
CLOUDNATIVE
SECURITYCON
NORTH AMERICA 2023

CLOUDNATIVE
SECURITYCON
NORTH AMERICA 2023

Multi-Service Without A Mesh

Evan Anderson

Why This Talk?

Use existing, mature technologies

“The hard way” … building understanding by building a thing

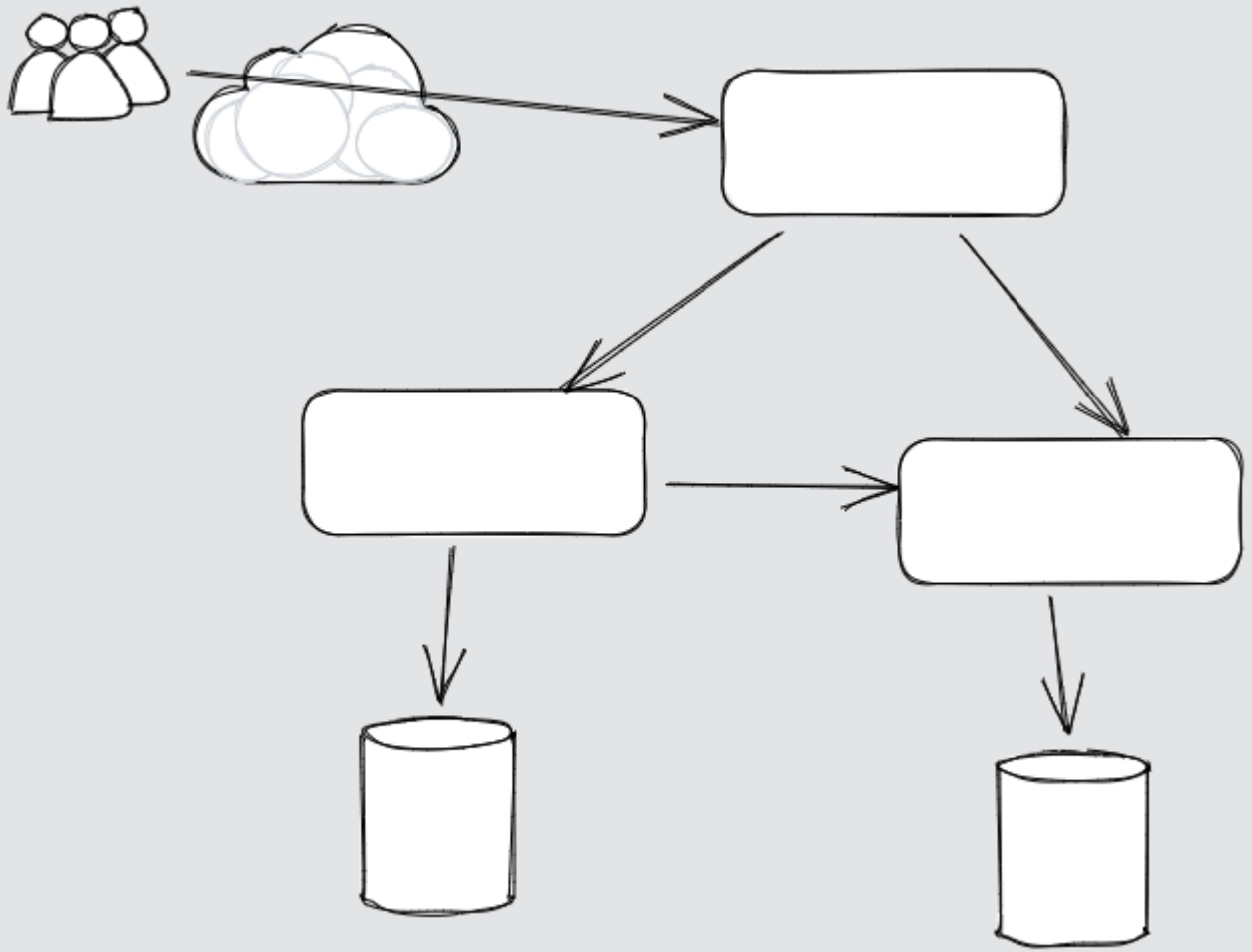
It’s not as easy as it should be

We can make it better!

