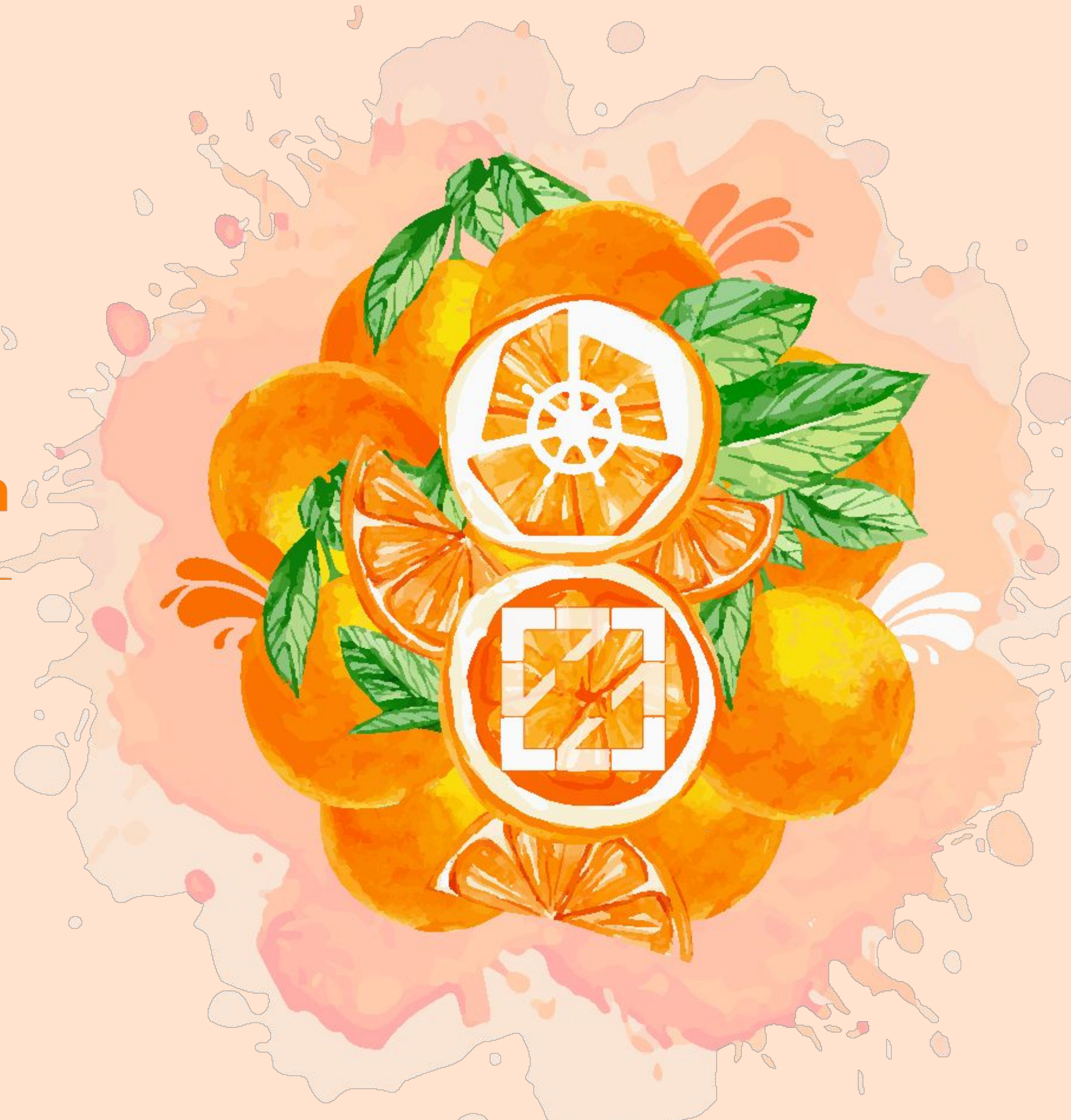
KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA



