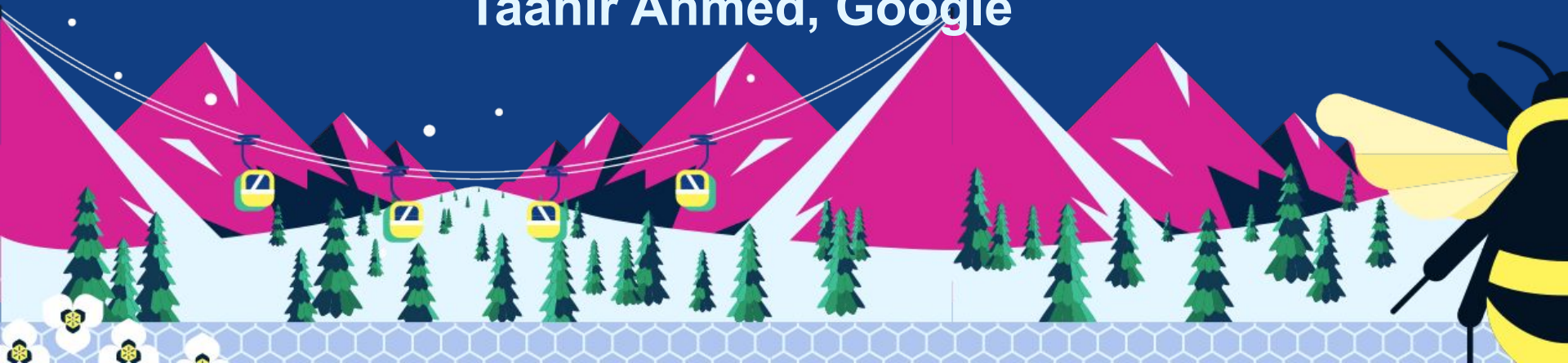
KubeCon CloudNativeCon North America 2026

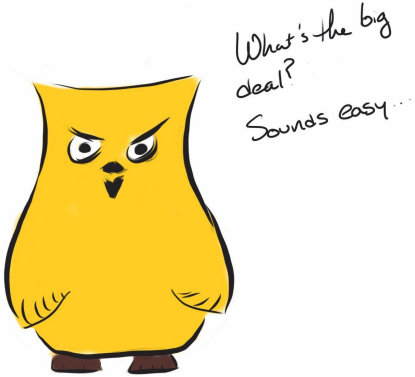
PKI Done Easy with Pod Certificates and Cluster Trust Bundles

Standa Láznička, Microsoft
Taahir Ahmed, Google



Principles of well-defined PKI

- Test-proven automated certificate rotation from signer to leaf
- Minimal, separate trust domains
  - Limit blast radius, simplify rotation
- Simple, automated trust distribution
- Automated cert-issuance driven by well-defined policies



PKI in Kubernetes - the old way

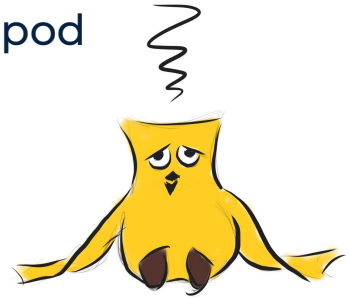
Trust distribution

- One **root ConfigMap**
- Synchronized **into every namespace**
- 😕 Needs write/CM everywhere, unnecessary duplicate objects in etcd, distribution is not atomic