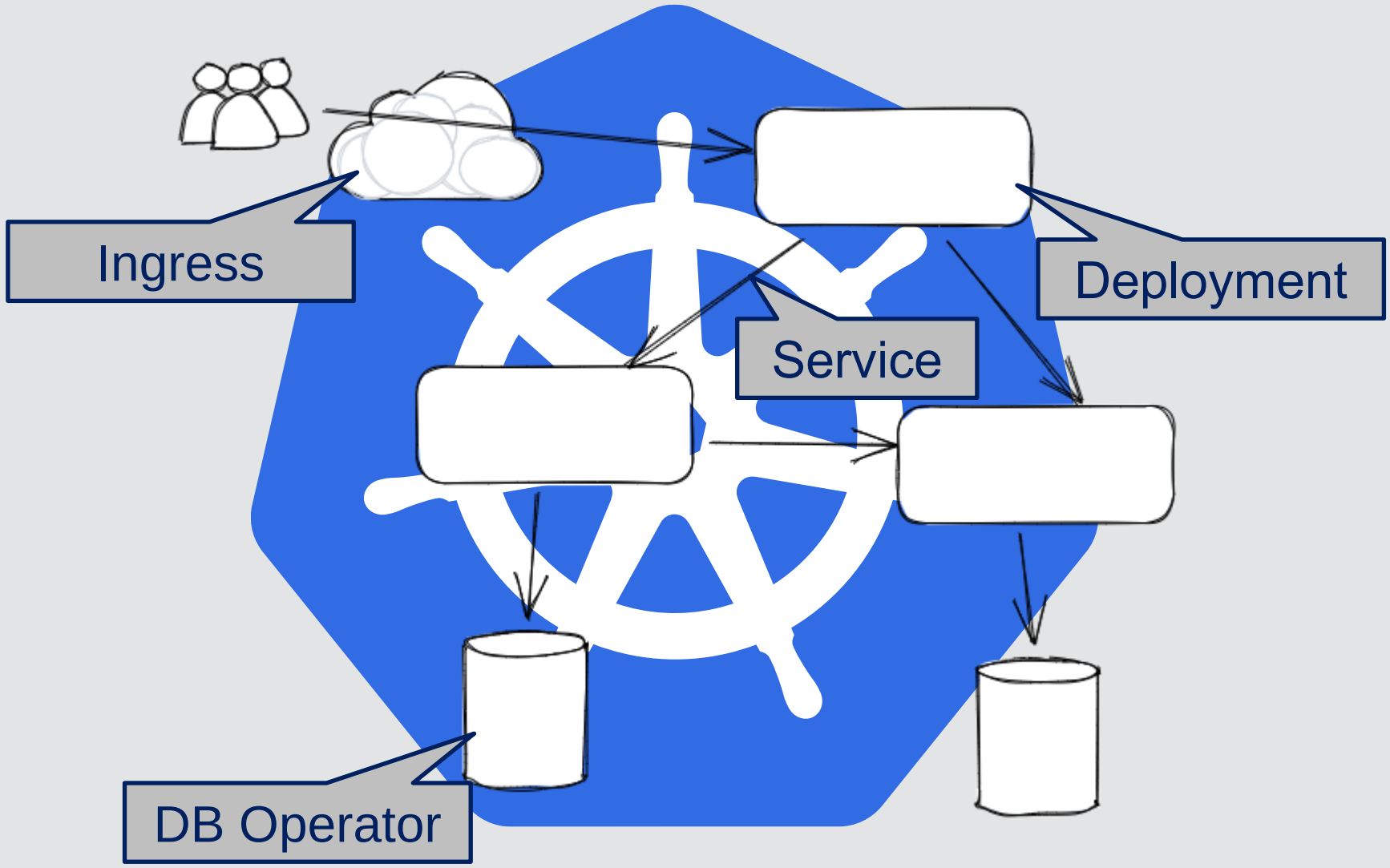
Multi-Service

So, you want a platform for microservices:

- What does that mean?
- Can I just slap some Kubernetes on it?
- I’m not ready for a Service Mesh



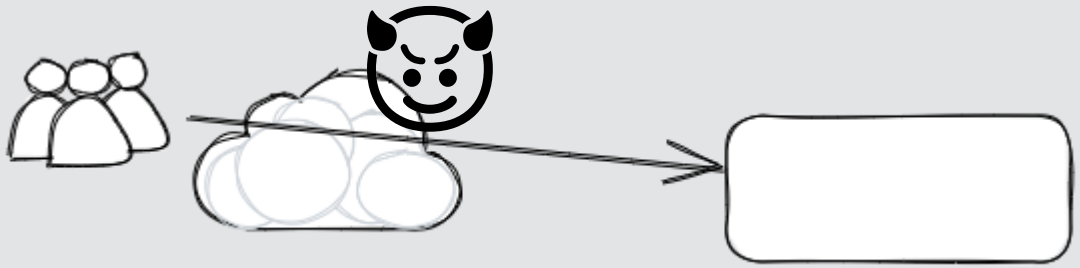
We Can Do This!



