Bridge Your Service Mesh and AWS

Santosh Ananthakrishnan
Harihara K Narayanan

Square Inc.

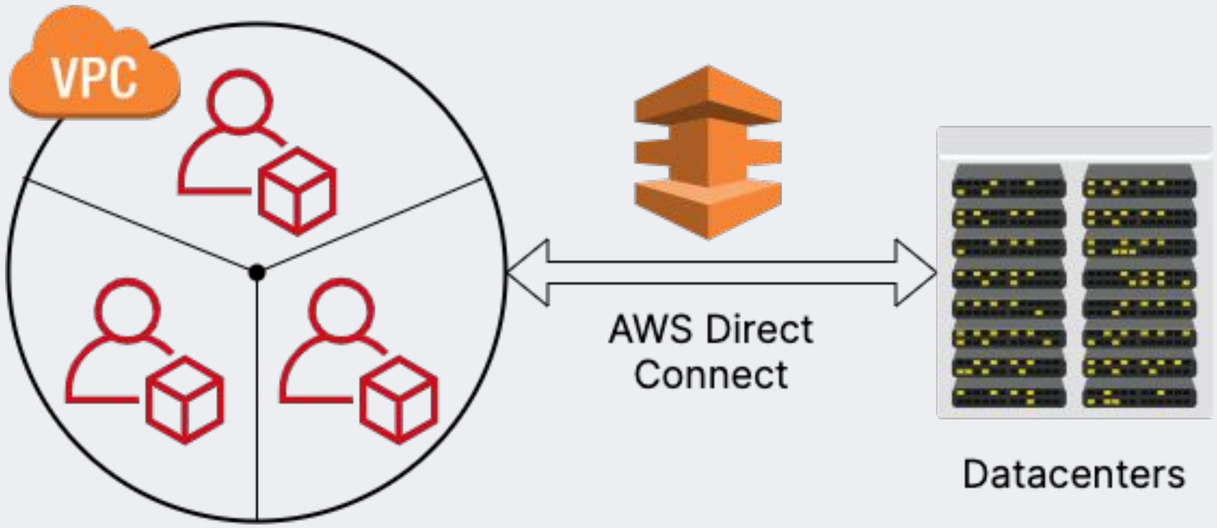
Cloud @ Square

Services (called *apps* internally) spread across our datacenters and cloud providers

Datacenters

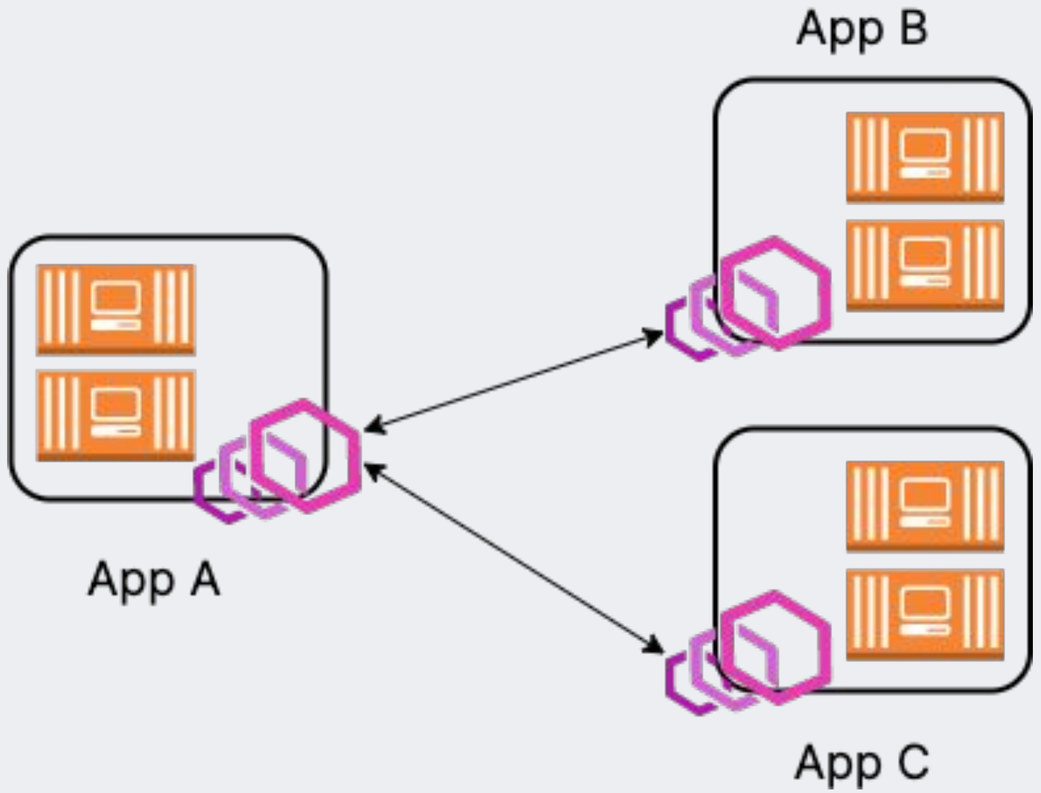
Cloud @ 

- Each app gets its own AWS account for strong isolation guarantees
- Accounts are connected to regional shared VPCs so services can talk to each other



