KubeCon | CloudNativeCon

North America 2024

Powering Automatic Authorization in Envoy Through Live Traffic Inspection

Dom Delnano - CEO/Founder Cosmic and Pixie core maintainer

Hi 👋

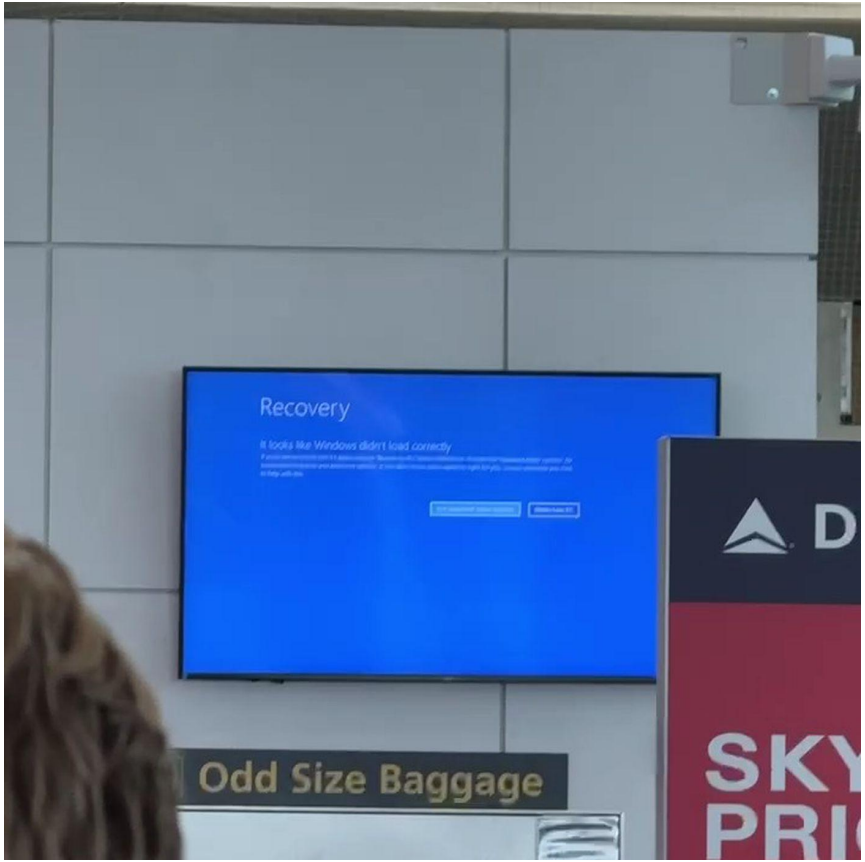
@ddelnano

Pixie core maintainer
CEO/Founder - Cosmic
Previous: CrowdStrike, New Relic, Twitter

PIXIE

Disclaimer

- Security is not a one size fits all solution
- Always consult your security experts