What About Security?

Users connect over the internet.

- Probably need TLS?



`cert-manager` to the rescue!

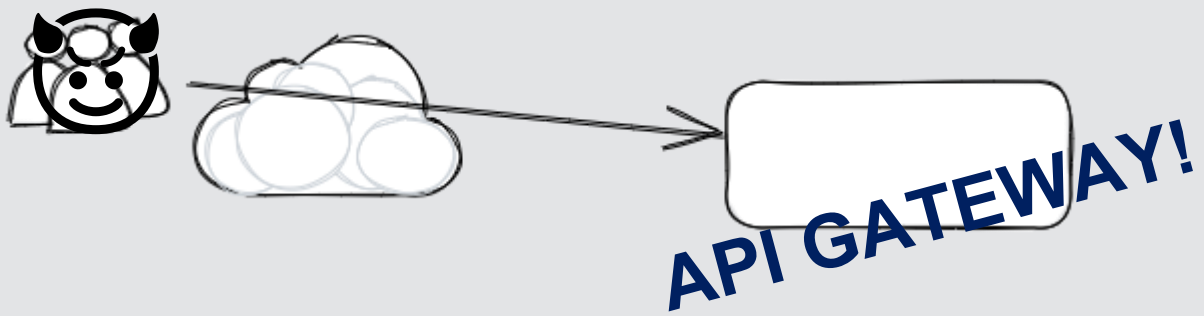
- Annotate each resource (Ingress or Gateway) to provision certs
- If you need other APIs (VirtualService, HTTPProxy, etc) you'll need to wire up the Certificate resources yourself

Encryption != Secure

We've ensured that bad people on the internet™ can't intercept our communications

… but, they can still connect as normal users!

We don't want an Ingress, we want an API Gateway



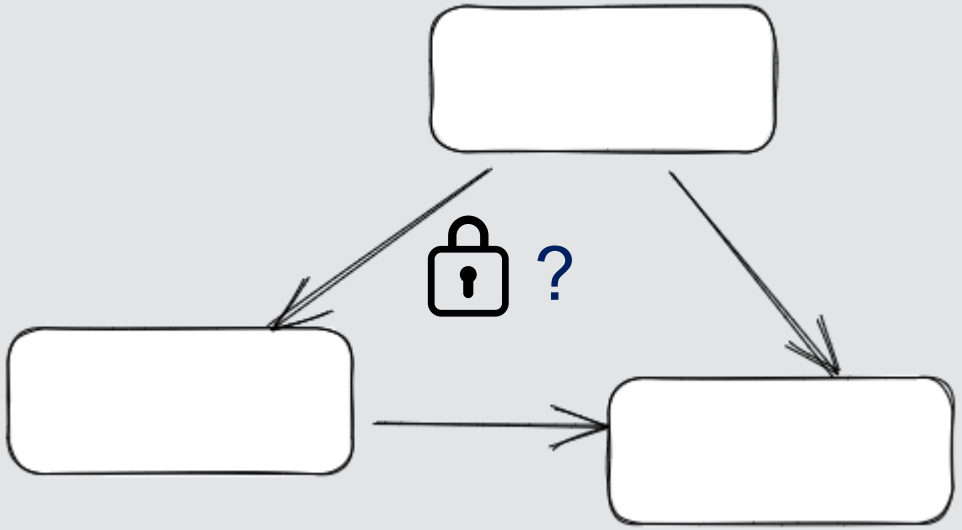
- Authentication
- Rate Limits & DoS protection
- API Keys / Feature Access

What About On-Cluster?