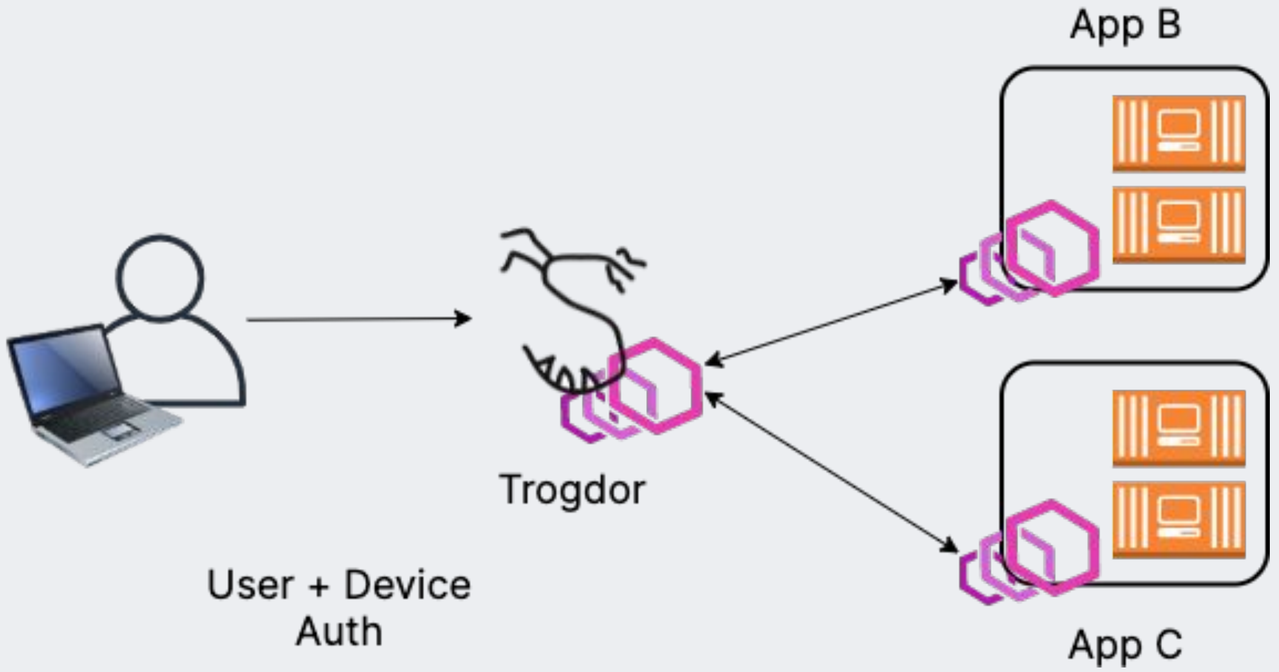
Service Mesh

- Apps talk to each other over Envoy sidecars
- Envoy handles mutual TLS and authorization (is app A allowed to call me?)
- From the app's perspective, reads and writes in the clear from a Unix domain socket



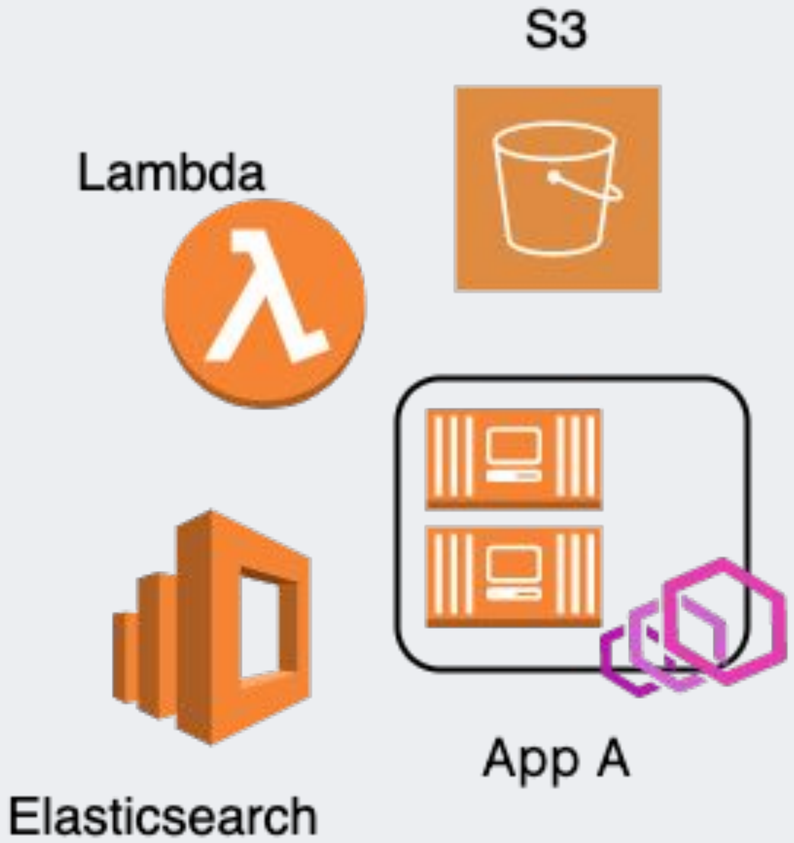
Access for Humans

- Square employees auth to services through a reverse proxy called Trogdor
- Trogdor auths the user and checks they're on a trusted, up to date device
- Talks to backends over Envoy, exactly like other traffic on the service to service mesh



