Proxyless gRPC

Sanjay Pujare, Google Cloud

LinkedIn: linkedin.com/in/sanjay-pujare-2157b2

Twitter: @PujareSanjay

IstioCon

Agenda

- What is gRPC?
- Proxyless gRPC in the Service Mesh
- Proxyless gRPC, xDS and Istio
- Traffic Management and Security with Proxyless gRPC
- How to use Proxyless gRPC
- What's coming: All language support, Agentless and Observability & More Resources

What is gRPC?

- A language independent glue for microservices
- Initially created by Google, as the next version of "Stubby"
  - "Stubby" was used to connect Google's large number of microservices
- De facto app level (L7) networking layer for connecting services
- Uses http2 as the underlying transport protocol,
  - Path identifies service/method
- Protobuf as the payload: request payload (method parameters) and response payload (method return value)
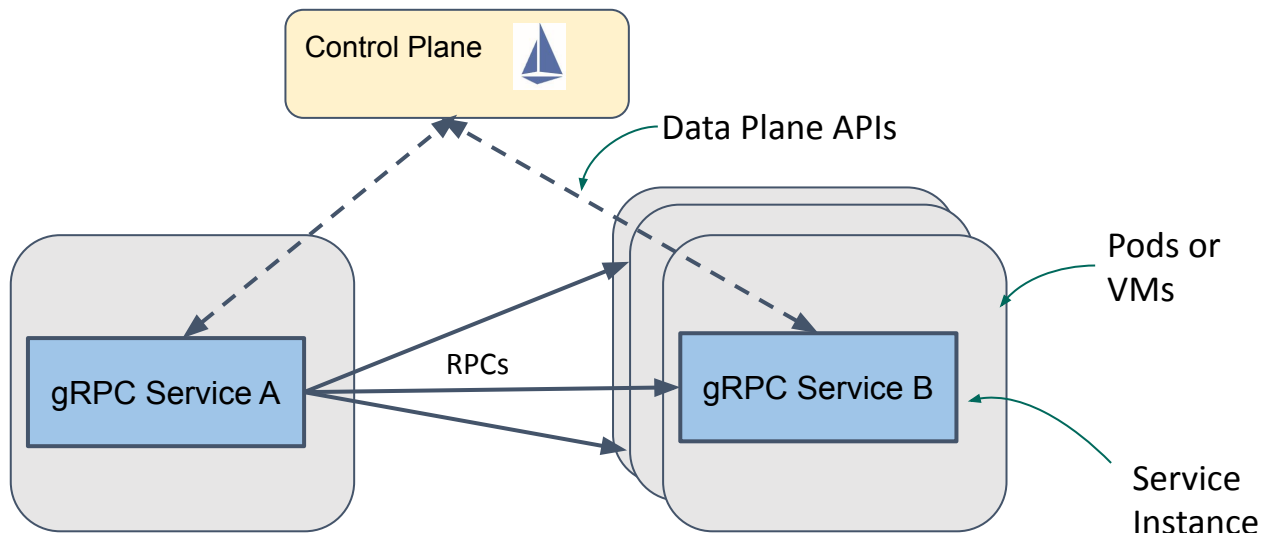
gRPC in the Service Mesh
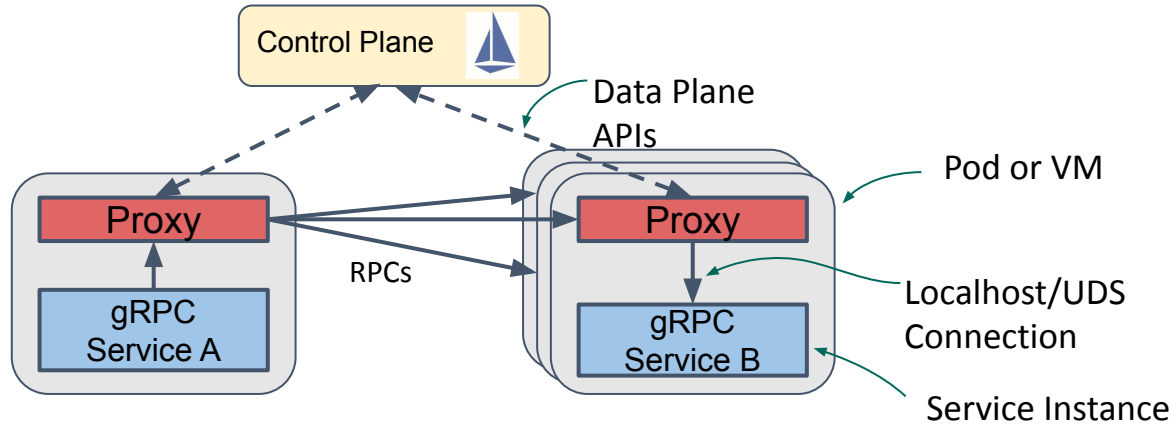
- Proxy used for service mesh policies
- gRPC sends requests to the virtual IP of the service
- Proxy intercepts requests, applies service mesh policies and sends out
- Server proxy receives request, applies policies and forwards to local service