NetworkPolicy allows us to enforce L4 (TCP) firewall rules

- If our CNI implements it…

Some CNI implementations also implement encryption



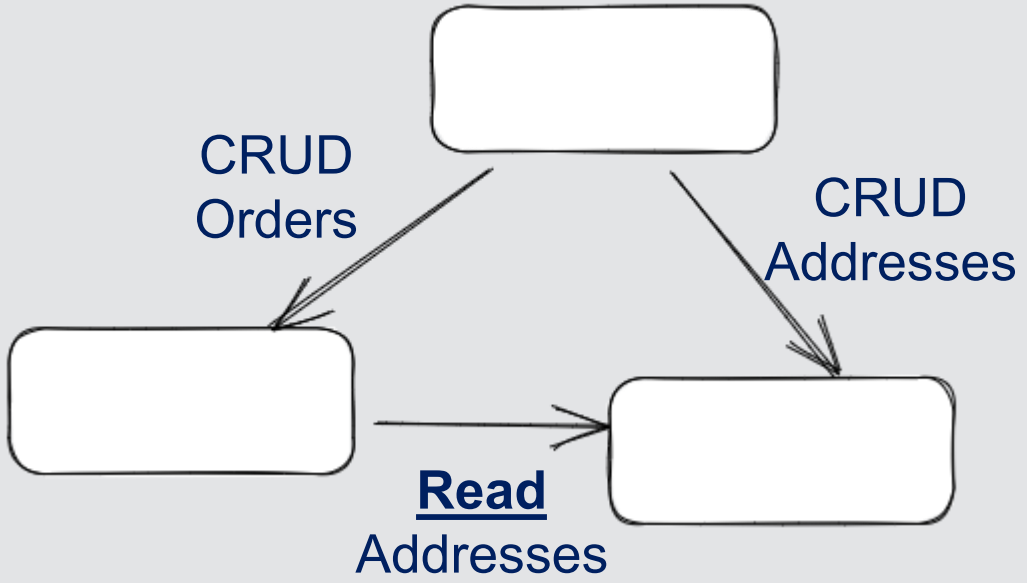
Service Identity

We also want to be able to control which services are exposed to which peers.

Even over REST and gRPC!

We need a way to identify callers…

- in a dynamic network environment



Can we use ServiceAccounts for this?

ServiceAccounts to the Rescue

Kubernetes automatically mounts an OpenID token for the Pod's ServiceAccount into the pod unless you set

```
automountServiceAccountToken: false
```

on the service account.

You can use this token to prove your app's identity.

DON'T DO THIS

That is a *bearer token* – anyone who has it can authenticate to the `apiserver` as the ServiceAccount in question!

Token Projection for Identity

Enter Service Account Projection!

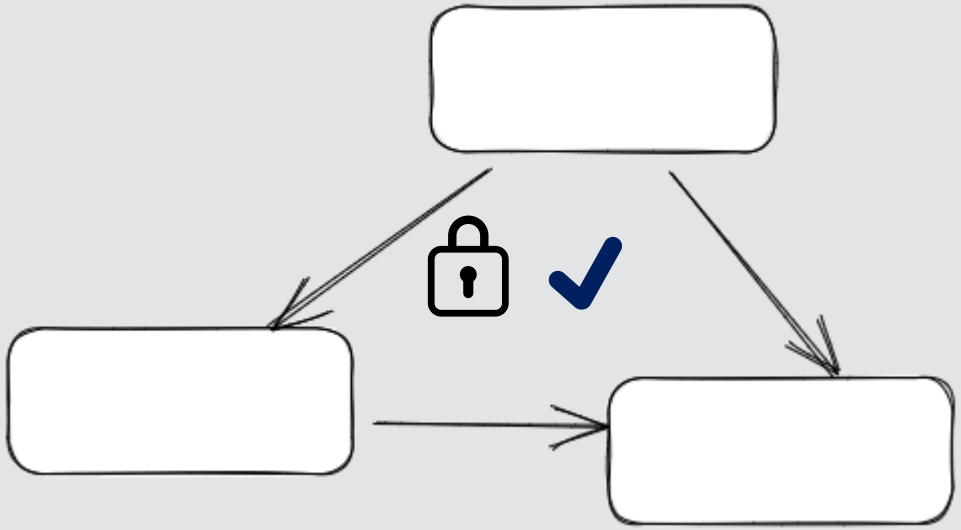
Allows applications to mount an OpenID Connect token corresponding to their Service Account

- Short-lived
- Audience-bound (different tokens for different endpoints)

Can be used outside the cluster or within the cluster.

Bearer Tokens

Awesome, now we've got these Bearer tokens that we're going to send over the wire, and if anyone reads them, they can impersonate another service…



We can define a self-signed CA and use a ClusterIssuer to sign

`foo.bar.svc.cluster.local`

Avoiding Bearer Tokens

Bearer tokens are *forwardable* – if I send you my Bearer token, you can check it, but you can also send it to another system!

An alternative is using TLS client certificates

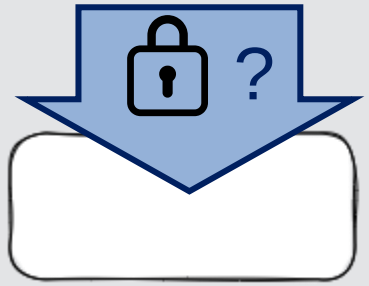
The SPIFFE project defines a way to assign client identies and manage trust

Requires a local agent

Go, Java, and Rust libraries

Converting An Auth Problem To a Key Distribution Problem

Now that we have a cert for each workload on the cluster, we need to use them:



1. Mount the certificate Secret and configure the app
2. Mount the CA cert and trust that CA

Most applications make (1) easy.