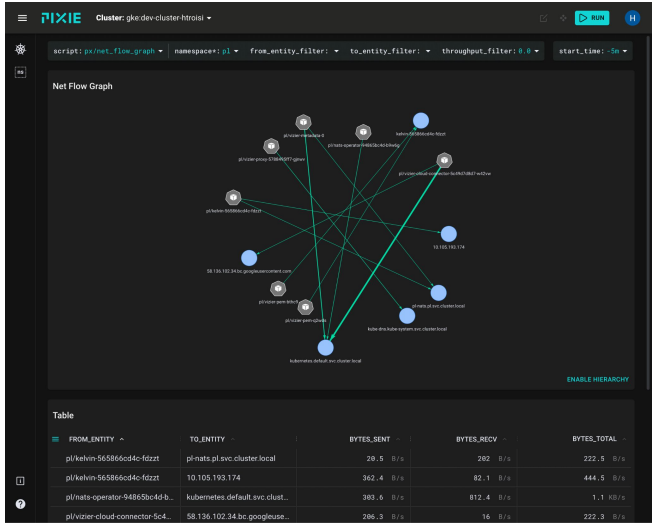


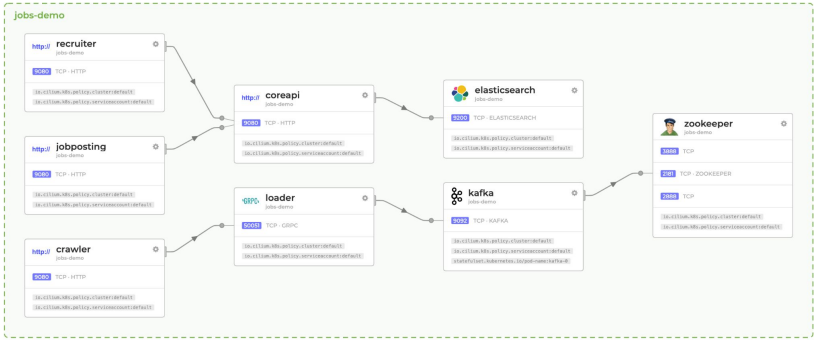
Motivation

- Security Misconfiguration is #5 in OWSAP's Top 10
- The AuthZ policies of last week, month, year eventually suffer from security/configuration drift
- Imagine a world where AuthZ policies change in response to microservice environment changes

Modern Observability to the Rescue

- Zero-instrumentation Observability tools such as Pixie and Hubble provide real time service maps and L7 visibility
- eBPF instruments all network traffic, providing full visibility of an environment
- This unlocks the ability to analyze L7 access patterns and scope access rules accordingly





Goal: Close AuthZ policy gaps as traffic evolves

- Leverage these tools to keep AuthZ policies in sync with existing traffic patterns
  - Provide means for future extension to further restrict L7 access
- This reference architecture will use Pixie and other CNCF components
  - AuthN: mTLS orchestrated by SPIFFE/Spire
  - AuthZ: Envoy with OPA-Envoy
  - Live traffic inspection: Pixie
  - Big data processing: Spark
- This proof of concept lays the foundation for applying Pixie and these principles to your technology stack

Automatic Authorization Playbook

Prerequisites

- **Understanding your Authentication**
  - Where in the payload is it? How can the details be accessed?
- **Authorization layer processing:**
  - How should the AuthN be processed/transformed in the AuthZ layer?
  - This part will be auto generated

