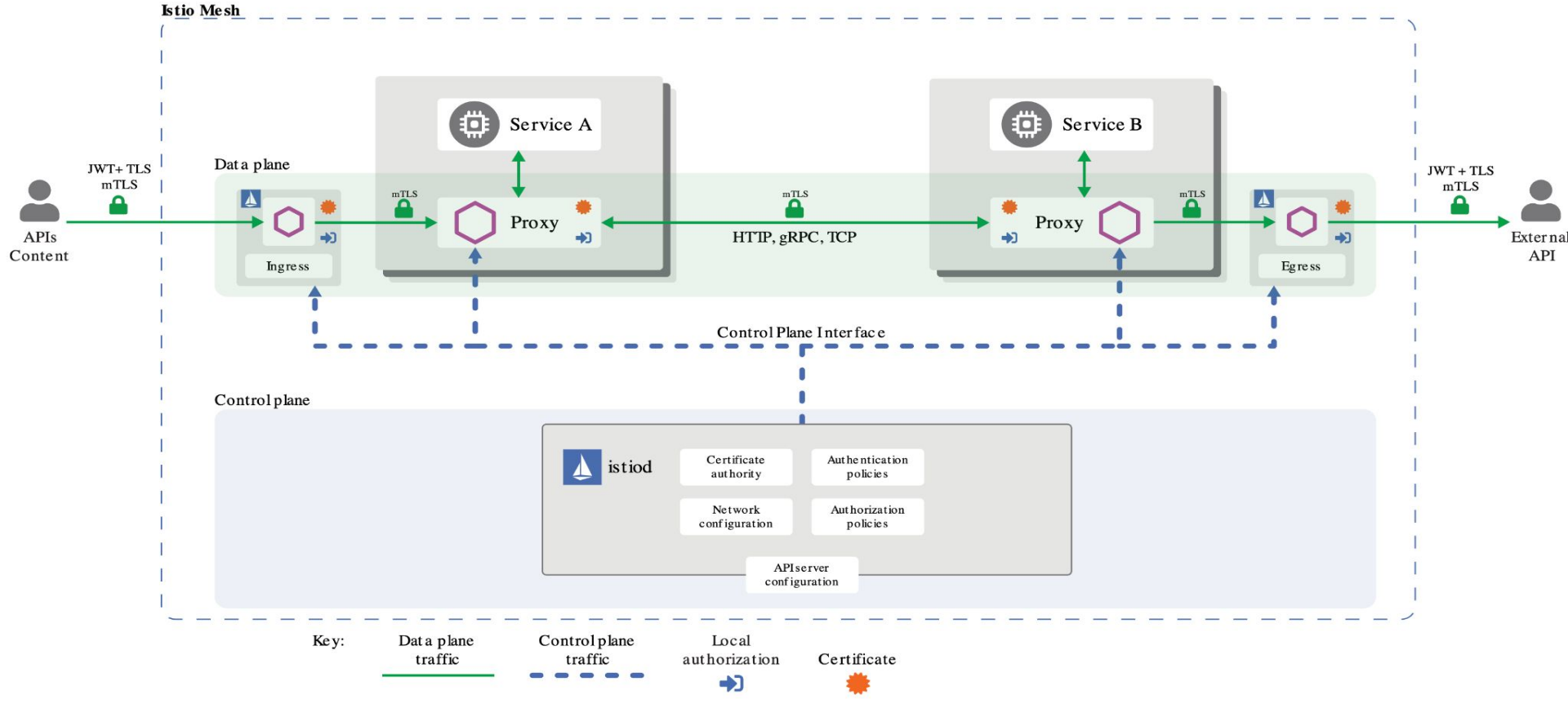
Auth Patterns: What to Use and When

Aaron Teague

About this talk

- An introductory on Authentication and Authorization
- Use cases for the different options
- Supplemental tools to complete the AuthN/Z toolset

Within the mesh: mTLS



AuthN+Z using mTLS

```yaml
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: foo
spec:
  mtls:
    mode: STRICT
```

