KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

SIG Auth

Mike Danese, Google
Tim Allclair, Google
Mo Khan, VMware

Intro

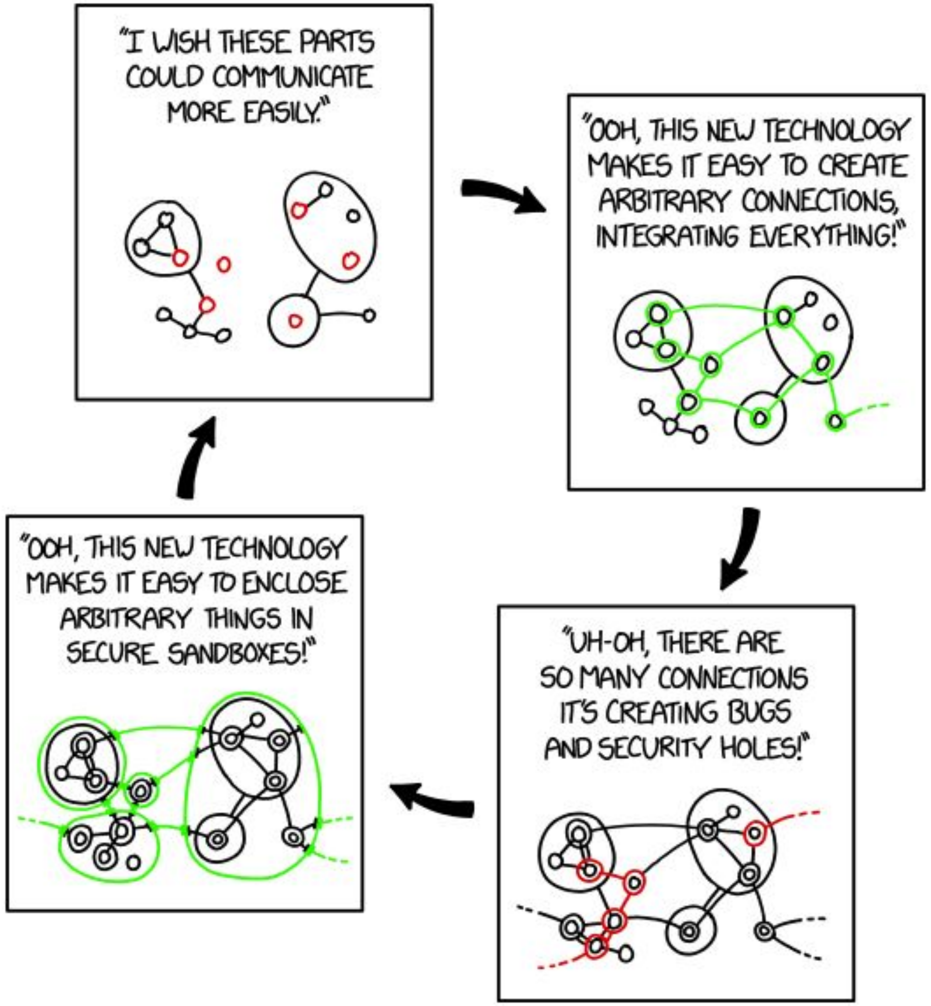
Who are we?

Chairs:
Tim Allclair (@tallclair), Mike Danese (@mikedanese), Mo Khan (@enj)

Subproject approvers:
@deads2k, @immutableT, @liggitt, @mikedanese, @smarterclayton, @sttts, @tallclair

Subproject reviewers:
@awly, @caesarxuchao, @CaoShuFeng, @david-mcmahon, @dims, @enj, @erictune, @errordeveloper, @hongchaodeng, @hzxuzhonghu, @jianhuiz, @krmayankk, @krousey, @lavalamp, @mbohlool, @mml, @ncdc, @nikhiljindal, @pweil-, @sakshamsharma, @sttts, @thockin, @timothysc, @wojtek-t, ...

What do we do?


"I WISH THESE PARTS COULD COMMUNICATE MORE EASILY."
"OOH, THIS NEW TECHNOLOGY MAKES IT EASY TO CREATE ARBITRARY CONNECTIONS, INTEGRATING EVERYTHING!"
"UH-OH, THERE ARE SO MANY CONNECTIONS IT'S CREATING BUGS AND SECURITY HOLES!"
"OOH, THIS NEW TECHNOLOGY MAKES IT EASY TO ENCLOSE ARBITRARY THINGS IN SECURE SANDBOXES!"