PKI in Kubernetes - the old way

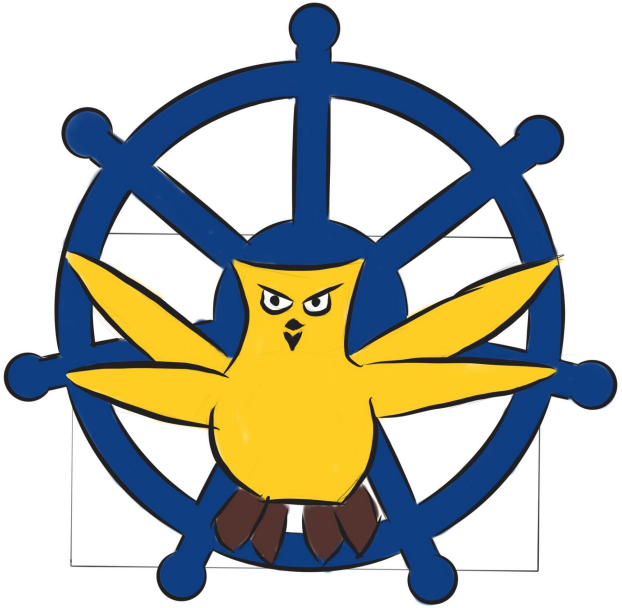
Certificate issuance for pods

- **Store private key** somewhere?
- **Create** a CertificateSigningRequest **object** containing CSR for the key
- Signer waits for the CertificateSigningRequest to **be approved**
- Signer **signs** the CertificateSigningRequest
- **Paste the certificate** to a pod-mountable location
- **Create the pod**, mounting the key and certificate
- 😕 Too many moving pieces, private key stored outside the pod



PKI in Kubernetes - the new way 🎉

- Kubernetes 1.37 - stable
- **Pod Certificates** handle secure provisioning of private keys and certificates
- **Cluster Trust Bundles** simplify distribution of root certificates



Pod Certificates

- Automates secure certificate issuance and rotation
- Kubelet:
  - Generates a private key
  - Creates a *PodCertificateRequest*
  - Waits for the signer implementation to issue the certificate
  - Automatically rotates the certificate before it expires.
- Accessed using *podCertificate* projected volume sources

Pod Certificates - Security

- Private key is generated by Kubelet, and never leaves the node
- Stored in the pod's filesystem, in a tmpfs
- The API server automatically enforces node restriction
- Short-lived certificates are enforced
  - kubernetes.io signers are constrained to 24 hour lifetimes
  - All other signers are constrained to 91 day lifetimes

Cluster Trust Bundles

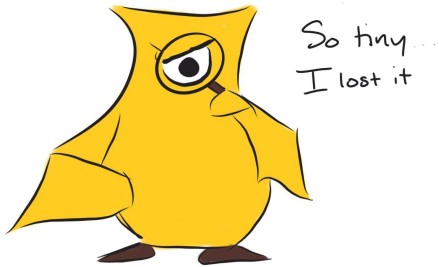
- Like *ConfigMaps*, but specialized for holding certificates
- Cluster-scoped
- Can be associated with a particular signer
- Or just freestanding
- *clusterTrustBundle* projected volume sources put them into the filesystem

How to use?

- Currently you need to bring your own signer implementation
- github.com/ahmedtd/tinycert is a minimal signer
- Multiple signer implementations can coexist in the same cluster
- More on this in "Future Roadmap"

TinyCert Introduction

- TinyCert provides two different signers to try out:
  - ahmedtd.github.io/tinycert-service — Certificates with DNS SANs based on the Kubernetes Services that your pod is a part of.
  - ahmedtd.github.io/tinycert-spiffe — SPIFFE certificates that identify your pod by its Kubernetes Service Account or Pod name
- Go libraries for loading and automatically refreshing credentials from a SPIFFE Filesystem Delivery folder.
- Deployable examples:
  - Server with DNS cert, client with no cert
  - Server with DNS cert, client with SPIFFE cert
  - Server with SPIFFE cert, client with SPIFFE cert



TinyCert Volume


```
volumes:
- name: tinycert-spiffe-credential-folder
  projected:
    sources:
    - podCertificate:
        signerName: ahmedtd.github.io/tinycert-spiffe
        keyType: ECDSAP256
        credentialBundlePath: credential-bundle.private-key.pem
    - clusterTrustBundle:
        signerName: ahmedtd.github.io/tinycert-spiffe
        labelSelector:
          matchLabels:
            "ahmedtd.github.io/tinycert-canarying": "live"
            "ahmedtd.github.io/tinycert-workload-trust-domain": "cluster.local"
            "ahmedtd.github.io/tinycert-peer-trust-domain": "cluster.local"
        path: cluster.local.spiffe-trust-bundle.pem
```

Ask for a SPIFFE certificate

TinyCert Volume


```
volumes:
- name: tinycert-spiffe-credential-folder
  projected:
    sources:
    - podCertificate:
        signerName: ahmedtd.github.io/tinycert-spiffe
        keyType: ECDSAP256
        credentialBundlePath: credential-bundle.private-key.pem
    - clusterTrustBundle:
        signerName: ahmedtd.github.io/tinycert-spiffe
        labelSelector:
          matchLabels:
            "ahmedtd.github.io/tinycert-canarying": "live"
            "ahmedtd.github.io/tinycert-workload-trust-domain": "cluster.local"
            "ahmedtd.github.io/tinycert-peer-trust-domain": "cluster.local"
        path: cluster.local.spiffe-trust-bundle.pem
```