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin
  namespace: foo
spec:
  action: DENY
  rules:
  - from:
    - source:
        namespaces: ["dev"]
    to:
    - operation:
        methods: ["POST"]
```

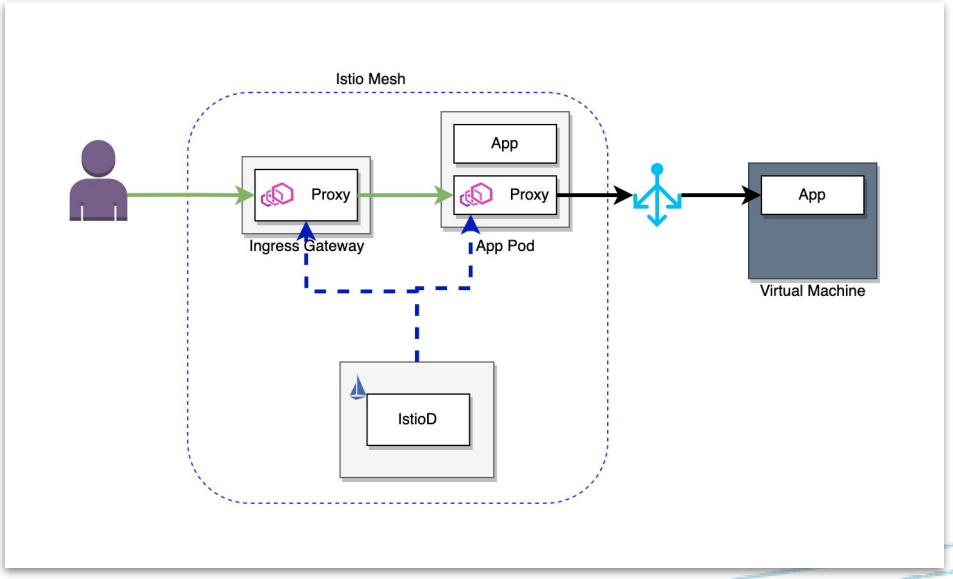

AuthorizationPolicy by example

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  namespace: ns1
  name: anyname
spec:
  selector:
    matchLabels:
      app: myapi
  action: ALLOW
  rules:
  - to:
    - operation:
        methods: ["GET"]
        paths: ["/user/profile/*"]
```

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: httpbin
 namespace: foo
spec:
 selector:
   matchLabels:
     app: httpbin
     version: v1
 action: ALLOW
 rules:
 - from:
   - source:
       principals: ["cluster.local/ns/default/sa/sleep"]
   to:
   - operation:
       methods: ["GET"]
   when:
   - key: request.headers[version]
     values: ["v1", "v2"]
```


From meshed to non-meshed service

- Using External app's authN/Z methods
  - API Key
  - Basic Auth
  - JWT
- Custom mTLS via client certificate
  - Create own cert from shared CA

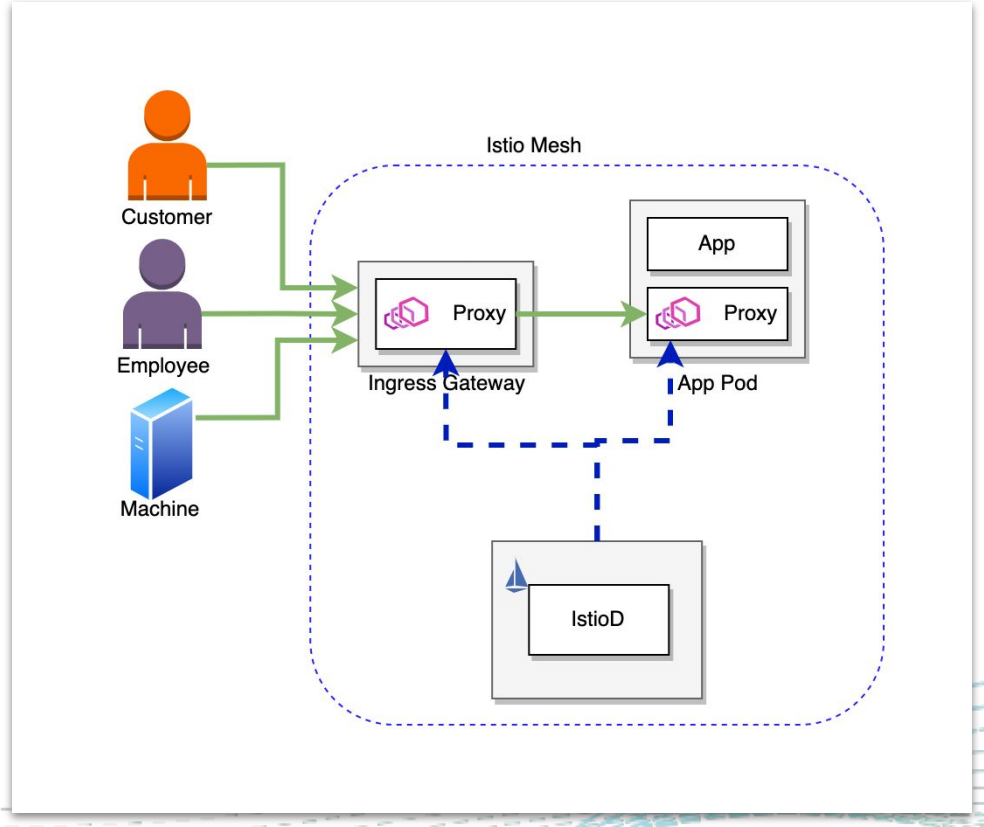


mTLS against non-meshed services