What do we do?

SIG Auth is responsible for features in Kubernetes that control and **protect access** to the API and other core components. This includes **authentication** and **authorization**, but also encompasses features like **auditing** and some **security policy**.

https://github.com/kubernetes/community/blob/master/sig-auth/charter.md

Sub-Projects

- Audit Logging
- Authenticators
- Authorizers
- Certificates
- Encryption At Rest
- Multi Tenancy
- Node Identity and Isolation
- Policy Management
- Service Accounts

Theme of 2020

Clean up clean up
everybody everywhere.

Clean up clean up
everybody do your share.

Theme of 2020

- GA even more things?
  - Certificate rotation
  - Bound service account token volumes
  - Allow Insecure Backend Proxy
  - Validating redirects

- Decide on the roadmap for:
  - PodSecurityPolicy (up later)
  - Dynamic auditing

Theme of 2020

- Deprecate things
  - ABAC
  - PodTolerationRestriction
  - PodNodeSelector
  - Other admission plugins? (SecurityContextDeny)
  - Streaming Proxy Redirects

- Finish Deprecating things:
  - Admission Controllers
    - AlwaysDeny
    - DenyExecOnPrivileged
    - DenyEscalatingExec

2019 Highlights

- Retroactive KEPs
  - Certifcates API k/enhancements/1097
  - External credential provider k/enhancements/1137
  - Bound service account tokens k/enhancements/1205
- Dynamic cert reloading
- Force kubelet and aggregated API servers delegated authz to use v1 APIs, allow webhooks to opt-in
- Performance improvements to token cache
- Node restriction improvements
- GA admission webhooks

How to get involved

New Contributors

- good first issue label
- Have a cool idea? Awesome! Prototype it through a plugin.
- Authorization & Authentication webhooks, Dynamic Admission, Dynamic Audit
- Expand test coverage & improve documentation

Experienced Contributors

- help wanted labels
- Help with PR reviews! (even if you're not a "sig auth reviewer")
- Help with issue triage, identify "good first issue" and "help wanted"

Where can you find us?

Slack channel: #sig-auth

Home page: https://github.com/kubernetes/community/tree/master/sig-auth

Mailing list: https://groups.google.com/forum/#!forum/kubernetes-sig-auth

Bi-weekly meetings Wednesday at 11PT (agenda/recordings links on home page)

Questions so far?

Deep Dive

Pod Security Policy

What is it?

- Built-in policy API
- Fine-grained permissions on pod security settings

Examples:

Can a pod be created with a privileged container?

Can I create a pod that mounts a sensitive host path?

What problems does it solve?

- Create {Pod, ReplicaSet, Deployment, ...}
  should not equal **root** on cluster

- Allow cluster administrators to encourage best-practices by configuring more secure defaults

- Decouple low-level linux security decisions from deployment

Current Status

- Beta since early Kubernetes, *beta-quality* since 1.8
- Opt-in

Super confusing, opt-in, bug prone

Problem 1: Flawed authorization model

I can create a pod if I have the USE permission on a PSP that allows that pod OR the *pod's ServiceAccount* has the USE permission on the allowing PSP

- Granting permission to the user is intuitive, but breaks controllers
- Dual model weakens security
  - Cannot have a privileged controller create pods on behalf of a user, enforced through PSP
- PSP use can be scoped to a namespace but privileged pods can break out of that isolation
  - PSP protects the node

Problem 2: Difficult to roll out

PSP **fails closed** in the absence of policy – with no PSPs, all pods are denied

- Cannot enable by default - *and can never be part of conformance*
- Users need to add PSPs for all workloads *before* enabling the feature
  - No audit mode
- Opt-in leads to insufficient test coverage, and frequent breakage due to cross-feature incompatibility
- No bootstrap PSP policy exists
  - Unlike RBAC, there is no strong culture of including PSP manifests with projects

Problem 3: Inconsistent unbounded API

- API has grown organically with lots of inconsistencies
- Many requests for niche use cases
  e.g. labels, scheduling, fine-grained volume controls, etc.
- Poor composability
  - Weak prioritization model
- Mutation priority can be unexpected

Effective usage still requires an understanding of linux security primitives.
e.g. MustRunAsNonRoot + AllowPrivilegeEscalation

How you can help

- Provide feedback on how you have or have not successfully used PSPs
- What PSP policies did you create?
- What features do you wish that PSP had?