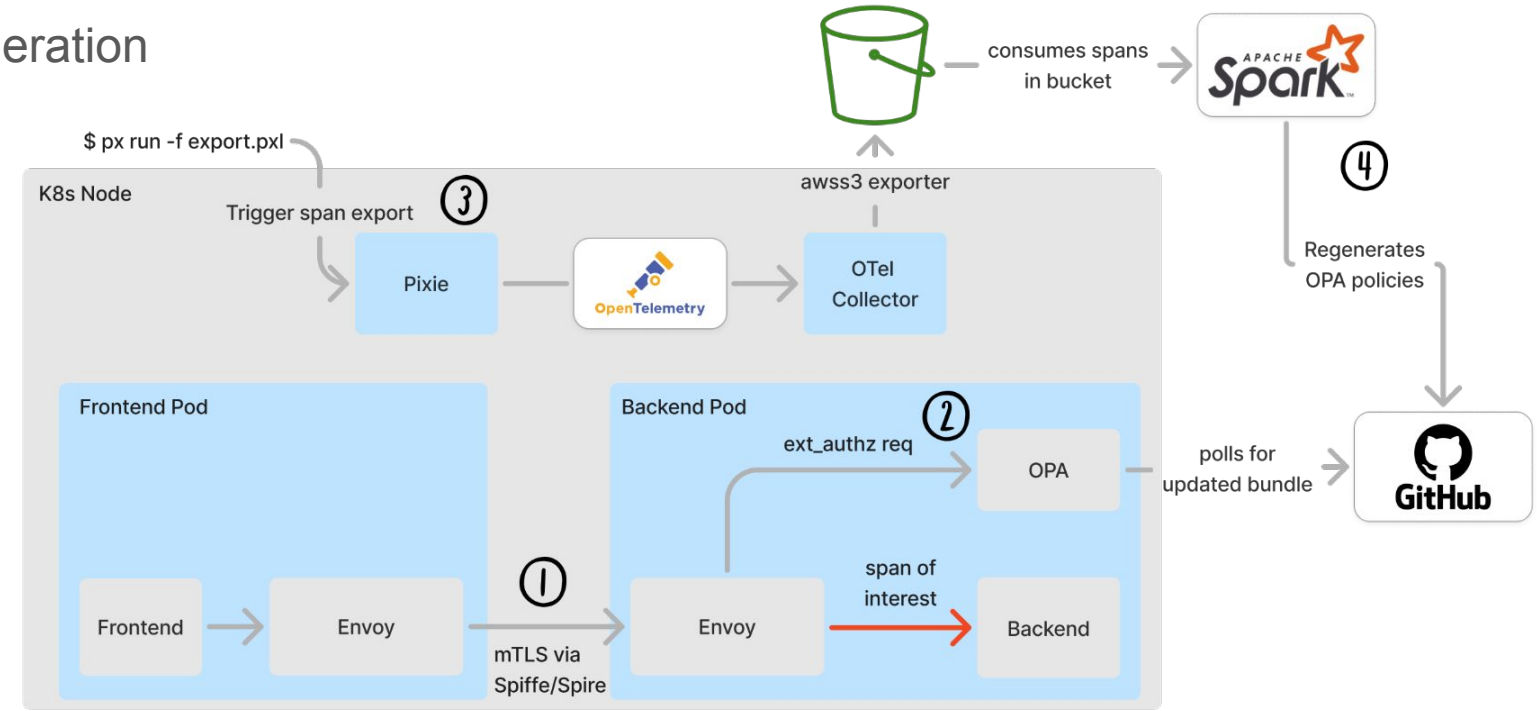
Implementation

1. **Live traffic inspection and data processing:**
   - Input: Network traffic
   - Output: Protocol spans in OTLP format annotated w/ AuthN and L7 access
2. **Span data processing and AuthZ rule generation:**
   - Input: Spans in OTLP format stored in an object store
   - Output: OPA policy files that match traffic contained in OTel spans