```yaml
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: external-svc-mongocluster
spec:
  hosts:
  - mymongodb.somedomain # not used
  addresses:
  - 192.192.192.192/24 # VIPs
  ports:
  - number: 27018
    name: mongodb
    protocol: MONGO
  location: MESH_INTERNAL
  resolution: STATIC
  endpoints:
  - address: 2.2.2.2
  - address: 3.3.3.3
```

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: mtls-mongocluster
spec:
  host: mymongodb.somedomain
  trafficPolicy:
    tls:
      mode: MUTUAL
      clientCertificate: /etc/certs/myclientcert.pem
      privateKey: /etc/certs/client_private_key.pem
      caCertificates: /etc/certs/rootcacerts.pem
```


To meshed app from outside client



The Mighty JWT

```
aaron@Aarons-MacBook-Pro ~ % gcloud auth print-identity-token
eyJhbGciOiJSUzI1NiIsImtpZCI6ImQzMzJhYjU0NWNjMTg5ZGYxMzNlZmRkYjNhNmM0MDJlYmY0ODlhYzIiLCJ0eXAiOiJKV1QifQ.eyJhdWQiOiIzMjU1NTk
0MDU1OS5hcHBzLmdvb2dsZXVzZXJjb250ZW50LmNvbSIsImF6cCI6Imphc3BlckBhbWF6aW5nLXNvdXJjZS0zNDA0MjAuaWFtLmdzZXJ2aWNlYWNjb3VudC5jb
20iLCJlbWFpbCI6Imphc3BlckBhbWF6aW5nLXNvdXJjZS0zNDA0MjAuaWFtLmdzZXJ2aWNlYWNjb3VudC5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiZXh
wIjoxNjUwODUwMTc3LCJpYXQiOjE2NTA4NDY1NzcsImlzcyI6Imh0dHBzOi8vYWNjb3VudHMuZ29vZ2xlLmNvbSIsInN1YiI6IjExODQzMTc1NzQxNDIwMzQyM
jkxMiJ9.CXMGRL7yM59aZW2nAZqe9eyQZrH30powD4WYpjSmtfB6XCP7iS0pv-1lbJ5uEQUsqV8RngCAHvhOGPp4Pt44jXJARbkckCjZOspfKZfd51fwuPlKiu
r6KyemZu-yZzfiANO04KNgxi1c9eFv91uNfWIratt0f5lc-yWCYddzF1DwDZ85aJbPEnKJLtK9JyOynwzwjmyf_vYWNB9pcYMR6HX3mhYvTSyp8Vzd3zB3gIFy
TRpXgsXqC9gDEROKt5drCr7w9nzSBuPYOPpbdQPqb6dAltVeMcxqLgIqANMpLw6RmI1WMFtiwgryryeNg3b5FxJpDNxt8I5Utm7l0vTz1A
```