Hypothesis

> 90% of users care about 2-3 policies

1. "Privileged" - I can do anything
2. "Restricted" - a.k.a. best practices
3. "Default" - I can run a minimally specified pod

```
apiVersion: v1
kind: Pod
metadata:
  name: default
spec:
  containers:
    - name: my-container
      image: my-image
```

Complications

- **Windows** - do those same 3 buckets apply?
- **Sandboxes** - privileged in sandbox != privileged on host
- **Managed addons** - cannot always be modified

Future?

- Fix PodSecurityPolicy (v2beta1?)
  Bind to namespaces, allow by default, migration path, audit mode

- New core (in-tree) minimalist policy mechanism
  - Distill PSP to the essentials, for everything else there are plugins
  - Privileged, Default, Restricted

- No in-tree policy mechanism, leverage webhook ecosystem
  - Love PSP? it can live on in a webhook model!
    - Convert OpenShift's Security Context Constraints into a webhook and migrate to that API over time
  - Work towards standardizing around a policy framework, OPA?

Does Kubernetes need a built-in mechanism for pod policy?

Open Policy Agent Gatekeeper

What is it?

This is **rego** for expressing that a container must run as a user:

```
violation[{"msg": msg}] {
 rule := input.parameters.runAsUser.rule
 input_containers[input_container]
 provided_user := run_as_user(input_container.securityContext, input.review)
 not accept_users(rule, provided_user)
 msg := sprintf("Container %v is attempting to run as disallowed user %v", [input_container.name, provided_user])
}
violation[{"msg": msg}] {
 rule := input.parameters.runAsUser.rule
 input_containers[input_container]
 not run_as_user(input_container.securityContext, input.review)
 rule != "RunAsAny"
 msg := sprintf("Container %v is attempting to run without a required securityContext/runAsUser", [input_container.name])
}
accept_users("RunAsAny", provided_user) {true}
accept_users("MustRunAsNonRoot", provided_user) = res {res := provided_user != 0}
accept_users("MustRunAs", provided_user) = res  {
 ranges := input.parameters.runAsUser.ranges
 matching := {1 | provided_user >= ranges[j].min; provided_user <= ranges[j].max}
 res := count(matching) > 0
}
input_containers[c] {
 c := input.review.object.spec.containers[_]
}
input_containers[c] {
 c := input.review.object.spec.initContainers[_]
}
run_as_user(container_security_context, review) = run_as_user {
 run_as_user := container_security_context.runAsUser
}
run_as_user(container_security_context, review) = run_as_user {
 not container_security_context.runAsUser
 review.kind.kind == "Pod"
 run_as_user := review.object.spec.securityContext.runAsUser
}
```

What is it?

Gatekeeper templatizes this as a **constraint** (a CRD)