Proxyless gRPC in the Mesh

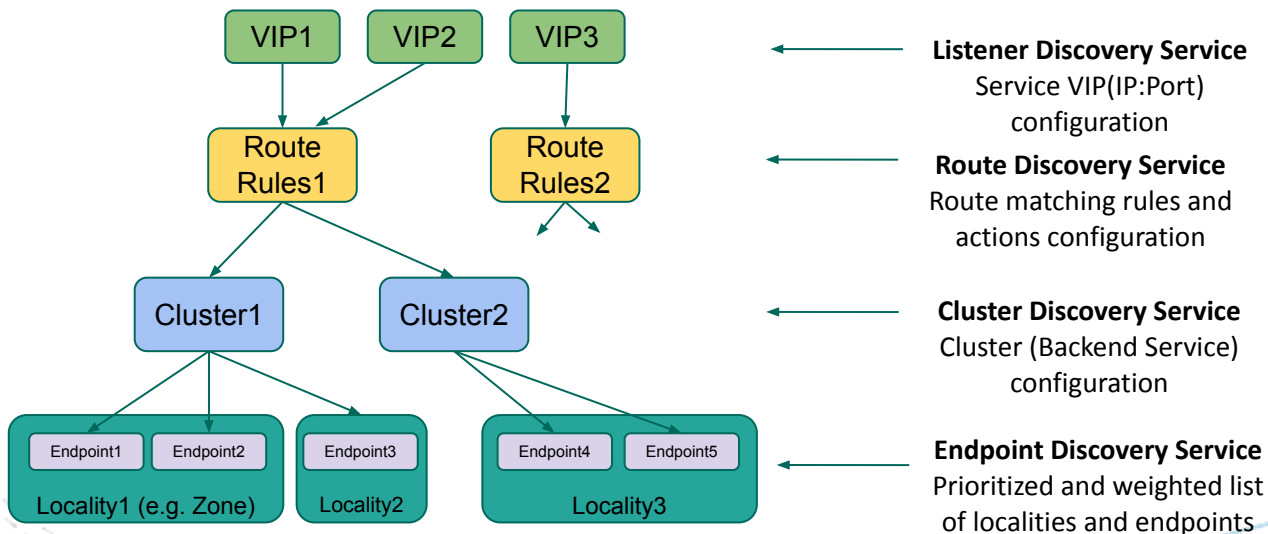Put proxy functionality in gRPC and eliminate proxy!

- gRPC client applies service mesh policies to outbound traffic
- gRPC server applies service mesh policies to incoming traffic
- Services talk to each other directly - no proxies!

Service Mesh With xDS

- xDS Data Plane APIs Developed for popular Envoy proxy
- Open, Extensible & Strong Community Support
- Right choice for gRPC's Service Mesh implementation!

Listener Discovery Service
Service VIP(IP:Port) configuration

Route Discovery Service
Route matching rules and actions configuration

Cluster Discovery Service
Cluster (Backend Service) configuration

Endpoint Discovery Service
Prioritized and weighted list of localities and endpoints

(x)Discovery Service:

- Listener
- Route
- Cluster
- Endpoint etc

Using Proxyless gRPC with Istio

Use "`Xds`"-Channel and Server credentials.