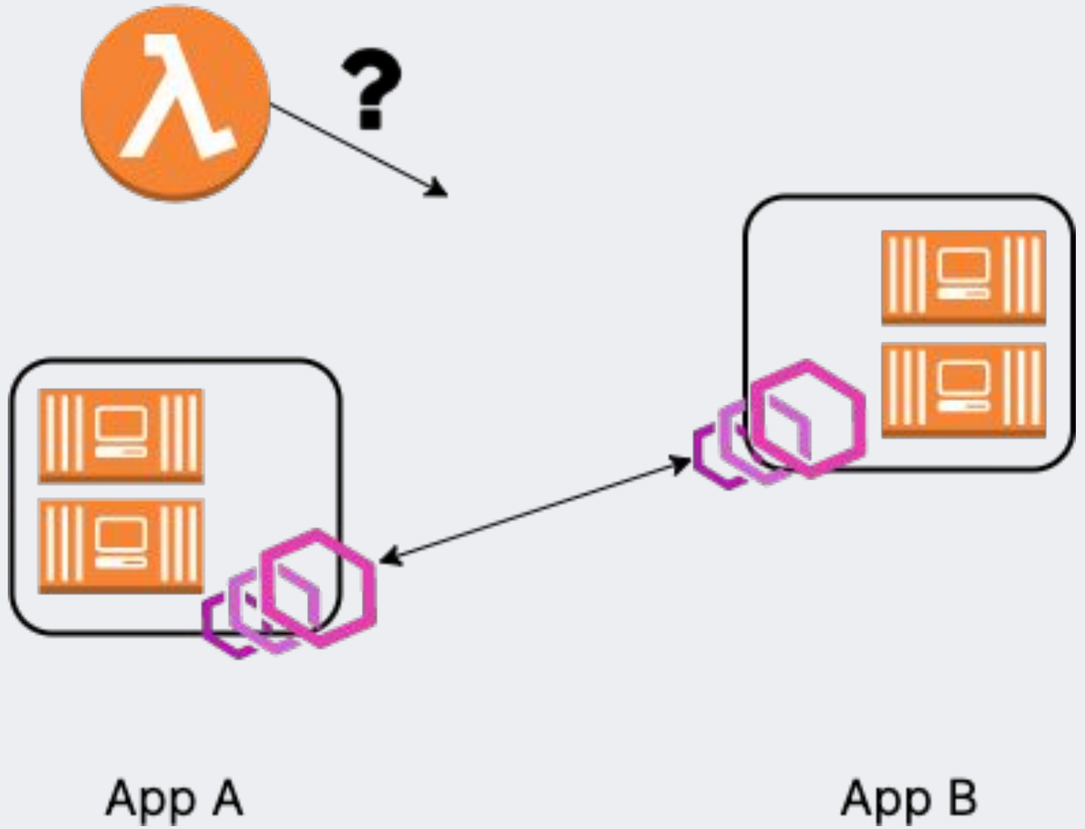
Inter-Operability is Hard

- Apps are more than just containers with a convenient sidecar :-(
- Some code lives in Lambda functions, static files served from S3
- Design ~~Goal~~ Dream: App components should be able to talk to each other *however owners implement them*, *wherever they live* (in the DC or on AWS)



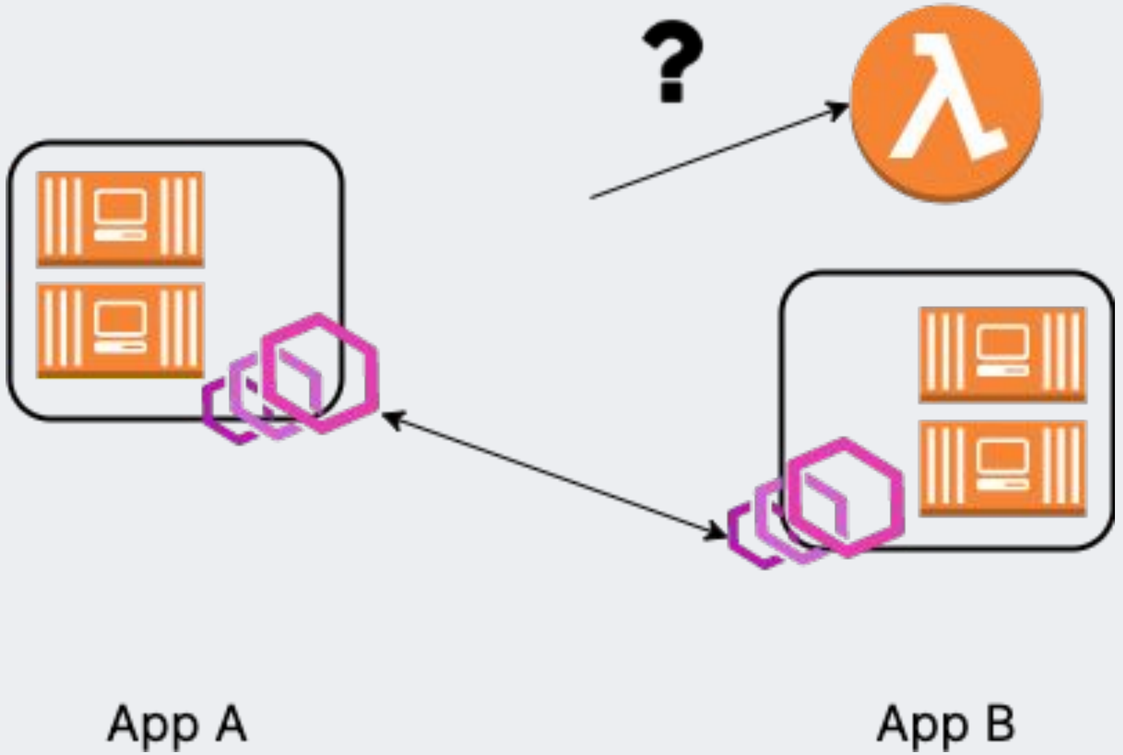
Prior Work

- Early need: support for offloading background async jobs to Lambda
- Built secrets distribution and identity infra to let Lambda functions call into the service mesh
- See [Bridging Security Infrastructure between the Datacenter and AWS Lambda] by Michael Weissbacher (Square) @ BlackHat 2021



Problem: Calling Into Lambda

App B has some application logic in a Lambda function that we'd like other apps to be able to call