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPAllowedUsers
metadata:
  name: psp-pods-allowed-user-ranges
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    runAsUser:
      rule: MustRunAs
      ranges:
        - min: 100
          max: 200
```

Apply this constraint to pods

Define the constraint parameters

Why not custom admission?

Cut out the boilerplate

- Simplify the install / deployment process
- Simplify implementation (if you know rego)

Policy outside the cluster

- Dryrun, pre-commit, CI/CD
- Data-plane policy

Gatekeeper: Core Features

- Validating admission control
  - Control what end-users can do on the cluster
- Context-aware/referential policies
- Constraints are parameterized and easily configurable by admins
- ConstraintTemplates provide the source code for constraints
  - Easily shared
  - Testable
  - Developed internally or sourced from the community
- Audit
  - Periodically evaluates resources against constraints
  - Allows for ongoing monitoring of cluster state to aid in detection and remediation of pre-existing misconfigurations

Gatekeeper: Latest Updates

- Dry run
  - Test canary releases in a cluster in stages without impacting the cluster and your users
  - Gain confidence for our policies for admins before enforcing them; gradual rollout
- Namespace Selector
  - Narrow the scope of resources a constraint can enforce to certain namespaces only
- Policy library
  - Community developed policies
  - Alternative to Pod security policies
- Multi-source constraint template
- Metrics

Gatekeeper: Potential Growth