Automatic Authorization Playbook

Bring Your Own:
1. AuthN
2. AuthZ
4. Policy generation

Required:
3. Pixie



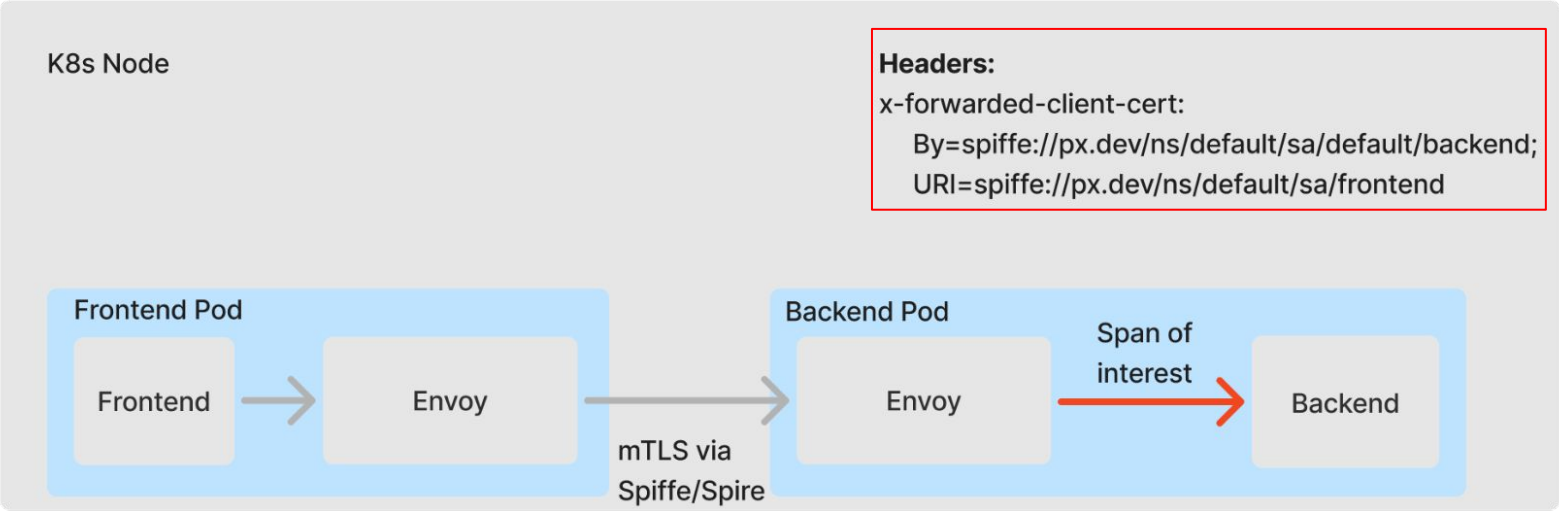
Understanding Your Authentication

- How the AuthN is represented dictates how the AuthZ layer must perform its checks

SPIFFE

- Defines standards for mutual authentication – JWT or X.509 (TLS)
- Easily deployed in cloud native environments w/ Spire – the SPIFFE Runtime Environment
- Spire issues SPIFFE compatible mTLS certs for AuthN and integrates natively with Envoy and other service meshes

Authentication: On the Wire

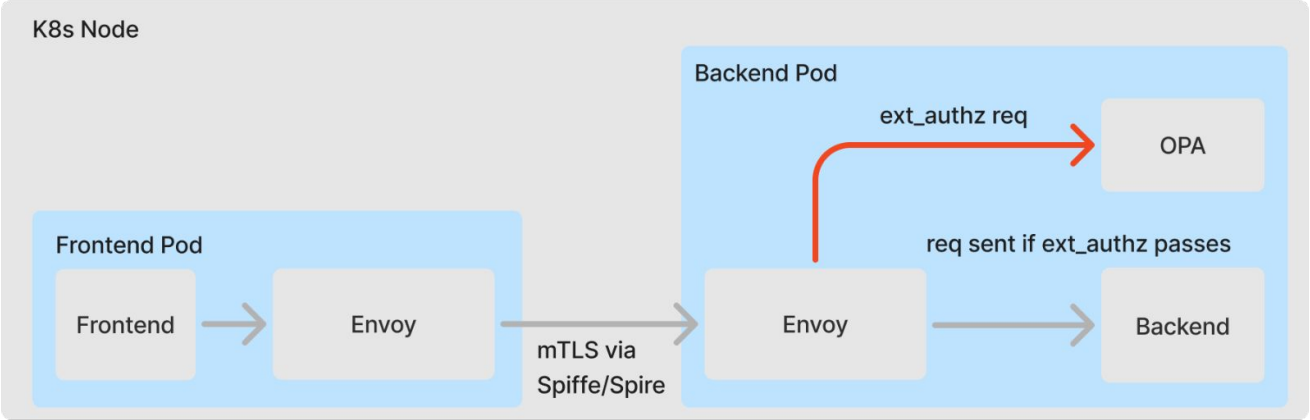


Envoy and ext_authz

- Envoy provides a mechanism for performing external AuthZ checks (ext_authz)
  - Authorization service is sent a request to verify if incoming request is authorized
- Open Policy Agent (OPA) is a popular policy engine with a native integration with Envoy (OPA-Envoy)
- OPA policy has access to details from incoming request to make authorization decision
  - AuthN: HTTP headers
  - L7 access: HTTP method, URI