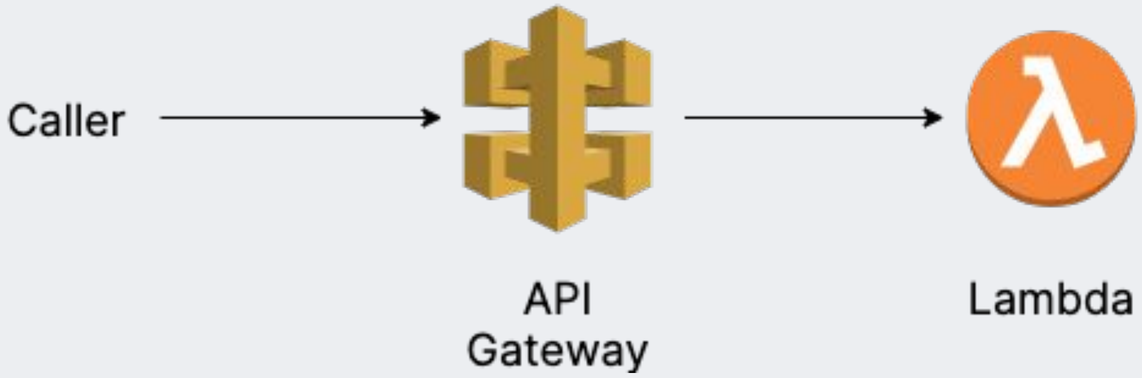
Lambda Invoke

- Lambda execution context does not say which role invoked the function - no easy way to do authz

- Custom schemes like pre-signed STS GetCallerIdentity possible, but prone to bugs (e.g. Vault), hard to deploy, would be bespoke just for Lambda

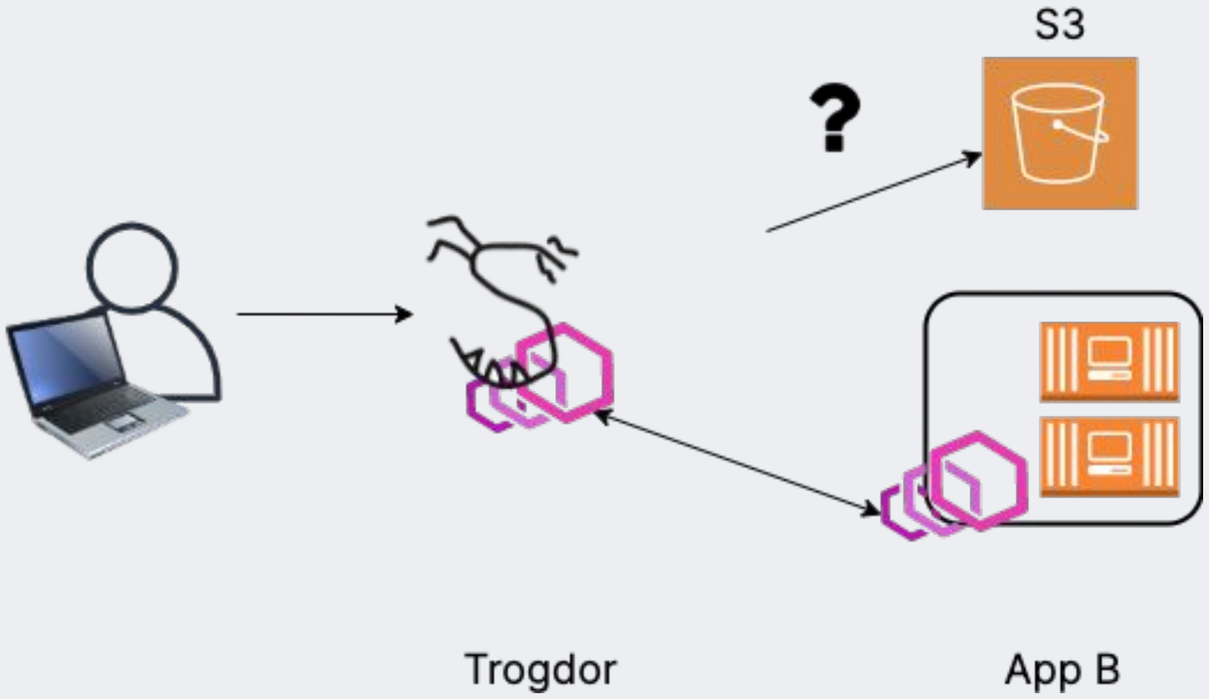
API Gateway

- API Gateway authenticates callers and passes their identity along to Lambda
- No mTLS available for non-public gateways, no SPIFFE SAN verification
- Could build something with Lambda authorizers or JWTs, but this would be a parallel auth mechanism for some kinds of traffic
- Ideally, implement your auth stack well *once*, use the same thing everywhere