- Production ready
- Mutation
- External Data
- More audit features
- More metrics
- More policies
- Developer tooling
- Authorization? (likely separate project, same general semantics)

Bound Service Account Tokens

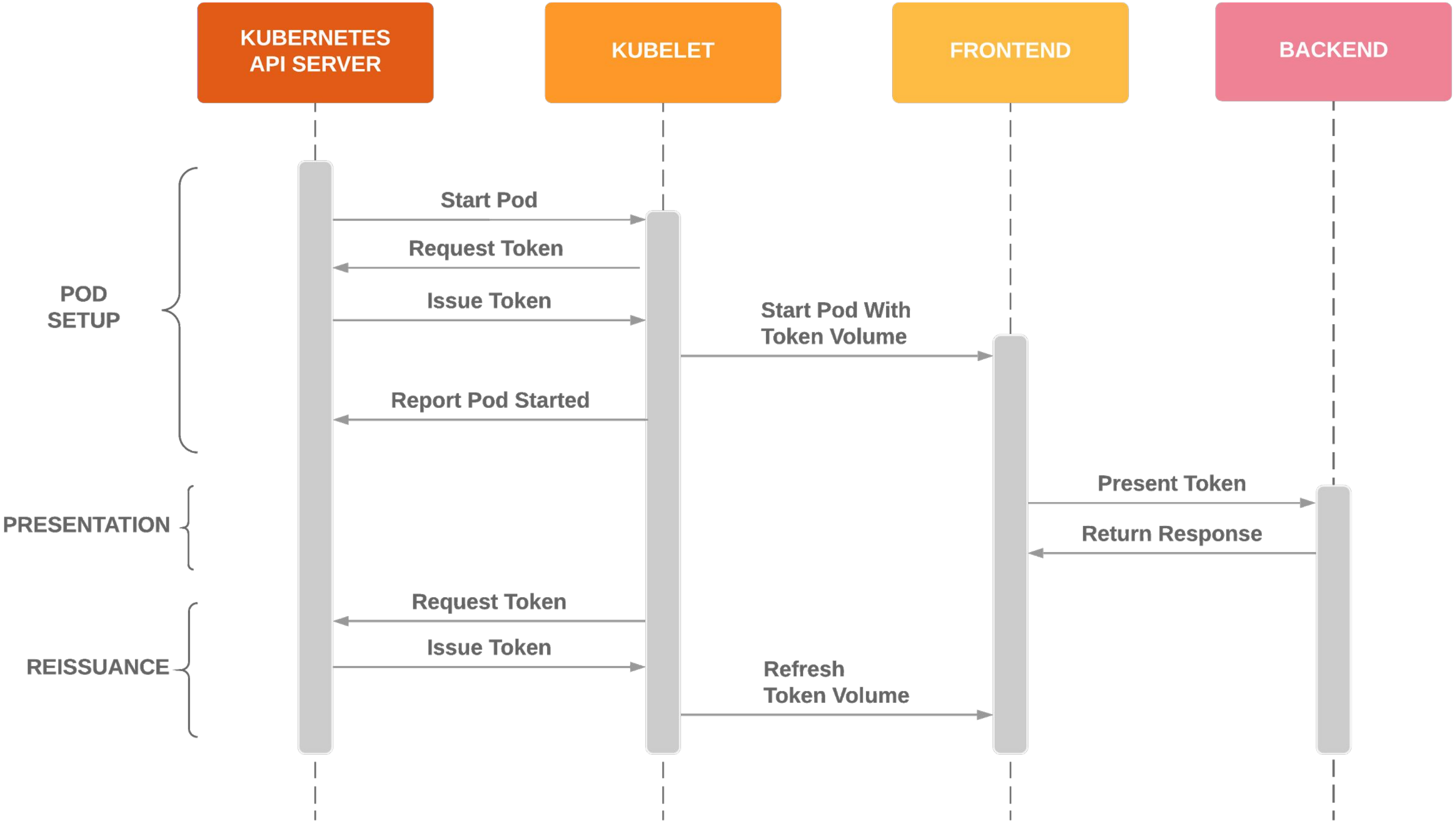
Legacy Service Account Tokens

- Requires a secret stored in etcd
  - Security risk via exfiltration
  - Performance concern in large clusters
- No expiration time
  - Encourages practice of never reloading the token
  - Revocation requires lookups (these are cached now)
- No audience binding
  - Using token against anything other than kube API server is unsafe
  - Cannot safely use these tokens to assert identity to external systems

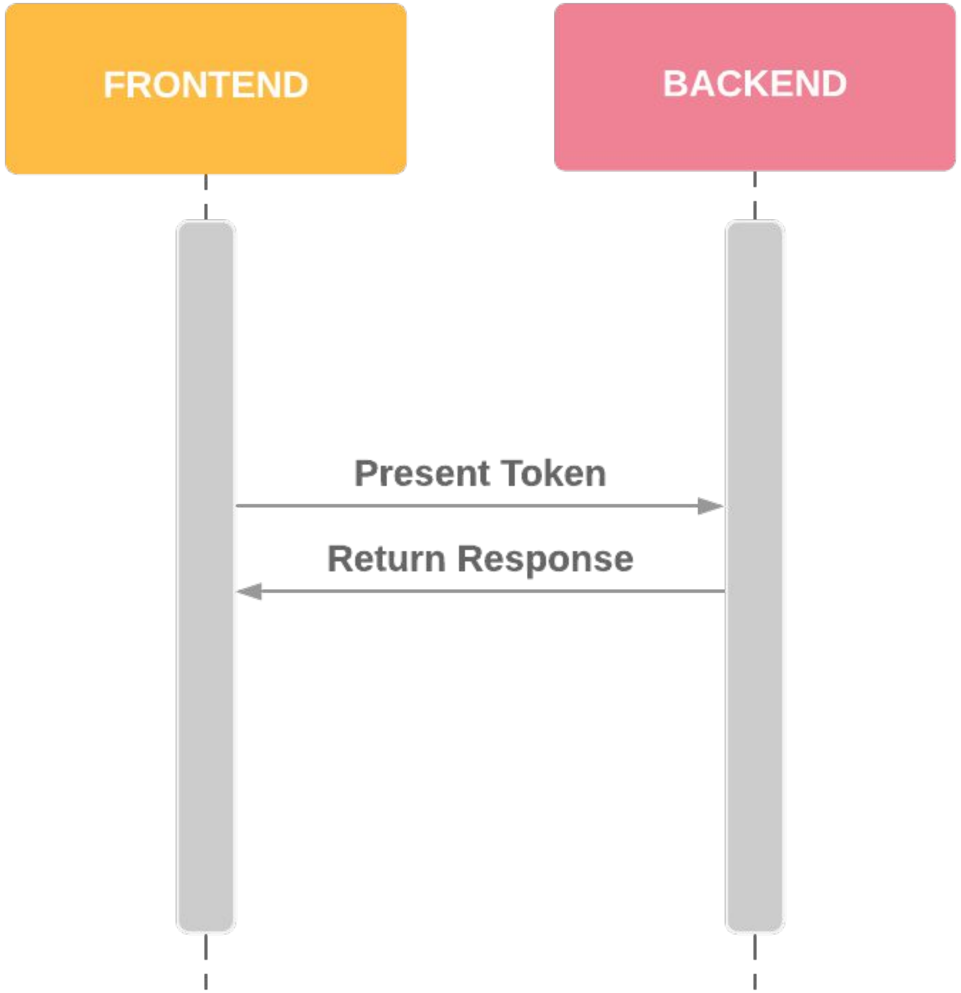
New Service Account Tokens

- Exposed to pods via a kubelet managed tmpfs
- Flexible verification
- Revocable via API
- Limited TTL with automatic rotation
- Support audience scoping
- Never stored in etcd
- Tighter file permissions

Token Issuance



Token Issuance



Incompatibilities

- API servers need a new flag!
- Client libraries need to change to reload tokens!
- PodSecurityPolicies that allowed secret volumes but not projected volumes will no longer be usable with newly created pods that auto-mount service account volumes.
- Pre-1.11 Kubelets (assuming they also enable alpha features) will no longer run new pods that mount service account volumes.

Why not tokens?

Tokens have a major downside

- Forwardable so may be replayed
- Don’t solve server authentication

Please rate the session

https://kccncna19.sched.com/event/Uakn