(2) requires that you know which SSL library you're using (Java, OpenSSL, golang, etc), OS details, and you may need to run a command (`openssl rehash`)

Loading CA Certificates

- ✓ Golang doesn't need openssl rehash
- ✓ openssl rehash needs write access to SSL_CERT_DIR
  1. Use an init container: build an SSL_CERT_DIR in an emptydir
  2. In the init container: run `openssl rehash $SSL_CERT_DIR`
  3. Set `SSL_CERT_DIR=<emptyDir mount>` in your app
- ✓ Java:
  1. You'll need to use `keytool` to add the cert, or write code

CNB and Service Bindings can add these steps when building

Why are we doing this again?

… What's So Bad About Mesh?

It simplifies a lot of the above, and can help with observability!

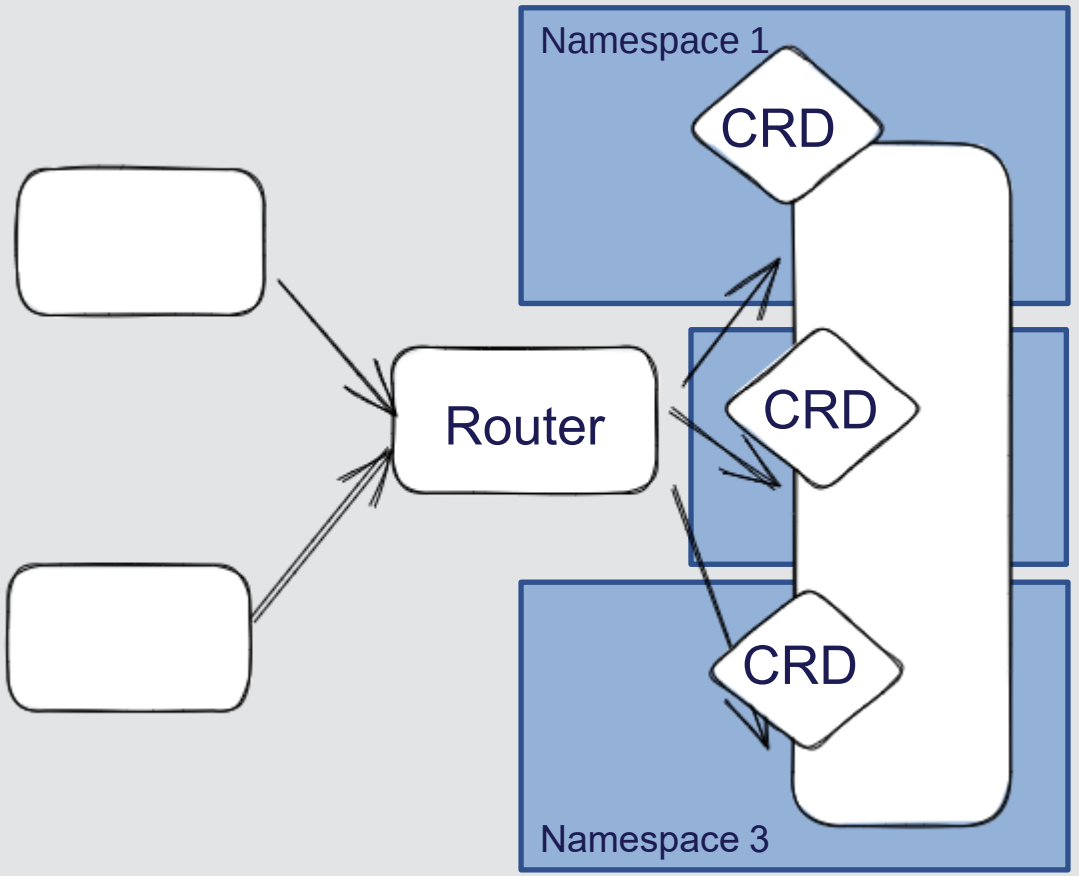
But:

- Rollout can be tough
- Sidecar injection can have complicated side effects
- Increased resource requirements
- High functionality, but high complexity and learning curve
- Some organizations aren't ready for the bleeding edge

Multi-Tenancy

Multi-Tenant services need to appear as different services to different clients

Consider an operator which provides slices of a shared Redis cache: we can share the backing Redis instance overhead across multiple namespaces, so the cost of 100MB of Redis is cheap



Tricks for Multi-Tenant Services

Multi-tenant services defeat a lot of our protections

Non-Mesh	Mesh
✗ NetworkPolicy	✗ NetworkPolicy / Mesh policy
✗ Token Projection	✗ Token Projection
✗ SPIFFE	✗ SPIFFE / Mesh identity
✓ TLS SNI	✓ TLS SNI

Fixing What We Broke

NetworkPolicy

Need some way to distinguish in the TCP 4-tuple:

`(src ip, src port, dst ip, dst port)`

First two don't vary by service; CNIs rewrite the `dst ip` to `pod ip`

Can we use destination port?