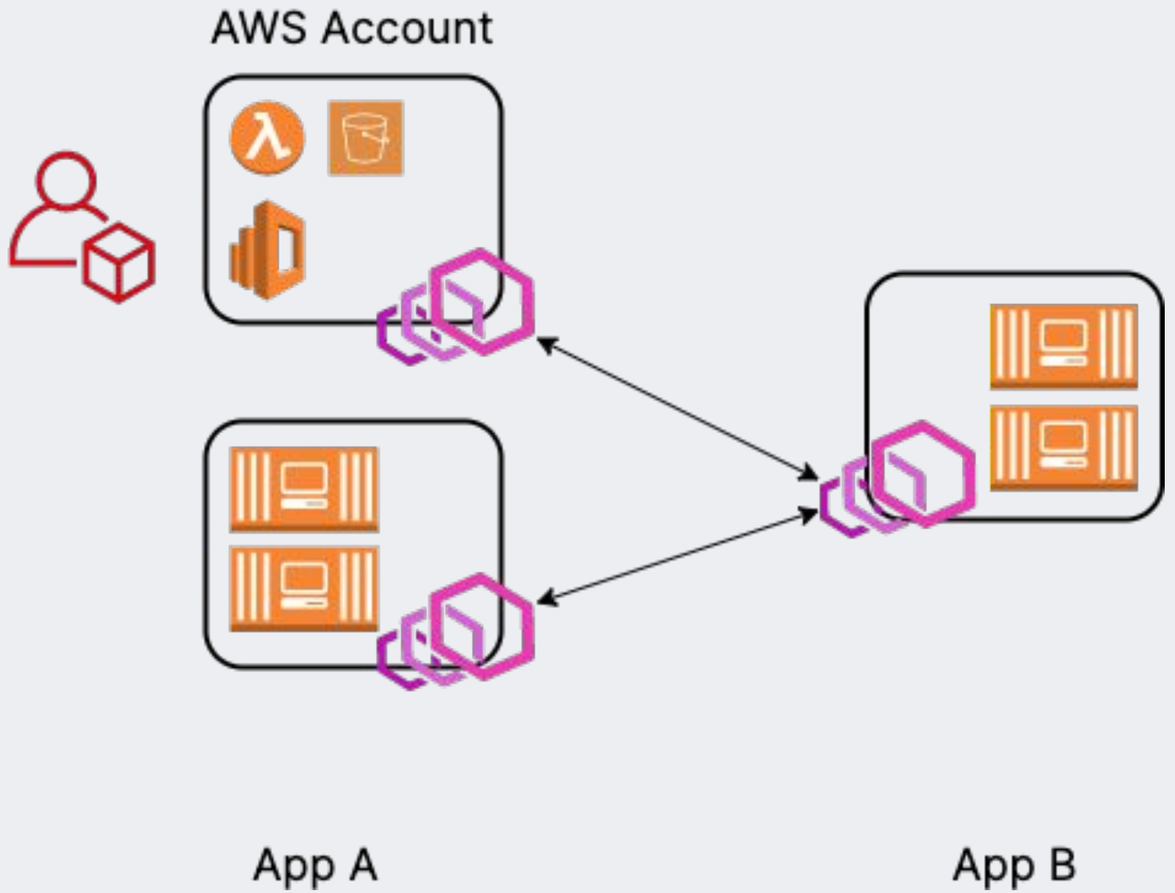
Problem: Internal Static Assets

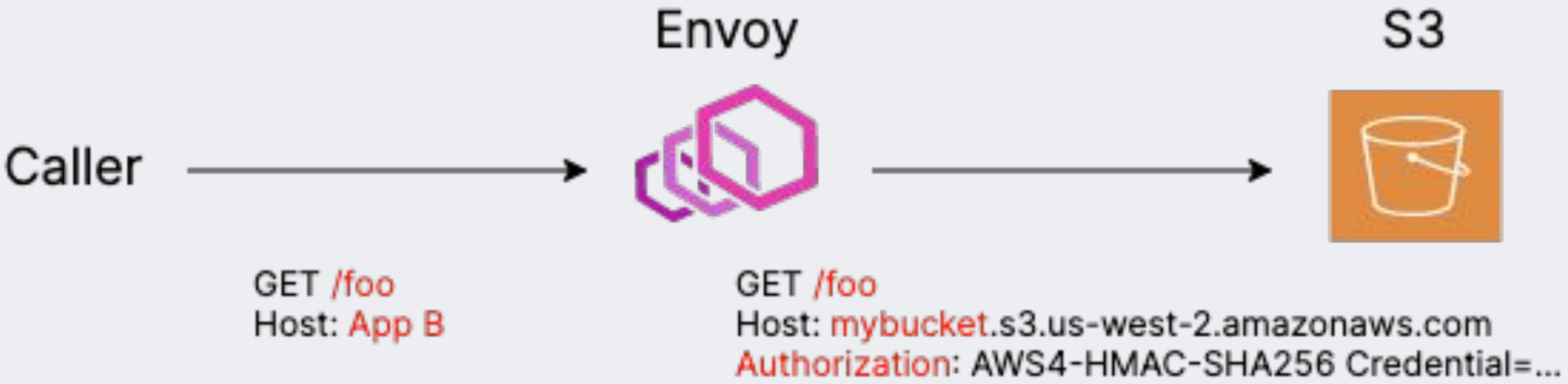
- Making files in an S3 bucket available to users internally (other Square employees) as a static site was a very common ask
- S3 has this support natively, but involves making objects public to everyone. Surprisingly non-trivial on AWS



AWS Account “Sidecar”

- Idea: proxy traffic to cloud resources through Envoy instances
- Instead of talking to a local app container, Envoy forwards incoming requests to an AWS service, like S3
- Envoy acts as a sidecar exactly like in the service mesh, but the backend is AWS





- Each such *ingress* Envoy is configured to point at a individual backend resource (S3 bucket mybucket)
- Incoming requests are signed and forwarded to the appropriate AWS API endpoint
- The caller is talking to an Envoy instance, exactly for any other service to service request. Everything behind the ingress Envoy is an implementation detail of the recipient app, from the perspective of the caller

Envoy SigV4 Signer

```
name: envoy.filters.http.aws_request_signing
typed_config:
  "@type": type.googleapis.com/envoy.extensions.filters.http.aws_request_signing.v3.AwsRequestSigning
  service_name: s3
  region: us-west-2
```

- Envoy contains logic to calculate SigV4 signatures given credentials, added by the AWS AppMesh team
- Turned into a extension that signs and forwards incoming requests by Pinterest ~ Feb 2020

