign roles or cluster roles to users or groups. Think of issuing badges or permits to specific individuals based on their job titles (roles).
- **Service Account:** A special account used by pods to access resources within the cluster. Imagine service accounts as departmental IDs used by different city departments (like sanitation or parks department) to access specific resources.
- **Subject Access Review (SAR):** Allows ad-hoc authorization checks for specific users or groups. Think of a temporary permit granted to a film crew for controlled access to a specific area for shooting.

Best Practices for Authorization in Kubernetes:

- **Implement RBAC with least privilege:** Grant users only the permissions they absolutely need. Start with the most restrictive settings and gradually increase permissions as required.
- **Utilize predefined roles:** Kubernetes provides predefined roles for common tasks, saving you time and effort. Think of leveraging standardized permit templates for different job functions.
- **Regularly review and update RBAC configurations:** As your cluster and user base evolve, ensure RBAC reflects current needs and permissions are not overly broad. Periodically review permits and access levels to identify outdated or unnecessary permissions.

• **Monitor API Server logs:** Track user activity and identify any suspicious attempts to access unauthorized resources. Monitor activity logs to identify potential misuse of permits or unauthorized access attempts.

Key Concepts

1. **Role-Based Access Control (RBAC)**
   ◦ RBAC uses roles and role bindings to define permissions.
   ◦ Roles define what actions can be performed on specific resources.
   ◦ Role bindings associate roles with users, groups, or service accounts.
2. **Attribute-Based Access Control (ABAC)**
   ◦ ABAC uses policies that include user attributes, resource attributes, and environment attributes to determine access permissions.
3. **Webhook Authorization**
   ◦ Webhook authorization allows for custom authorization policies by calling an external service.
4. **Node Authorization**
   ◦ Node authorization restricts access for kubelets based on the nodes they manage.

Security Best Practices

1. **Use RBAC for Fine-Grained Control**
   ◦ Implement RBAC to define detailed permissions for users and applications.
   ◦ Regularly review and update roles and role bindings.
2. **Principle of Least Privilege**
   ◦ Grant the minimum permissions necessary for users and applications to perform their tasks.
   ◦ Avoid assigning cluster-admin privileges unless absolutely necessary.
3. **Monitor and Audit Authorization Logs**
   ◦ Enable audit logging to track authorization decisions and detect unauthorized access attempts.
   ◦ Regularly review logs and set up alerts for suspicious activities.

4. **Implement Policy Automation**
    - Use tools and scripts to automate the management of RBAC policies and ensure consistency.
5. **Secure Webhook Authorization**
    - Ensure webhook authorization services are highly available and secure.
    - Use mutual TLS to encrypt communication between the API Server and the webhook service.

Note: Follow the official Documentation for lab practice, no exclusive Lab Exercise for Authentication and Authorization.