➔ Yes, if we create a service per CRD!

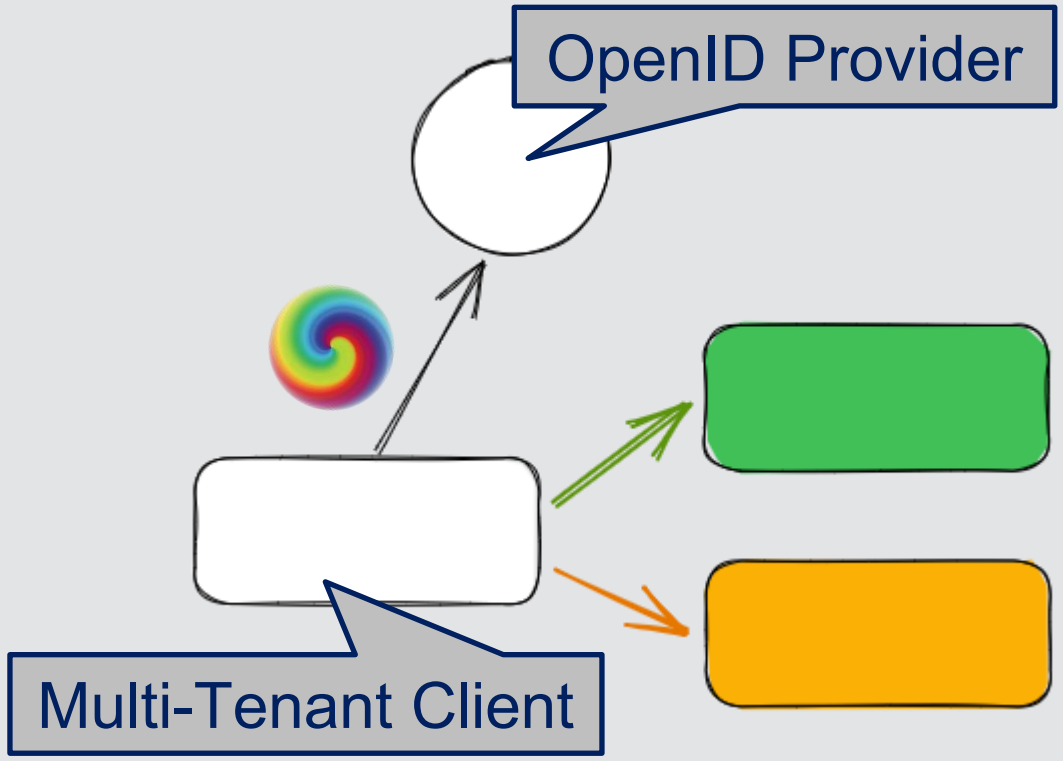
```
kind: Service
metadata:
  name: for-foo
spec:
  type: ClusterIP
  ports:
  - name: redis-foo
    port: 6380
    targetPort: 16844
```

Fixing What We Broke, Part 2

Token Projection

Kubernetes is *an* OpenID provider, but you can have several

If you need to auth as multiple clients, use your own ID provider or delegation



You May Need These *and* a Mesh

A mesh secures service-to-service communication

Which works as long as a service or client has one identity

What about multi-tenant services?