Credentials

- Not using the CPP SDK - implementation only checks if AWS_* environment variables are set, and tries to connect to EC2, ECS IMDS endpoints

- Each ingress Envoy is a pod in our Kubernetes platform, built on EKS. In EKS, IAM roles can be bound to pods, letting pods easily get short-lived creds (this mechanism is called *IRSA*, or **I**AM **R**oles for **S**ervice **A**ccounts)

- Unlike recent SDKs, no support to fetch creds over IRSA in Envoy :-(

Credentials

- Not using the CPP SDK - implementation only checks if AWS_* environment variables are set, and tries to connect to EC2, ECS IMDS endpoints

- Each ingress Envoy is a pod in our Kubernetes platform, built on EKS. In EKS, IAM roles can be bound to pods, letting pods easily get short-lived creds (this mechanism is called *IRSA*, or **I**AM **R**oles for **S**ervice **A**ccounts)

- Unlike recent SDKs, no support to fetch creds over IRSA in Envoy :-(... **or is there**?

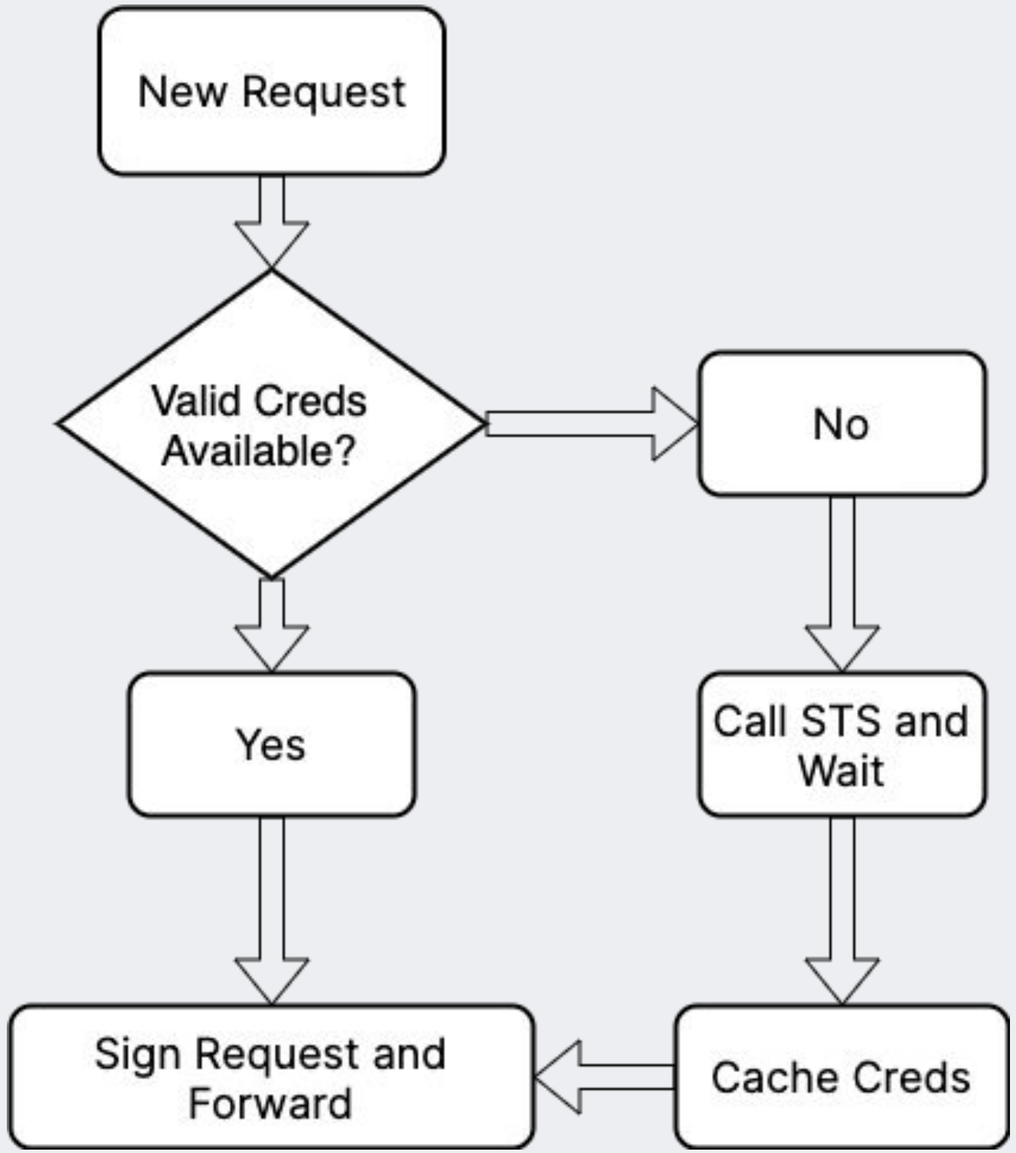
Credentials

```
[2021-03-23 23:35:02.241][7493][debug][aws]
[source/extensions/common/aws/credentials_provider_impl.cc:204] Getting AWS web
identity credentials from STS: sts.us-west-2.amazonaws.com
[2021-03-23 23:35:02.312][7493][debug][aws]
[source/extensions/common/aws/credentials_provider_impl.cc:273] Received the
following AWS credentials from STS: AWS_ACCESS_KEY_ID=ASIA[redacted],
AWS_SECRET_ACCESS_KEY=*****, AWS_SESSION_TOKEN=*****
[2021-03-23 23:35:02.312][7493][debug][aws]
[source/extensions/common/aws/credentials_provider_impl.cc:281] AWS STS credentials
expiration time: 2021-03-24T00:35:02Z
```

