Problem Statement: Enterprise Control for K8S

raew@google.com, rayc@google.com March 2017

Public Document

Customer Story

A SaaS provider builds an online retail platform both for their own stores and for their customers' stores. Their customers are the retailers who run stores on this platform. The SaaS provider likes the appeal of K8S and want to use a single cluster for the many tenants on their platform, running a dedicated stack per tenant. This way they can optimize resource usage and have a central place for managing the infrastructure. The software for each store is run by allied teams who are part of the same organization.

However they have the following policy needs

- Each store should only be able to access their own data
- Each store can have quota set separately to prevent abuse
- Each store's cost can be tracked separately so that customers (other retailers on their platform) can be billed

The different departments in the company operate as autonomous units with some central constraints

- Each team should have separate permissions, quota, cost analysis (similar requirements for the different stores above)
- Different teams (eg. HR vs R&D) and different environments (eg. dev/test vs prod) can also set different configuration constraints. For example production environments should only have the minimal set of services enabled, only use approved VM images etc. Whereas dev/test can offer more flexibility

Analysis

K8S allows customers to binpack many services to a cluster and manage the lifecycle of these services and clusters. Enterprise customers are realizing that like other resources, a K8S cluster doesn't exist in isolation. How multiple departments, teams, services interact with each other become an important part of the enterprise K8S experience. Some common challenges

- Multiple services run on the same cluster are owned by different teams and therefore need separate policies such as billing, quota, IAM etc at service granularity.

- A company owns multiple clusters - eg. required by compliance, to minimize blast radius, to be close to customers, etc. The multiple clusters still need to be centrally visible and controlled by the organization's admin team. Workloads running on clusters need to adhere to company wide policies (eg. which images are allowed, whether external IP can be set up etc)
    - Single services running across multiple clusters should have one set of policies, have their usage be on the same bill, etc.
    - Identities and authentication are centralized among all clusters running for a company.
    - The ability to partition out a cluster to certain teams. E.g. X team gets access to the european and asian clusters but Y team does not.

This is aligned with enterprise control needs for Cloud in general - the flexibility of granular policies + the power of central control

Related

Cluster federation provides a single API entry and global domain/namespace across multiple clusters. It enables services across clusters to communicate with each other and simplifies deployments across clusters. A cluster joining a federation is voluntary. Therefore federation alone doesn't provide organization wide enforcement. An employee can create a cluster without joining the federation and therefore the admin won't know about it.

In addition to a long history of hierarchical configuration in Active Directory, all three major cloud providers (aws, gcp, azure) are converging on this model for management of resources. AWS recently released "AWS organizations", GCP has the "Organization Hierarchy" and Azure has "Resource Groups". This suggests that customers will be well conditioned to using hierarchies to manage cloud resources. Aligning Kubernetes with this model would make cross-cloud deployments easier.

OpenShift 3 syncs groups from LDAP. OpenShift 3 is built on Kubernetes, so its patterns should be directly applicable.
https://docs.openshift.com/enterprise/3.1/install_config/syncing_groups_with_ldap.html

CoreOS Tectonic can extract LDAP groups, via Dex, and map them to RBAC Roles via Kubernetes OIDC support.

Some requirements

- Hierarchical namespace configuration that allows per-namespace Auth and Resource Quotas to be defined at multiple levels.
- Authentication is centralized among all clusters and compatible with existing on-prem and cloud authentication systems (Any OIDC Provider, AD, Okta, Ping, etc)

- Ability to hierarchically set restrictions on configuration such as allowed docker repos etc.

K8S primitives support

- There is a namespace controller.
- When it is activated (the cluster is enrolled):
    - The cluster is configured to give the namespace controller permission to create and delete namespaces.
    - The cluster is configured to deny users access to CRUD namespaces. They cannot use kube commands directly on a cluster to create namespaces.
    - Users can only create namespaces by editing an external database (e.g. LDAP). This database may be selfserve or centrally controlled which is a matter of IT policy.
    - The Namespace Controller syncs the namespaces to the cluster from LDAP.
    - There is no hierarchy that Kubernetes core is aware of. Hierarchy exists only in the LDAP database and the things that control what it contains TBD: is there hierarchy discoverable from the individual namespace names (via name, annotation, label, etc?) Or only by consulting the LDAP database?
- It seems possible to reuse much of the work in Cluster Federation to sync namespaces into clusters.
- The namespace provider could also sync resource quota from LDAP. Maybe also configure what are the allowed docker repos, and other configuration stuff.

Strawman for LDAP Implementation using new K8S primitives

- Create an LDAP Namespace Provider that allows hierarchal storage of configuration. LDAP is an open source directory service protocol that is still widely supported by Active Directory and various open source implementations including OpenLDAP and Apache Directory Service.
    - Create a service that can read all the namespaces out of LDAP and sync them with every enrolled cluster.
- Create a service that can use LDAP as the source of truth for identities.
    - We would use standard identity schemas (inetorgperson, etc) where possible.
    - This service would integrate with K8S authentication.
- Develop a new set of custom K8S LDAP object classes and attributes to define a namespace and the types of attributes that are allowed to be set on a namespace..
    - For orgs that have existing directories, they can add the K8S schema to their directories and use their own business processes for configuring cluster namespace and their policies.

- Write a webhook authorizer that uses LDAP hierarchies to provide authorization to users.
    - Each namespace and its parents would have an authorization policy attached that is inherited. This allows centralized authorization to all clusters.
    - Within the LDAP directory, the top level DN becomes the root node for all K8S namespaces giving Admin Control.
- We develop admission controllers that talk to LDAP and read the custom schema to enforce hierarchical configuration.
- It should be possible using different directory domains to isolate clusters from one another.

Open Questions

- Do we use a webhook authorizer and delegate authorization to an external system, or do we use K8S RBAC and sync permissions into K8S clusters.
    - A downside of RBAC: It seems that RBAC stores policies locally in K8S clusters and therefore would requiring syncing from a master system. This could lead to permissions revocations failing or taking a while to reach eventual consistency with the master store.
    - I don't think you want to remove RBAC entirely because then it is hard to handle user-provided APIs, without putting LDAP in the loop. May need some exceptions where users can add RBAC that is not synced, as long as it is not any more powerful than LDAP-based policy would allow.
    -
- Can we use admission controllers to:
    - restrict quota? Yes, can sync to ResourceQuota objects from LDAP or other DB.
    - allowed repos? Use webhook.