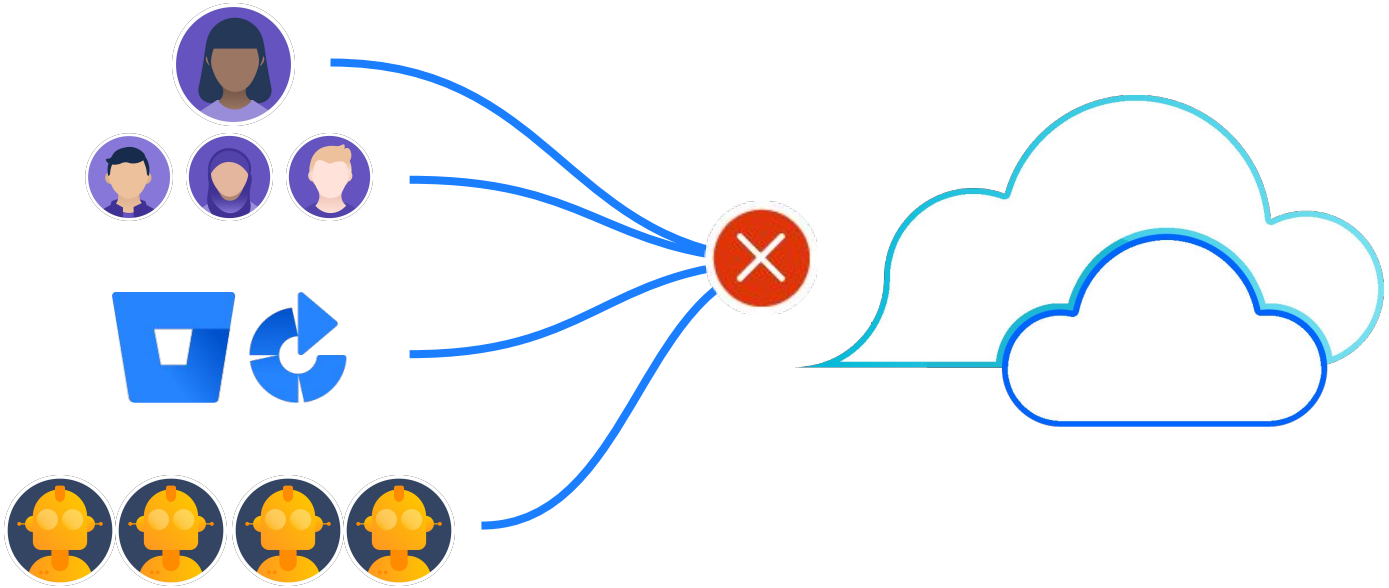
Defining Multi-Tenant Access Controls for a Cluster

Anund McKague | Senior Developer | Anund on Slack

Multi-Tenant

Team has access to a piece of a cluster isolated from other teams

Authn/Authz use cases



Some background on our clusters

Isolated locations

PaaS customers choose where to place their resources

Few things are restricted

Customers can create nearly anything within a namespace

Kube API only

All interactions go through Kube API

Existing PaaS

Users access their service,
and its resources
in multiple availability zones
across multiple regions
in dev, staging, and prod

Kubernetes based v2

Users access their namespace,
and its objects
provisioning the same resources
in separately available clusters
in dev, staging, and prod

So, how do we

Let users in

and their team

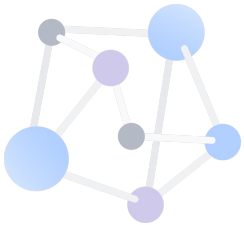
and their CI and automation.

Create their own namespace

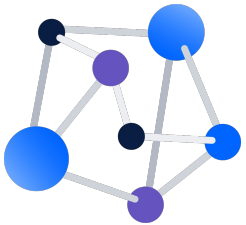
Deny requests based on content

Let users in

Static RBAC



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: paas:ops:view
rules:
- apiGroups:
  - "ops-gateway.voyager.atl-paas.net"
  resources:
  - clusterproviders
  verbs: ["get", "list"]
```



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: paas:ops:view
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: paas:ops:view
subjects:
- kind: User
  name: some_user
```