- AWS AppMesh's Envoy image supports IRSA! Has been around since 2019
- But changes haven't made it back to upstream yet :-(

Credentials

- Now knew this was possible, just had to build it! … in C++ :-/
- Wrote an internal extension that exchanges IRSA OIDC token for IAM credentials by calling STS AssumeRoleWithWebIdentity
- Credential fetching logic caches returned secrets, keeps track of expiry and refreshes them when needed
- Uses Envoy's async http client interfaces to talk to STS, existing signing logic to calculate Authorization header



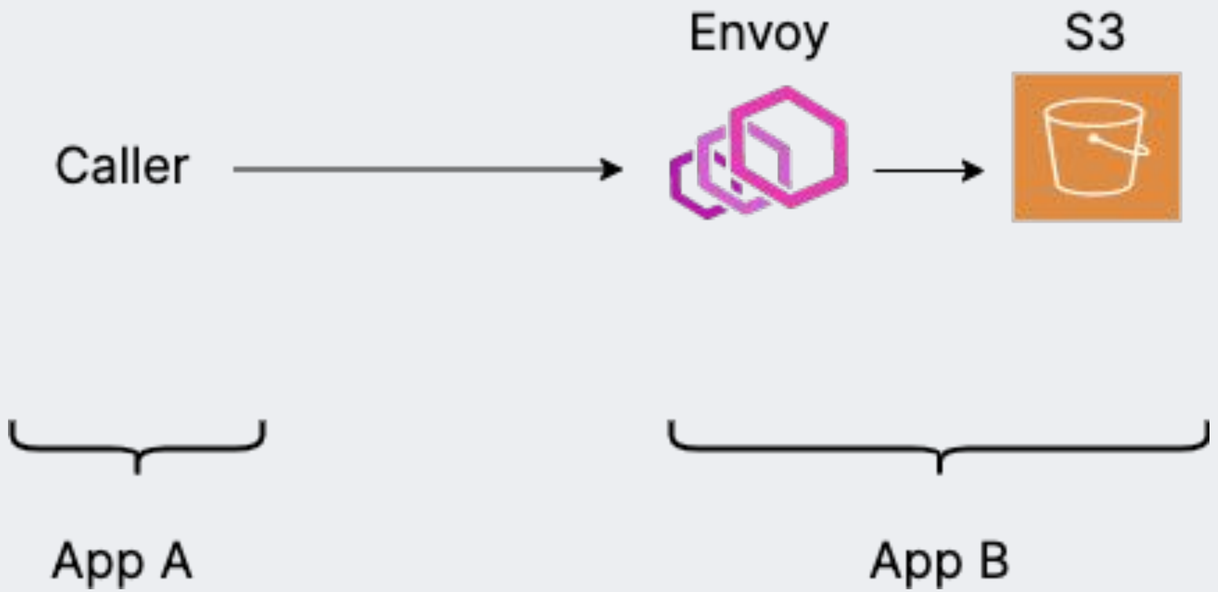
Configuring an Ingress Envoy

```
name: squareup.envoy.aws_irsa_signer
typed_config:
  "@type": type.googleapis.com/squareup.envoy.AwsIrsaSigner
  service_name: s3
  region: us-west-2
  backend_hostname: my-bucket.s3.us-west-2.amazonaws.com
  role_arn: arn:aws:iam::123456789012:role/mybucket-proxy
```

- We support S3 and API Gateway (for Lambda), but no service-specific logic in Envoy, makes it easily extensible
- **role_arn** field tells the extension what IAM role it should assume
- **backend_hostname** is the host we are proxying for - S3 bucket hostname or API Gateway url

Security Model

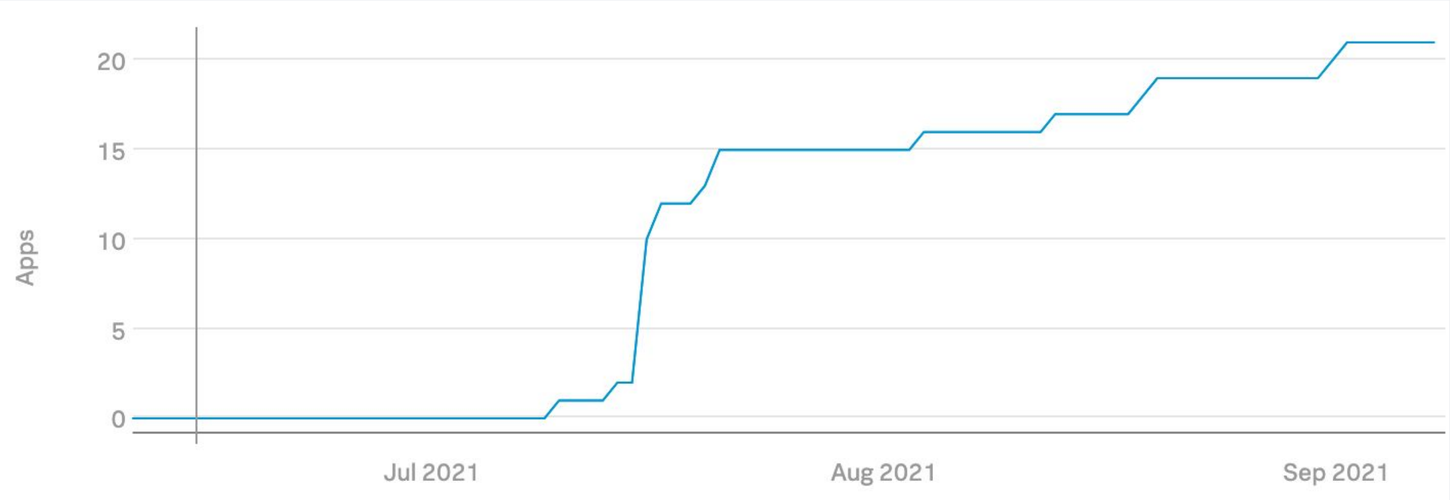
- Each backend (e.g. API Gateway **my-gateway**) gets its own ingress Envoy instance and IAM role
- Ingress Envoys run in the scope of the destination app. No central all-powerful proxy
- mTLS authn + authz happens first! Ingress Envoy effectively a signing oracle, but only to authorized callers
- Possible to use different session names for different callers for CloudTrail attribution, at the cost of credential caching efficiency