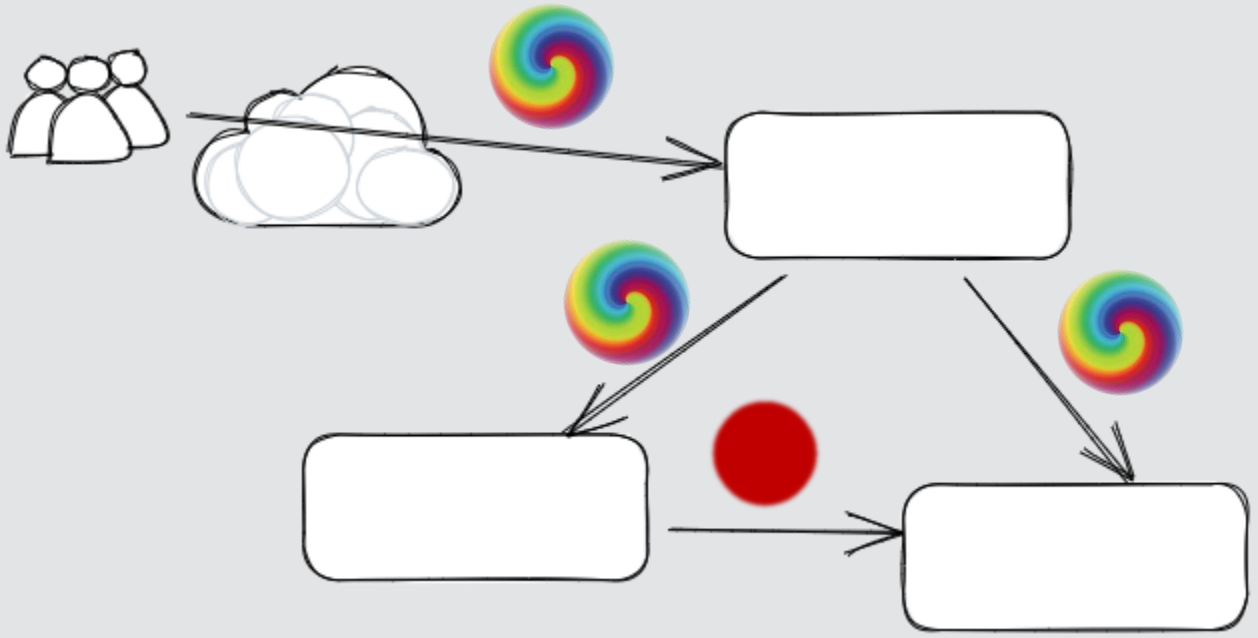
And, what service isn't multi-tenant if you have fine-grained authorization?

Everyone is Special

Most applications are multi-tenant in some way.

If users can log in and have different permissions, you have a multi-tenant system.

You need application involvement in security for a multi-tenant system.



Multi-tenancy and Mesh

A service mesh can enforce APIs and permissions between services

But it's hard to replicate application auth rules

… and you don't want to!

```
spec:
  selector:
  matchLabels:
    app: details
  action: ALLOW
  rules:
    - from:
      - source:
          requestPrincipals:
          - userA@corp.com/userA@corp.com
      to:
      - operation:
          methods: ["GET"]
          paths:
          - "/account/me"
          - "/orders/userA"
          - "/address/userA"
          - "/payments/userA"
          - "/recipts/userA"
    - from:
      - source:
          requestPrincipals:
          - userB@corp.com/userB@corp.com
```

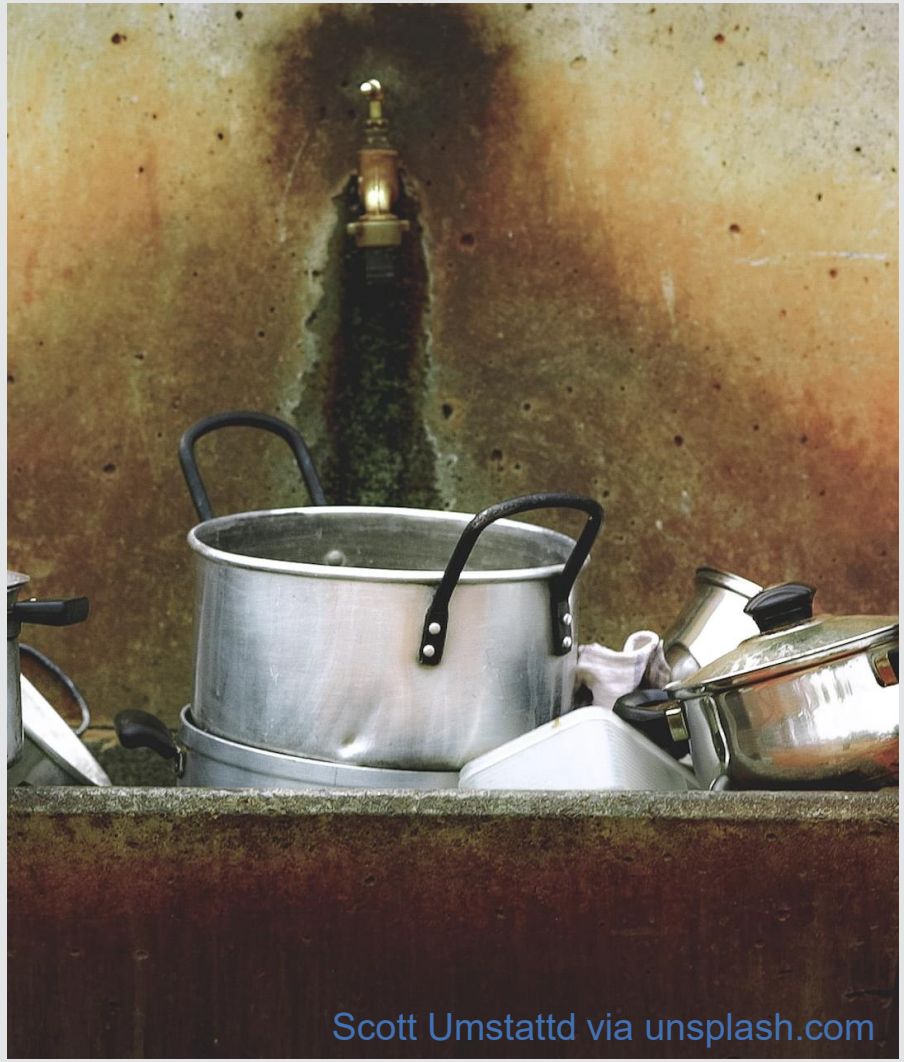
API Gateway Mesh?