Java example from A29-xds-tls-security.md#java

`XdsChannelCredentials` on the client (channel):

```java
    ChannelCredentials credentials

            = XdsChannelCredentials.create(InsecureChannelCredentials.create());

    ManagedChannel channel = Grpc.newChannelBuilder(target, credentials).build();
```

`XdsServerCredentials` passed to `XdsServerBuilder`:

```java
    ServerCredentials credentials

        = XdsServerCredentials.create(InsecureServerCredentials.create());

    Server server = XdsServerBuilder.forPort(port, credentials)

            .addService(new HostnameGreeter(hostname)).build().start();
```

Proxyless gRPC with Security
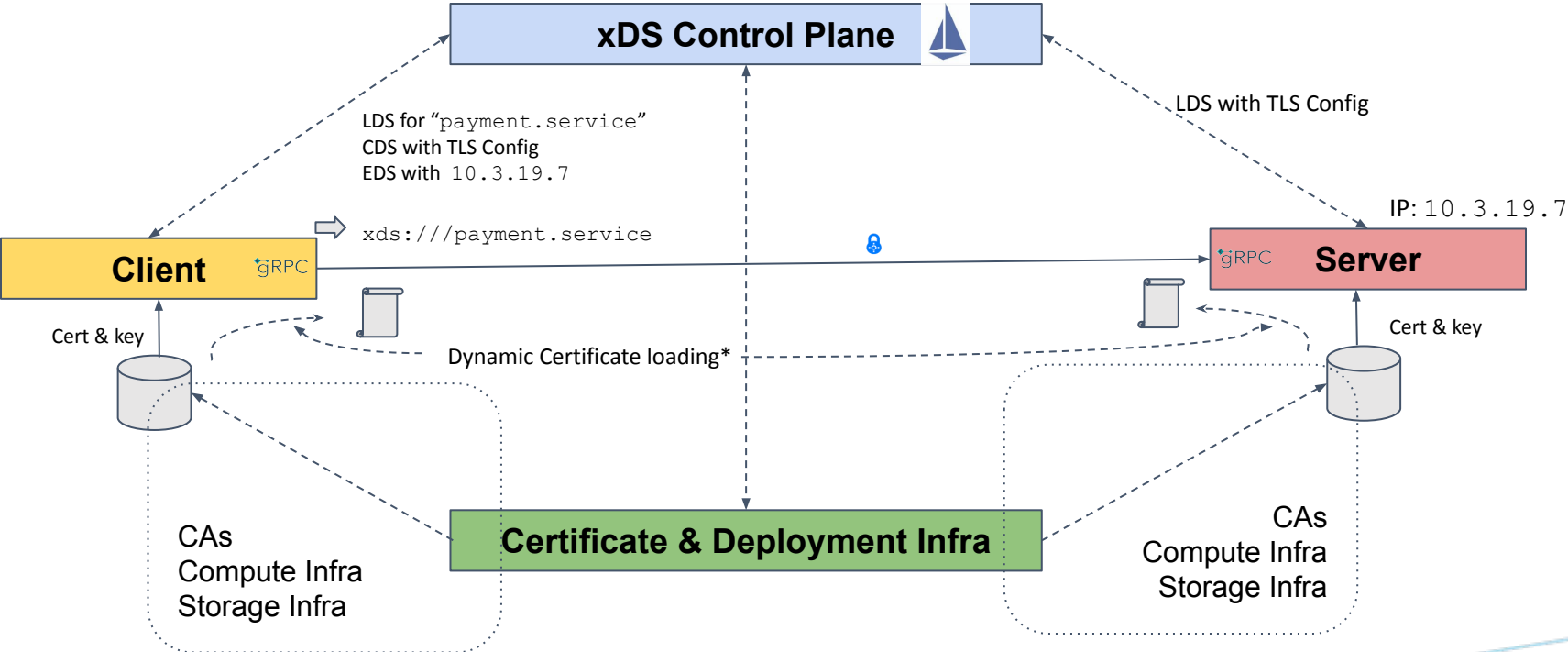
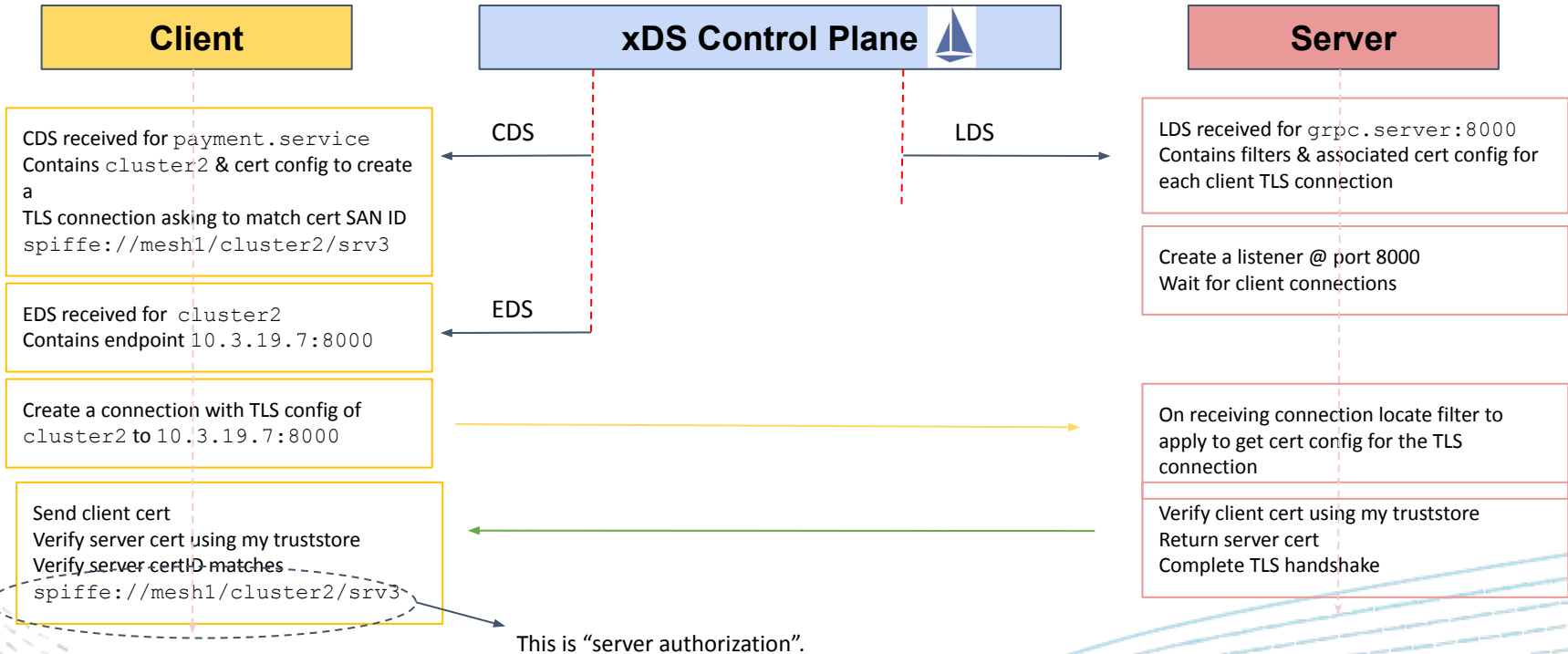* Certificates are dynamically updated and are reflected on both client and server