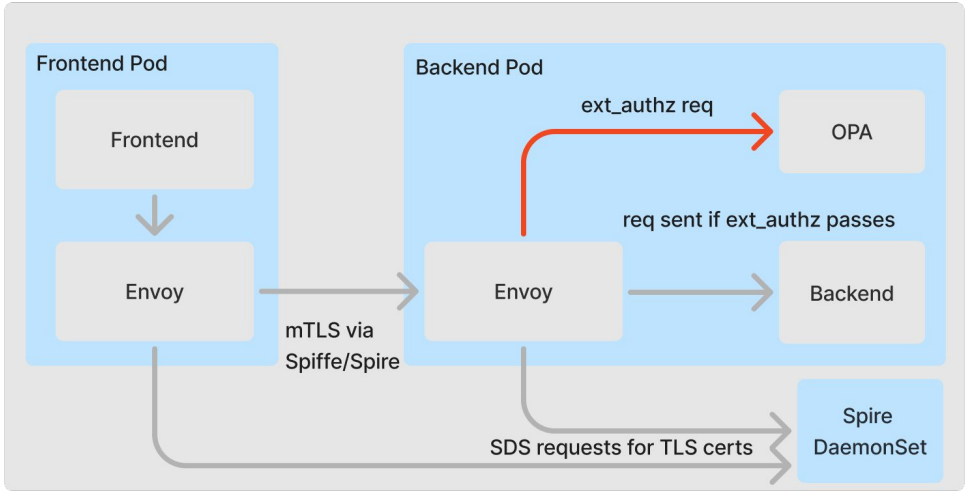
```
package envoy.authz

import rego.v1

import input.attributes.request.http

default allow := false

allow if {
    is_token_valid
    action_allowed
}

is_token_valid if {
    token.valid
}

token := {"valid": valid} if {
    [_, _] := split(http.headers.authorization, " ")
    # Use Authorization header to perform the
    # necessary checks
    valid = true
}

action_allowed if {
    http.method == "GET"
    glob.match("/people/*", ["/"], http.path)
}
```

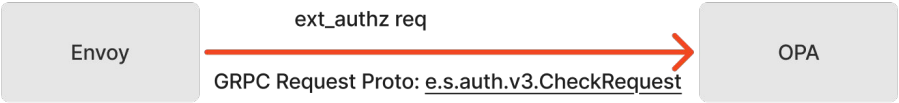
Authorization Architecture



Headers:
x-forwarded-client-cert:
By=spiffe://px.dev/ns/default/sa/default/backend;
URI=spiffe://px.dev/ns/default/sa/frontend

Authorization: Traffic to OPA Policy





GRPC Request Proto: e.s.auth.v3.CheckRequest
Headers:

x-forwarded-client-cert:
By=spiffe://px.dev/ns/default/sa/default/backend;
URI=spiffe://px.dev/ns/default/sa/frontend

OPA AuthZ policy based on traffic patterns

```
package envoy.authz

import rego.v1

default allow := false

# By=spiffe://.../backend;Hash=<hash>;URI=spiffe://.../frontend
headers := input.attributes.request.http.headers
xfcc := headers["x-forwarded-client-cert"]

# Parse XFCC to pull out src and dest service names
[dest_svc, source_svc] := dest_source_spiffe_svc if {
    # [ ... ]
}

# authz_by_service templated based on client/server details
# in spans and HTTP details (method, URI, etc)
allowed_sources := object.get(authz_by_service, dest_svc, {})
src_svc_http_access := object.get(allowed_sources, src_svc, {})

allow if {
    # Verify src svc is allowed to access dest
    some dest, sources in authz_by_service
    dest == dest_svc

    some src, _ in sources
    src == src_svc
    # Verify L7 access should be permitted
    check_l7_access(src_svc_http_access, http_path, http_method)
}
```

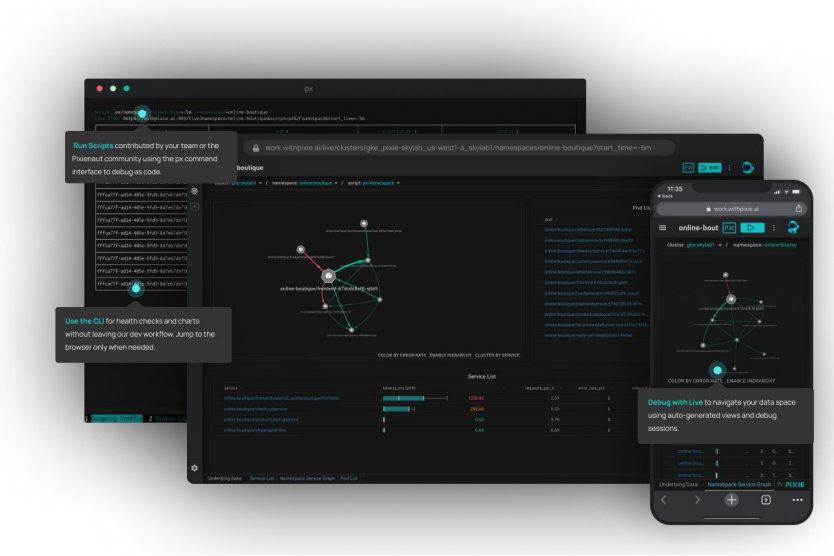
Introduction to Pixie

Goal: Performance debugging without manual instrumentation

- CPU/Memory/Network
- Protocol tracing (spans) w/ latency and payload
- Performance profiles (flamegraphs)