API Gateways have a lot of rich capabilities.

- Payload transformations
- Caching
- Dynamic policy control

You could build all those into your mesh

… maybe?



Layered Payloads

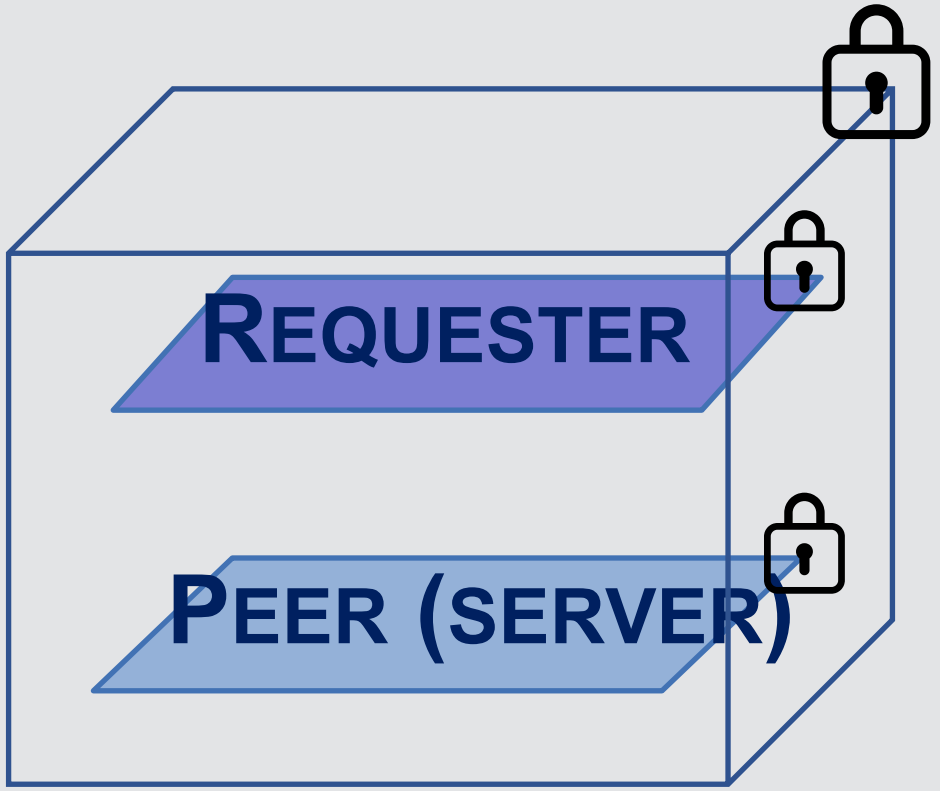
Layers of authorization:

NetworkPolicy or Mesh:

- API Gateway allowed to talk to me
- Request came from API Gateway

OpenID & Application

- User has access to resource



Authorization decision

Everything Is Awesome

On-cluster TLS is too hard!

Authentication (identity) is hard!

Authorization is hard!

Meshes are good for some things, but aren’t a security panacea

Working around these may bring up new scalability challenges…

Thank You!

CLOUDNATIVE SECURITYCON NORTH AMERICA 2023

Please scan the QR Code above to leave feedback on this session