xDS Sequence and mTLS Sequence

IP: `10.3.19.7`
CertID: `spiffe://mesh1/cluster2/srv3`

Client	xDS Control Plane	Server

CDS

CDS received for `payment.service`
Contains `cluster2` & cert config to create a
TLS connection asking to match cert SAN ID
`spiffe://mesh1/cluster2/srv3`

LDS

LDS received for `grpc.server:8000`
Contains filters & associated cert config for each client TLS connection

Create a listener @ port 8000
Wait for client connections

EDS

EDS received for `cluster2`
Contains endpoint `10.3.19.7:8000`

Create a connection with TLS config of `cluster2` to `10.3.19.7:8000`

On receiving connection locate filter to apply to get cert config for the TLS connection

Send client cert
Verify server cert using my truststore
Verify server certID matches
`spiffe://mesh1/cluster2/srv3`

Verify client cert using my truststore
Return server cert
Complete TLS handshake

This is "server authorization".

#IstioCon

Latency Gains with Proxyless

- Performance gains (latency)
  - Almost 10 to 20 times latency gain especially as # of connections go up

Source: https://istio.io/latest/blog/2021/proxyless-grpc/#latency
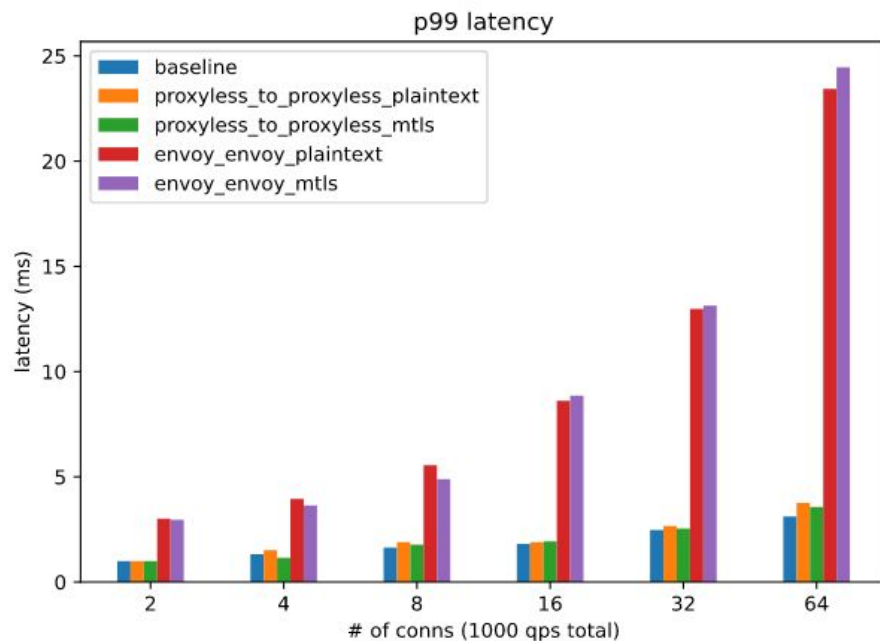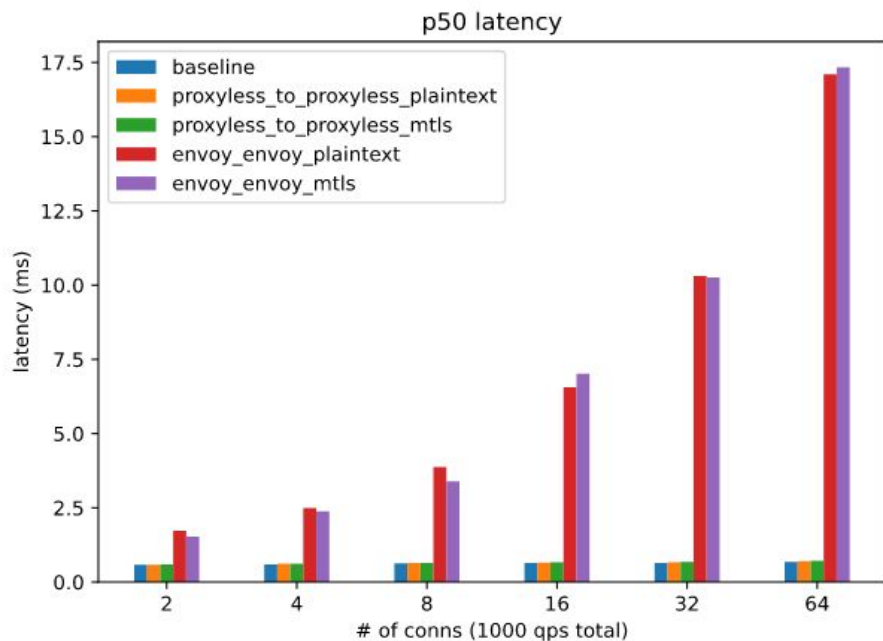
Resource Usage with Proxyless

- Resource Usage
  - Proxyless uses sidecar container for the istio-agent+xds-proxy and Envoy uses the sidecar container for Envoy+istio-agent+xds-proxy
- For Proxyless the sidecar use of the vCPU is less than 1% that of the Envoy case and for memory less than half of what running Envoy requires.