Characteristics

- **Zero-instrumentation:** eBPF captures all data w/o application changes
- **Distributed architecture** with in-memory data store
- **Scriptable Python/Pandas interface** to query and process telemetry



Pixie is Scriptable

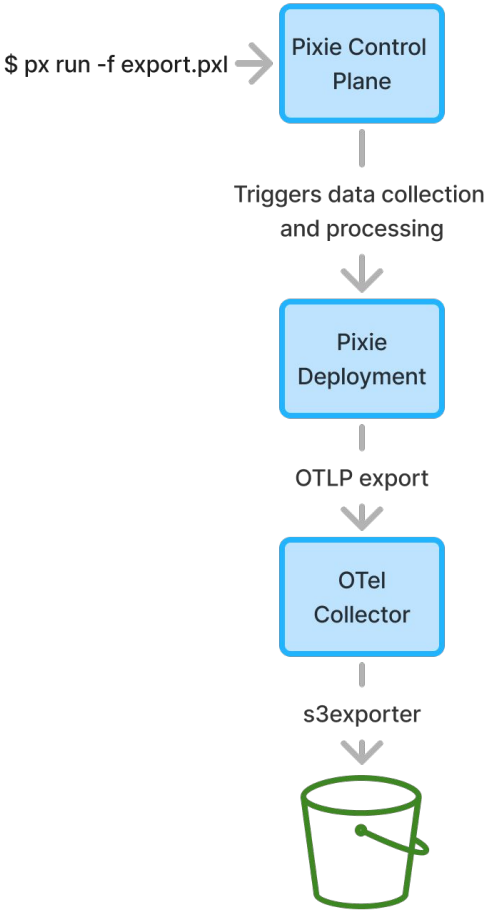
- Valid python™
- Valid pandas
- Built for data analysis and ML
- UI widgets and data processing completely configurable
- Integrates in the o11y ecosystem
  - OTel export
  - Grafana data source

```python
import px

def http_data():
    df = px.DataFrame(table='http_events', start_time='-30s')

    df.pod = df.ctx['pod']  # Annotate dataframe with k8s pod

    df.xfcc = px.pluck(df.req_headers, 'x-forwarded-client-cert')

    return df[['pod', 'req_path', 'resp_latency_ns', 'xfcc']]

px.display(http_data())
```

Live Traffic Inspection: Pixie

- Pixie's PxL language provides a means for enriching protocol spans
  - Src and dest k8s service/deployment/pod
  - L7 access (HTTP method, URL)
  - AuthN details – transformation of XFCC header

- Pairing its OTel export with the OTel collector makes it easy to send spans to object storage for big data processing



Span data processing and Policy Generation

OTel collector S3 export path

s3://.../year=YYYY/month=MM/day=DD/hour=hh/min=mm/*.json

```
# Entry for each span. OTLP format highly nested
[
  [
    {
      "http.xfcc_by": "By=spiffe://example.org/ns/default/sa/default/backend",
      "http.xfcc_uri": "URI=spiffe://example.org/ns/default/sa/default/frontend",
      "http.method": "GET",
      "http.url": "/balances/balance_1"
    },
    {
      "http.xfcc_by": "By=spiffe://example.org/ns/default/sa/default/backend",
      "http.xfcc_uri": "URI=spiffe://example.org/ns/default/sa/default/frontend",
      "http.method": "GET",
      "http.url": "/transactions/transaction_2"
    },
    [ ... ]
  ]
]
```

Traces of allowed traffic from Pixie

Aggregate and output OPA policy files

```
$ tree opa
opa
├── imports
│   ├── backend_a.rego
│   ├── backend_b.rego
│   ├── backend_c.rego
│   └── backend_d.rego
└── opa-policy.rego # Main file that uses all imports

2 directories, 7 files
```