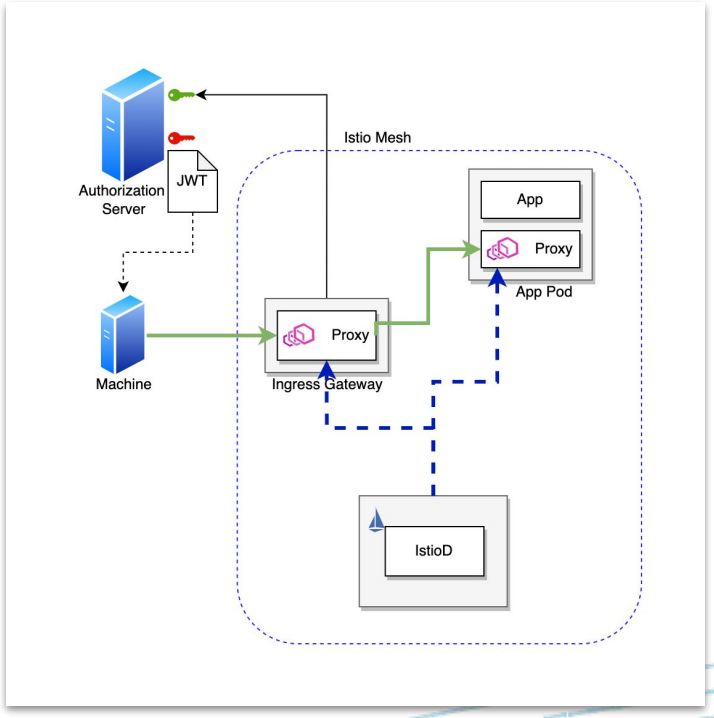
PAYLOAD: DATA


```
{
  "aud": "32555940559.apps.googleusercontent.com",
  "azp": "jasper@amazing-source-
340420.iam.gserviceaccount.com",
  "email": "jasper@amazing-source-
340420.iam.gserviceaccount.com",
  "email_verified": true,
  "exp": 1650850177,
  "iat": 1650846577,
  "iss": "https://accounts.google.com",
  "sub": "118431757414203422912"
}
```


- JWT - A fancy data structure
- Asymmetric encryption via an authorization server

JWT AuthN/Z in Istio

- Good for **machines**, not great for **people**
- Straight forward flow
- Handled by Istio
- Limited in options



JWT AuthN/Z in Istio by example

```yaml
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: httpbin
  namespace: istio-system
spec:
 selector:
   matchLabels:
     app: istio-ingressgateway
  jwtRules:
  - issuer: "https://accounts.google.com"
#    jwksUri: https://example.com/.well-known/jwks.json # Only needed if Discovery endpoint not available (ISSUER/.well-known/openid-configuration)
---
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: httpbin
  namespace: istio-system
spec:
  selector:
   matchLabels:
     app: istio-ingressgateway
  rules:
  - from:
    - source:
        requestPrincipals: ["https://accounts.google.com/118431757414203422912"] # ISSUER/SUBJECT for jasper@amazing-source-340420.iam.gserviceaccount.com
```