Their team

RBAC + groups

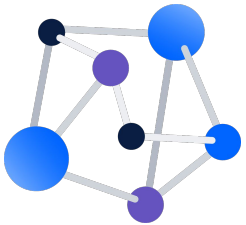
RBAC for teams

Groups

RBAC with groups as the subject

Built in groups

Better to avoid reusing built in groups for internal teams



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: paas:ops:view
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: paas:ops:view
subjects:
- kind: Group
  name: some_team
```

Their CI and automation

Service Accounts, and Authentication webhooks

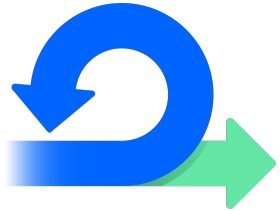
CI and automation

Service Accounts

Built in, username + password for a pod

Existing Services

Should be able to reuse existing service to service authn



Webhook

Webhook Token Authentication can cover both

Create their own namespace

Custom Resources

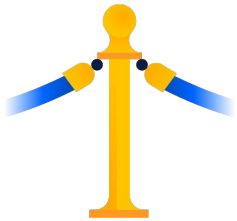
Custom resource permissions

Namespace creator

Control creation of namespace via a custom resource

Custom RBAC verb

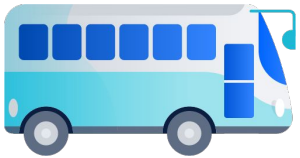
RBAC resourceName rules are limited to certain verbs



Business logic

Custom Resource controller for namespaces can host access logic

API server bridge

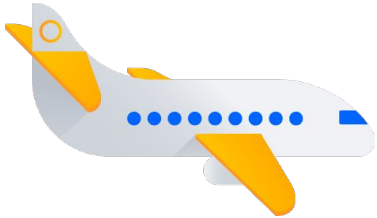


Existing systems

Allow existing services to run as if they are in all clusters

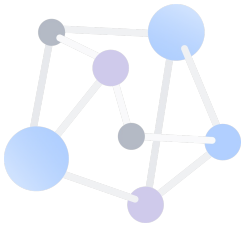
Reacting via watch

Implementing watch lets clusters react to objects from existing services

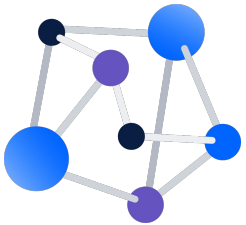


k8s/api-server

Brings in automated authz and auditing



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: paas:generated:some_service
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs: ["get", "list", "claim"]
  resourceNames: some_service
```



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: paas:generated:some_service
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: paas:generated:some_service
subjects:
- kind: Group
  name: some_team
```

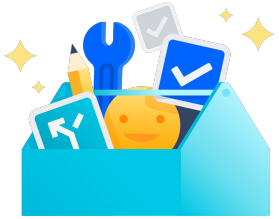
Deny requests based on content

Validating webhooks

Validating Webhook Permissions

Add authz context

Combining custom claims and request contents in webhooks



Avoid user authz

system:serviceaccount:kube-system:generic-garbage-collector also wants in

Webhooks are good for

Verifying the contents of object fields

Performing custom claims

Returning better error messages when something is wrong

Webhooks are bad for

User or group permission whitelists

Allowing silent mutation of user fields

Running expensive or long running checks

Groups and authn

github.com/atlassian/kubetoken
Is an example of stitching LDAP, 2fa, and group centric permissions together

Authn

Other mechanisms

Certs

X509 client certs the API Server validates and extracts username + group information

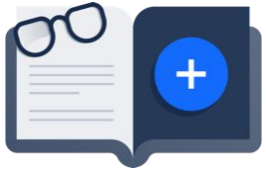
OpenID Connect Tokens

Supported by Azure, Salesforce, and Google, usernames + groups information

Authenticating proxy

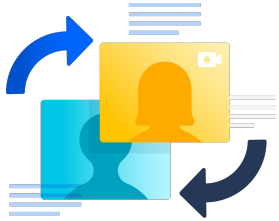
Trusted proxy in front of the Kubernetes API using headers, usernames + groups + extras

Closing



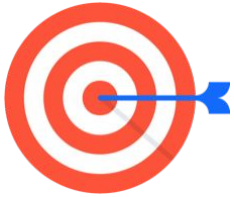
Authentication

Customizable authn via webhook



Authorization

RBAC with a few fancy steps



API Server

Bridge existing systems into Kube

Validation

Everything else RBAC can’t cover

Thank you!

Anund McKague | Senior Developer | Anund on Slack