Making It Just Work™

- Terraform modules to create IAM roles for Envoy with the right permissions, setting up API Gateway to front Lambda
- Interactive script that prompts for inputs above and sets up Envoy pod correctly
- Setup page that walks users through steps needed. The setup page is just some static files in an S3 bucket, hosted on this infra too
- Portfolio of apps that teams built on this infra to give users a better sense for what is possible on the new platform

Where We’re At



- GA in late July, 20+ apps onboarded
- Used for admin UIs for app operators, dynamically generated HTML reports, documentation and wikis, standup Slackbots
- Each ingress Envoy costs $0.60 a day

Future Work

- Introduce support for route-based ACLs so we can expand to apps that expect this
- Per-route configuration support for the Envoy extension so multiple backends can share ingress Envoys (we use one ingress Envoy per backend today)
- Auto-scaling with traffic volume

Learnings

- SigV4 canonicalization is complicated! Some risk of implementation errors and feature gaps from Envoy not using the SDK. We found a couple of bugs that AWS fixed promptly

```
$ curl "localhost:10000/v1/?a=test&b=value"

{"statusCode": 200, "body": "{}"}
```

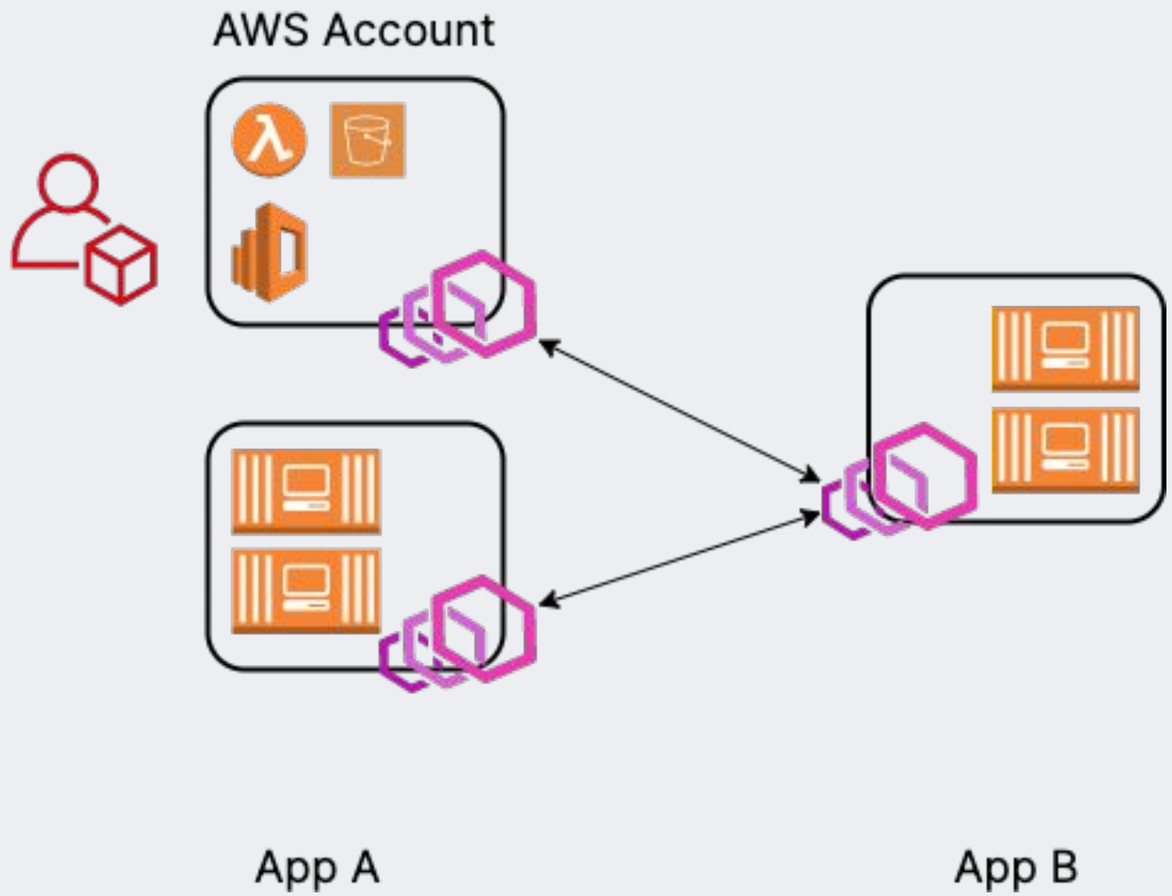
```
$ curl "localhost:10000/v1/?b=value&a=test"

{"message":"The request signature we calculated does not match the
signature you provided. Check your AWS Secret Access Key and signing
method. Consult the service documentation for details
```

Learnings

- Build smooth, paved roads and users will come
- We now have a secure, usable way for teams to make private, internal sites on S3

Learnings

- Reusing established patterns in your organization is powerful
- Ownership boundaries, logs, requesting ACL changes all work the way teams expect them to, because we're using the same service mesh



Questions?

Santosh Ananthakrishnan @santosh_ankr
Harihara K Narayanan @harihara89

Square Inc.