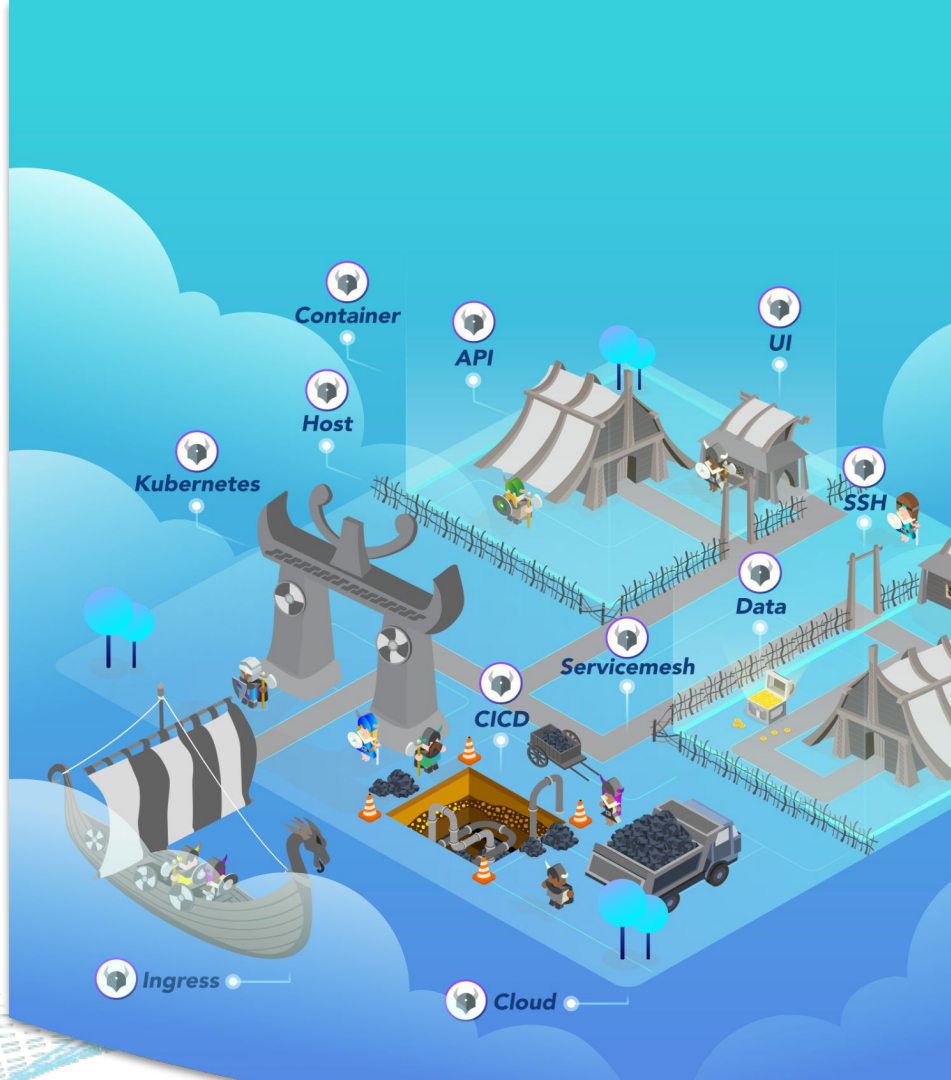

External authorization

```
===
data:
  mesh: |-
    extensionProviders:
    - name: "oauth2-proxy"
      envoyExtAuthzHttp:
        service: "oauth2-proxy.foo.svc.cluster.local"
        port: "4180" # The default port used by oauth2-proxy.
        includeRequestHeadersInCheck: ["authorization", "cookie"] # headers sent to the oauth2-proxy in the check request.
        headersToUpstreamOnAllow: ["authorization", "path", "x-auth-request-user", "x-auth-request-email", "x-auth-request-access-token"]
        headersToDownstreamOnDeny: ["content-type", "set-cookie"] # headers sent back to the client when request is denied.
===
```

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ext-authz
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  action: CUSTOM
  provider:
    # The provider name must match the extension provider defined in the mesh config.
    # You can also replace this with sample-ext-authz-http to test the other external authorizer definition.
    name: oauth2-proxy
  rules:
  # The rules specify when to trigger the external authorizer.
  - to:
    - operation:
        paths: ["/"]
```


OpenPolicyAgent

```
package istio.authz

import input.attributes.request.http as http_request
import input.parsed_path

default allow = false

allow {
    parsed_path[0] == "health"
    http_request.method == "GET"
}

allow {
    roles_for_user[r]
    required_roles[r]
}

roles_for_user[r] {
    r := user_roles[user_name][_]
}

required_roles[r] {
    perm := role_perms[r][_]
    perm.method == http_request.method
    perm.path == http_request.path
}

user_name = parsed {
    [_, encoded] := split(http_request.headers.authorization, " ")
    [parsed, _] := split(base64url.decode(encoded), ":")
}