OPA Policies that reflect current live traffic

Span data processing and Policy Generation


```
$ tree opa
opa
├── imports
│   ├── backend_a.rego
│   ├── backend_b.rego
│   ├── backend_c.rego
│   └── backend_d.rego
└── opa-policy.rego # Main file that uses all imports

2 directories, 7 files
```

```
# imports/backend_a.rego
package px.service.backend_a

valid_sources := {
  "frontend": [
    {
      "method": "GET",
      "prefix": "/profiles/profile_1",
    },
  [ ... ]
  ]
}
```

```
# opa-policy.rego

# AUTOGENERATED: imports are added for each destination service
import data.px.service.backend_a
import data.px.service.backend_b
[ ... ]

authz_by_service := {
    "backend_a": data.px.service.backend_a.valid_sources,
    "backend_b": data.px.service.backend_b.valid_sources,
    [ ... ]
}
# END AUTOGENERATED
```

OPA Policies that reflect current live traffic

Demo

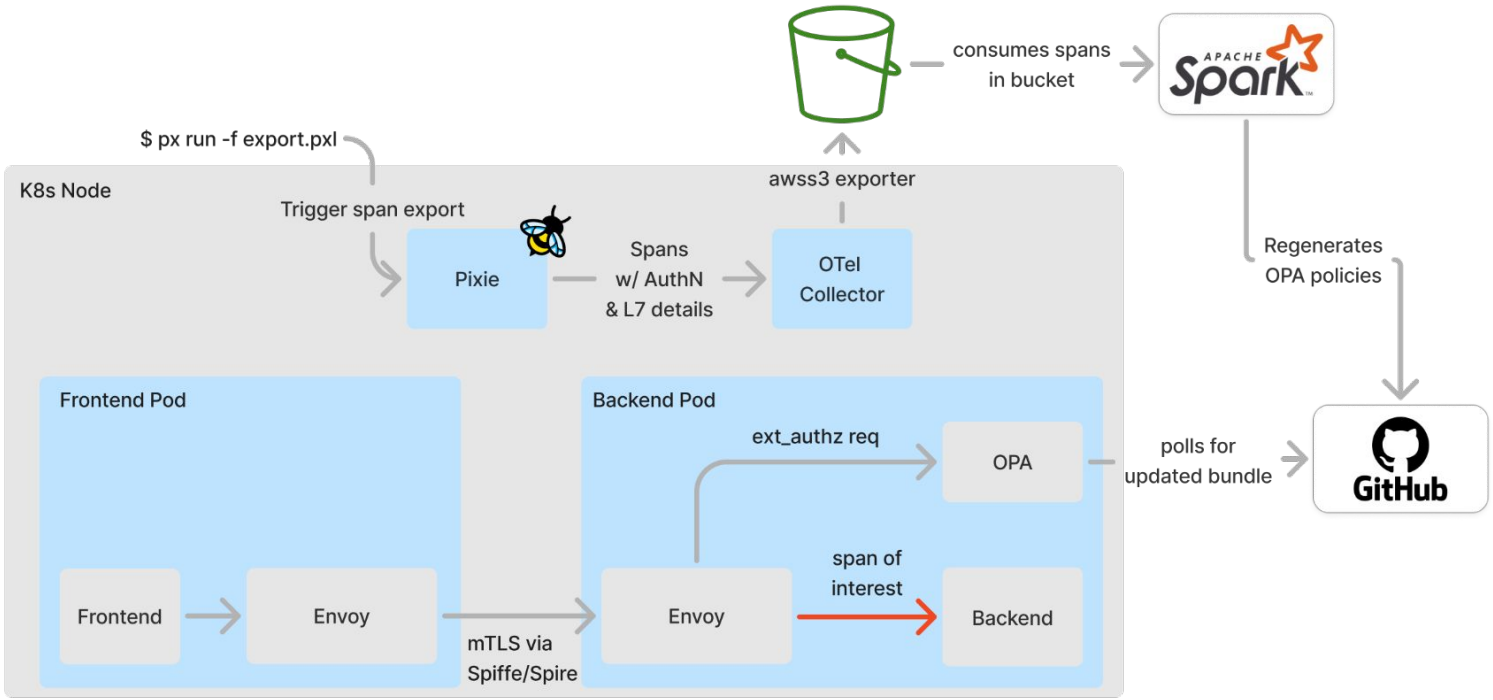
work.testing.getcosmic.ai/live/clusters/ddelnano-test-cluster?destination_filter=default%2F&max_num_records=1000&script=px%2Fhttp_data&sou...