	Client mCPU	Client Memory (MiB)	Server mCPU	Server Memory (MiB)
Envoy Plaintext	320.44	66.93	243.78	64.91
Envoy mTLS	340.87	66.76	309.82	64.82
Proxyless Plaintext	0.72	23.54	0.84	24.31
Proxyless mTLS	0.73	25.05	0.78	25.43

Source: https://istio.io/latest/blog/2021/proxyless-grpc/#istio-proxy-container-resource-usage

What's Coming...?

- Full Language Support
  - gRPC-Go currently supported
  - gRPC-C++, Python being added ...
  - gRPC-Java coming soon...
  - Other gRPC Languages PHP, C# coming soon...
- Observability Support for Istio/ASM
  - Metrics, Logs and Distributed Tracing
  - Observability API for Workload Selection and Provider Selection
  - Metadata Exchange or Peer Metadata for Fine Grained Telemetry
  - Visualizing your mesh: Service Graph visualization

What's Coming...?

Fully Agentless Architecture:
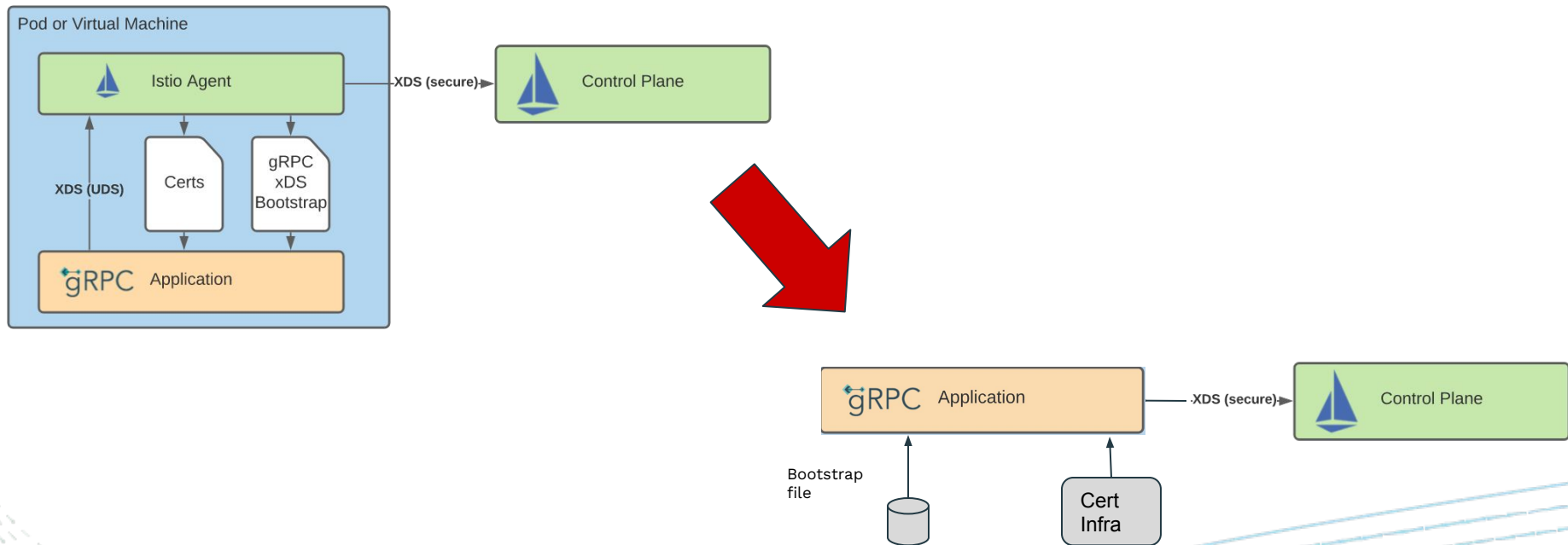
Resources

- gRFC [A27: xDS-Based Global Load Balancing](#)
- gRFC [A36: xDS-Enabled Servers](#)
- gRFC [A29: xDS-Based Security for gRPC](#)
- gRFC [L74: Java Channel and Server Credentials](#)
- gRFC [A41:xDS RBAC Support](#)
- Istio Blog: [gRPC Proxyless Service Mesh](#)
- Google Blog: [Traffic Director and Proxyless gRPC](#)

Thank you!

@PujareSanjay

https://linkedin.com/in/sanjay-pujare-2157b2