Ask for the signer's root certificates

TinyCert Volume


```
volumes:
- name: tinycert-spiffe-credential-folder
  projected:
    sources:
    - podCertificate:
        signerName: ahmedtd.github.io/tinycert-spiffe
        keyType: ECDSAP256
        credentialBundlePath: credential-bundle.private-key.pem
    - clusterTrustBundle:
        signerName: ahmedtd.github.io/tinycert-spiffe
        labelSelector:
          matchLabels:
            "ahmedtd.github.io/tinycert-canarying": "live"
            "ahmedtd.github.io/tinycert-workload-trust-domain": "cluster.local"
            "ahmedtd.github.io/tinycert-peer-trust-domain": "cluster.local"
        path: cluster.local.spiffe-trust-bundle.pem
```



Place the leaf cert and root certificates in a SPIFFE Filesystem Delivery folder layout

TinyCert Volume Mount


```
volumeMounts:
  - name: tinycert-spiffe-credential-folder
    mountPath: /run/tinycert-spiffe
```

Place the SPIFFE Filesystem Delivery folder at a path where your workload will find it

Live Demo

Current Real-World Usage

- agent-substrate/substrate
  - Agent Substrate ships a Service DNS and SPIFFE certificate signer
  - Used for authentication between system components of the Substrate install — ate-api-server, atelet, and the egress gateway.
  - Nothing specific to Substrate — these are a polyfill for functionality that we eventually want K8s to natively provide with built-in signers.

SPIFFE Filesystem Delivery

- SPIFFE Filesystem Delivery - A new (incubating) standard in the SPIFFE project
- How does a "SPIFFE-compatible" workload find its private key and certificate
- Previous answer was the SPIFFE Workload API
- SPIFFE Filesystem Delivery provides a solution that only relies on built-in K8s capabilities
  - One folder
  - credential-bundle.private-key.x509.pem
    - Private key and certificate chain
  - Multiple <trust-domain>.spiffe-trust-bundle.x509.pem files
    - Root certificates for each federated trust domain

Future Roadmap - Built-in Signers

- kubernetes.io/service-serving
  - Defined in KEP-6283
  - Issues certificates with DNS SANs for the Services your Pod is part of
  - Easy solution for server TLS within the scope of a single cluster

Future Roadmap - Built-in Signers

- kubernetes.io/pod-identity
  - Issues SPIFFE certificates that identify your pod as a Service Account or Pod
    - `spiffe://${cluster-trust-domain}/service-account/${namespace}/${name}`
    - `spiffe://${cluster-trust-domain}/pod/${namespace}/${name}`
  - An extension will carry other K8s-useful information, similar to Service Account JWTs
  - Should be usable to authenticate to kube-apiserver
  - Should be federatable to third parties, similar to Service Account JWTs

Future Roadmap - Cluster Trust Bundles

- KEP-6283: Leveraging certificate APIs in the platform
  - There are many places where the Kubernetes APIs currently use a ConfigMap full of root certificates
    - APIServices
    - Validating/MutatingWebhookConfiguration
  - Today, automatically-injected kube-api-access volumes use a ConfigMap to plumb the root certificates for verifying kube-apiserver
    - The ConfigMap has to be published per-namespace with a controller
    - Migrating to ClusterTrustBundles will allow the cluster to keep one central object updated

Future Roadmap - Not ACME Signers

Why not build a native integration with ACME signers like Let's Encrypt?

- Restrictive rate limits on requesting certificates for new private keys
- They generally assume:
  - Generate private key once
  - Share it among all TLS-terminating backends or load balancers
- This is at odds with Pod Certificate's "private key never leaves the workload" model.
- We trialed this, and quickly hit rate limits even for normal operation of modestly-sized workloads
- The CA's multi-perspective verification process often takes ~10 minutes, which is painful when it's blocking the startup of each replica pod.
- Some ACME-based CAs that require stricter account registration and payment might have rate limits high enough to work.

Questions



- Questions!