OSMIC cluster: ddelnano-test-cluster RUN DD

script: px/http_data source_filter: destination_filter: default/ max_num_records: 1000 start_time: -5m

Table Showing 1 - 15 out of 1000 records

TIME_	SOUR...	DESTINATION	LATE...	MAJ...	REQ_PATH	REQ_...	REQ_...	REQ_...
11/14/2...	127.0.0.1	default/frontend-2-c54c84d5c-h6bm5	88.8 ms	1	/	GET	{ Accep...	
11/14/2...	127.0.0.1	default/frontend-2-c54c84d5c-h6bm5	27.8 ms	1	/transactions/transaction_2	GET	{ Accep...	
11/14/2...	127.0.0.1	default/backend-57f6f6864c-5552v	180.6 µs	1	/transactions/transaction_2	GET	{ accep...	
11/14/2...	127.0.0.1	default/backend-57f6f6864c-5552v	8.5 ms	2	/envoy.service.auth.v3.Authorization/Check	POST	{ :auth...	1: "\nM\..
11/14/2...	127.0.0.1	default/frontend-2-c54c84d5c-h6bm5	16 ms	1	/profiles/profile_2	GET	{ Accep...	
11/14/2...	127.0.0.1	default/backend-57f6f6864c-5552v	252.5 µs	1	/profiles/profile_2	GET	{ accep...	
11/14/2...	127.0.0.1	default/backend-57f6f6864c-5552v	2.4 ms	2	/envoy.service.auth.v3.Authorization/Check	POST	{ :auth...	1: "\nM\..
11/14/2...	127.0.0.1	default/frontend-2-c54c84d5c-h6bm5	9.9 ms	1	/balances/balance_2	GET	{ Accep...	
11/14/2...	127.0.0.1	default/backend-57f6f6864c-5552v	178.5 µs	1	/balances/balance_2	GET	{ accep...	
11/14/2...	127.0.0.1	default/backend-57f6f6864c-5552v	2.1 ms	2	/envoy.service.auth.v3.Authorization/Check	POST	{ :auth...	1: "\nM\..
11/14/2...	127.0.0.1	default/frontend-5884c9c8bf-hrxqc	73 ms	1	/	GET	{ Accep...	
11/14/2...	127.0.0.1	default/frontend-5884c9c8bf-hrxqc	15.1 ms	1	/transactions/transaction_1	GET	{ Accep...	
11/14/2...	127.0.0.1	default/backend-57f6f6864c-5552v	168.5 µs	1	/transactions/transaction_1	GET	{ accep...	


consumes spans
in bucket
APACHE Spark
$ px run -f export.pxl
K8s Node
Trigger span export
awss3 exporter
Pixie
Spans
w/ AuthN
& L7 details
OTel
Collector
Regenerates
OPA policies
Frontend Pod
Backend Pod
ext_authz req
OPA
polls for
updated bundle
GitHub
span of
interest
Frontend
Envoy
Envoy
Backend
mTLS via
Spiffe/Spire

Future Work: Stricter L7 Policies

- Integrate rule generation logic with richer service route information
  - Many RPC frameworks, API tooling and generated client libraries provide routing information
    - GRPC
    - OpenAPI (Swagger)
- Using this in the OPA policy file generation can produce least privileged access

Recap

- This lays the foundation for continuous enforcement of current traffic patterns
- Eliminate security/configuration drift for AuthZ rules
- Pixie's data processing is key for powering use cases like this
  - Bring Your Own existing AuthN, AuthZ and architecture

Thank you!

pixie-io/pixie

https://px.dev

- Demo - pixie-io/pixie-demos#54