user_roles := {
    "alice": ["guest"],
    "bob": ["admin"]
}

role_perms := {
    "guest": [
        {"method": "GET",  "path": "/productpage"},
    ],
    "admin": [
        {"method": "GET",  "path": "/productpage"},
        {"method": "GET",  "path": "/api/v1/products"},
    ],
}
```

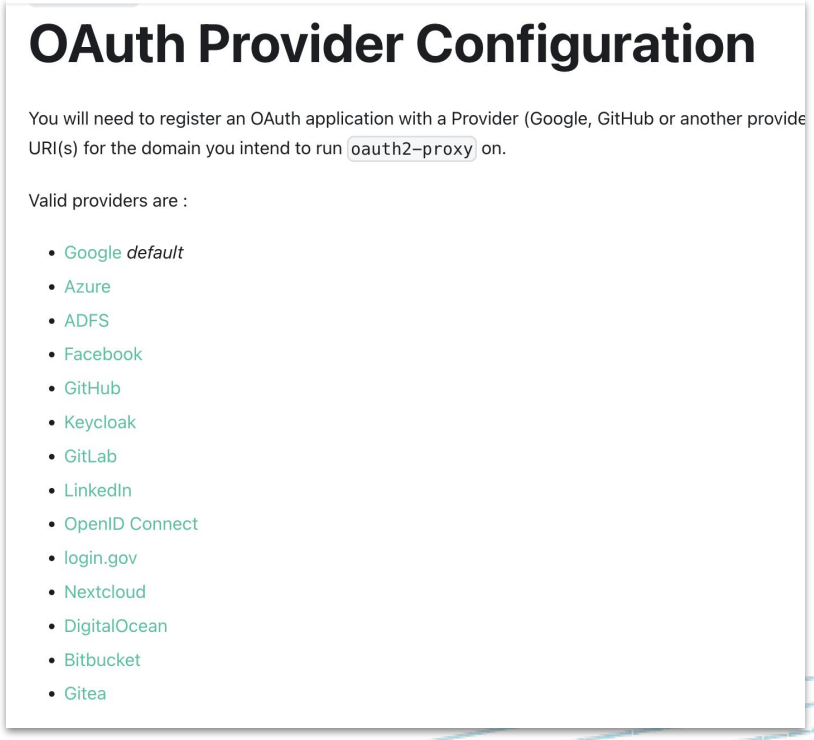


OpenPolicyAgent continued

- Good for machine to service
- *Maybe* okay for employee to service
- An “as config” option for policy management
- Seperate from Istio, can give to separate team
- Very programmable

oauth2-proxy

- Great for humans
- Proxy to leverage other authentication services
- Uses OIDC of upstream service for authN/Z



OAuth Provider Configuration

You will need to register an OAuth application with a Provider (Google, GitHub or another provide URI(s) for the domain you intend to run `oauth2-proxy` on.

Valid providers are :

- Google *default*
- Azure
- ADFS
- Facebook
- GitHub
- Keycloak
- GitLab
- LinkedIn
- OpenID Connect
- login.gov
- Nextcloud
- DigitalOcean
- Bitbucket
- Gitea

oauth2-proxy configuration (employee)

The following config should be set to ensure that the oauth will work properly. To get a cookie secret follow these steps

```
--provider="gitlab"
--redirect-url="https://myapp.com/oauth2/callback" // Should be the same as the redirect url for th
--client-id=GITLAB_CLIENT_ID
--client-secret=GITLAB_CLIENT_SECRET
--cookie-secret=COOKIE_SECRET
```

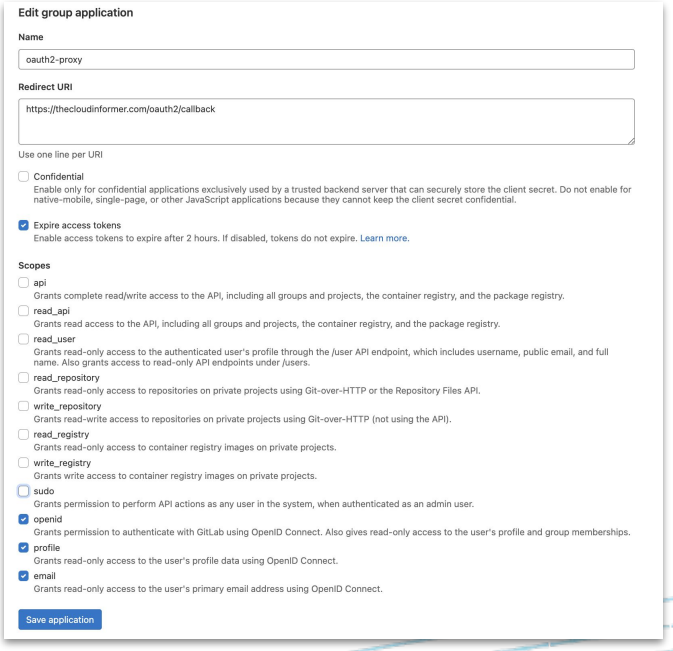
Copy

Restricting by group membership is possible with the following option:

```
--gitlab-group="mygroup,myothergroup": restrict logins to members of any of these groups (slug), separa
```

If you are using self-hosted GitLab, make sure you set the following to the appropriate URL:

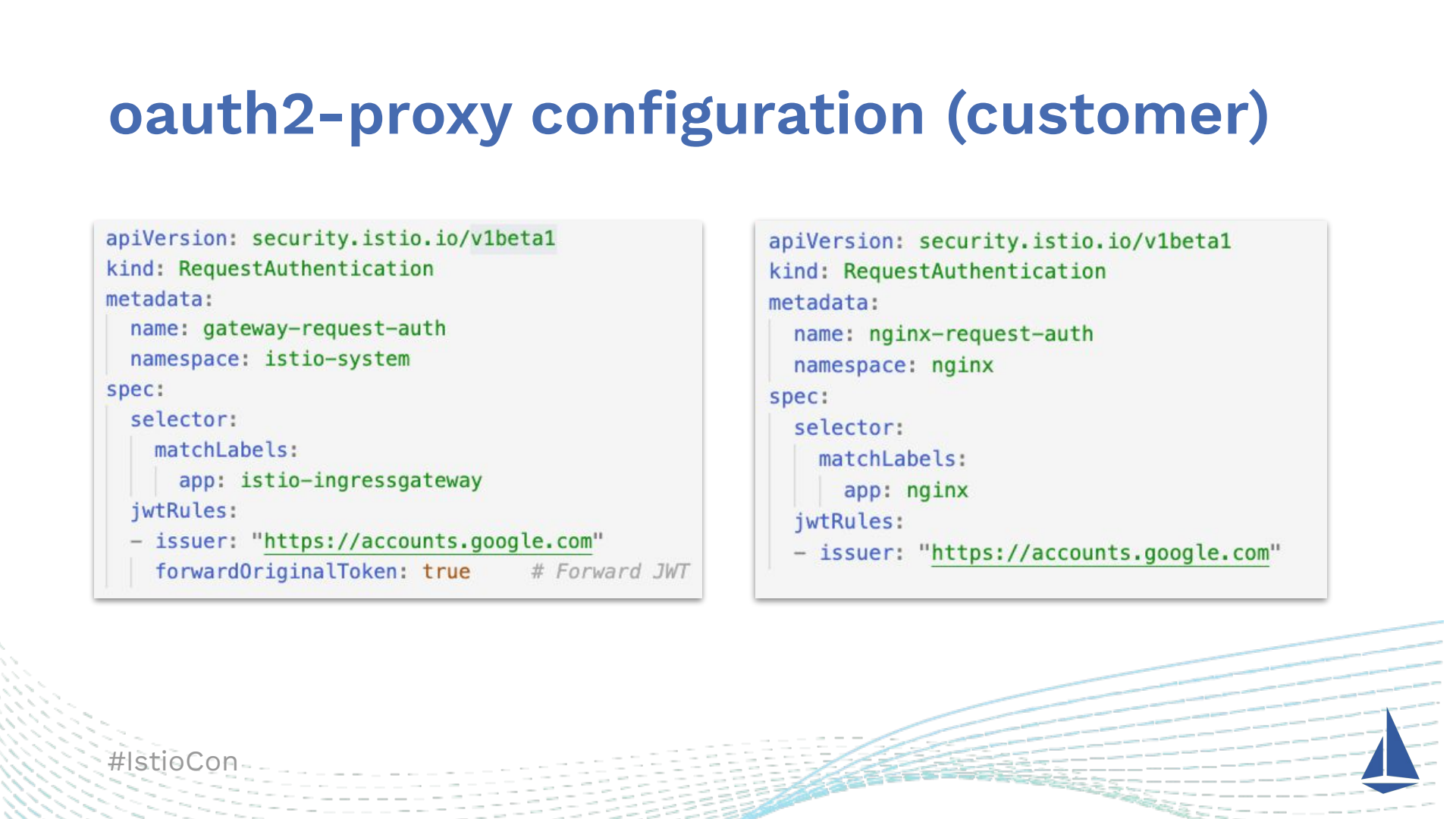
```
--oidc-issuer-url="<your gitlab url>"
```



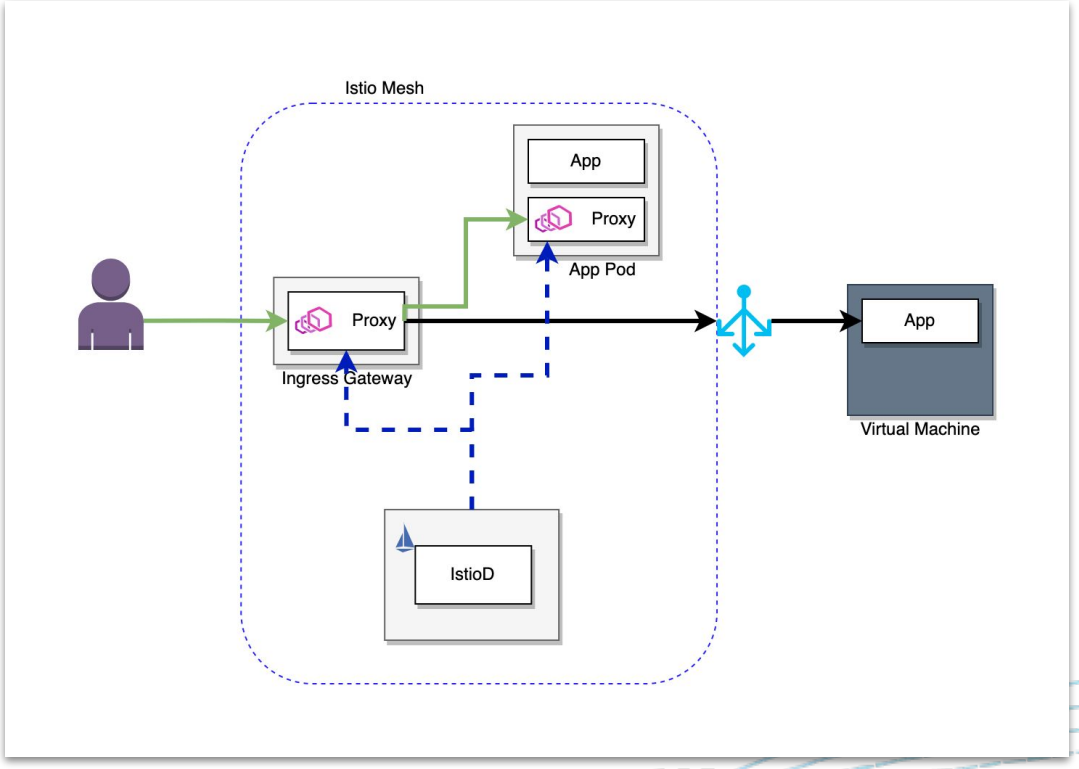
oauth2-proxy configuration (customer)

```yaml
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: gateway-request-auth
  namespace: istio-system
spec:
  selector:
    matchLabels:
      app: istio-ingressgateway
  jwtRules:
  - issuer: "https://accounts.google.com"
    forwardOriginalToken: true      # Forward JWT
```

```yaml
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: nginx-request-auth
  namespace: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  jwtRules:
  - issuer: "https://accounts.google.com"
```

Bonus: Authentication for non-mesh apps



What about API gateways

Istio Ingress Gateway

- Load Balancing
- TLS Termination
- Advanced Traffic Routing
- JWT Validation

API Gateway

- Load Balancing
- TLS Termination
- Advanced Traffic Routing
- JWT Validation
- Request Billing
- Data Transformation
- API Quota

In summary

- **Within mesh:** *Use mTLS*
- **From mesh to non-mesh service:** *non-mesh app's auth* or *mTLS*
- **Machine to the mesh:** *JWT from client machine*
- **Employee to the mesh:** *OPA* or *oauth2-proxy*
- **Customer to the mesh:** oauth2-proxy or API Gateway
- **Extending our authN/Z:** Re-use Ingress Gateway to upstream
- **API gateways:** Use them or don't. You decide!

Thank you!