KubeCon | CloudNativeCon
Europe 2022

gRPC for Microservices: Service Mesh & Observability

Sanjay Pujare, gRPC Engineering, Google

Agenda

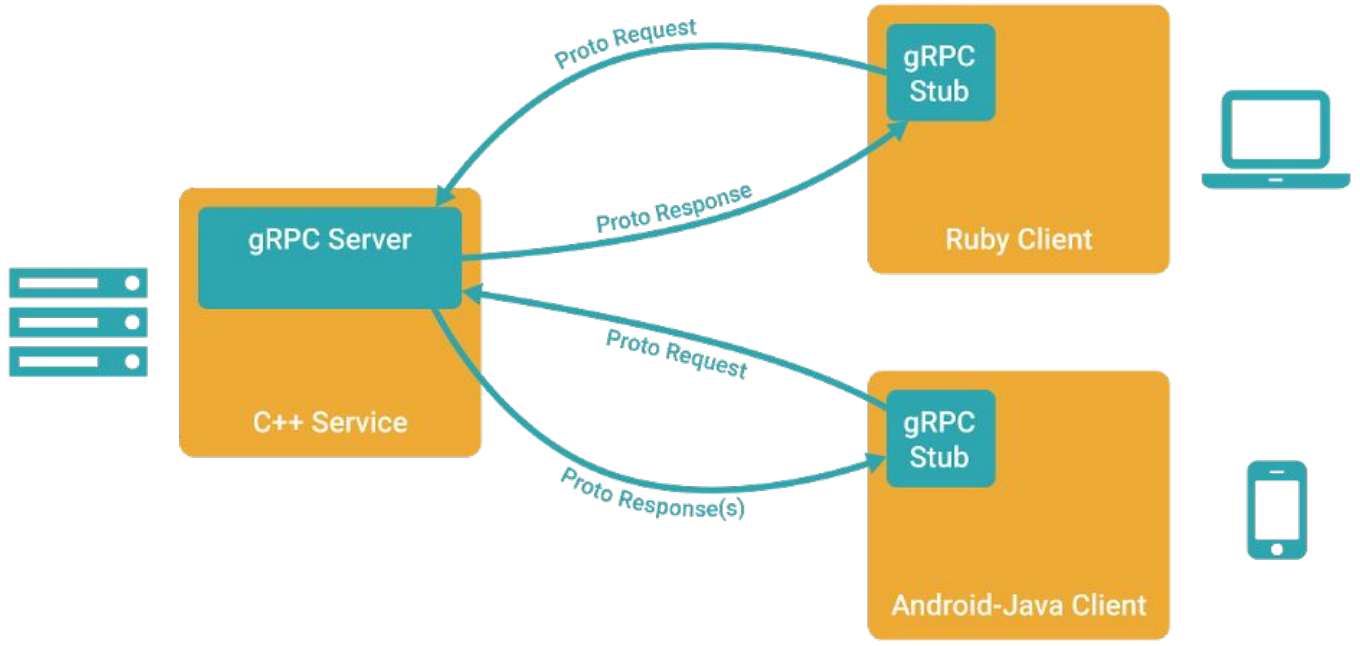
- Brief Intro to gRPC
- Proxyless gRPC in the Service Mesh
- Service Mesh: Traffic Management
  - Service Discovery, Routing, Load Balancing …
- Service Mesh: Security
  - mTLS and Authorization
- Use of xDS as a Vendor Agnostic Open Protocol
- How to Use Proxyless gRPC in the Service Mesh
- Observability in gRPC: What’s Happening?
- Q & A

Intro to gRPC

What is gRPC?

- Language & platform independent glue for microservices
- Created by Google, as next version of *Stubby*
  - *Stubby* connected large number of microservices at Google scale: O(10^{10}) RPCs per second
- Uses http2 and benefits from binary framing, multiplexing, streaming and HPACK compression
- Generally used with Protobuf for payload serialization

Intro to gRPC



- Use Protobuf IDL in .proto
- Generate server & client stubs using protoc compiler
- Extend server stub to add server logic
- Use client stub to invoke methods

Note: protobuf is not mandatory to use gRPC!

There are other integrations like google/flatbuffers and Microsoft/bond

Intro to gRPC: Protobuf

```
syntax = “proto3”;

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber phone = 4;
}
```

What’s Protobuf aka Protocol Buffers?

- Google’s *Lingua Franca* for serializing data: on the network and in storage
- Strongly typed
- Binary format
- Extensibility and backward compatibility
- Code generators for Java, C++, Go and many other languages

To reiterate: *protobuf is not mandatory to use gRPC!*

But very convenient and optimized.

Intro to gRPC: In a nutshell...

Multi-language	On every platform	Strict Service contracts
Performant & Efficient on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Intro to gRPC: Wrap-up

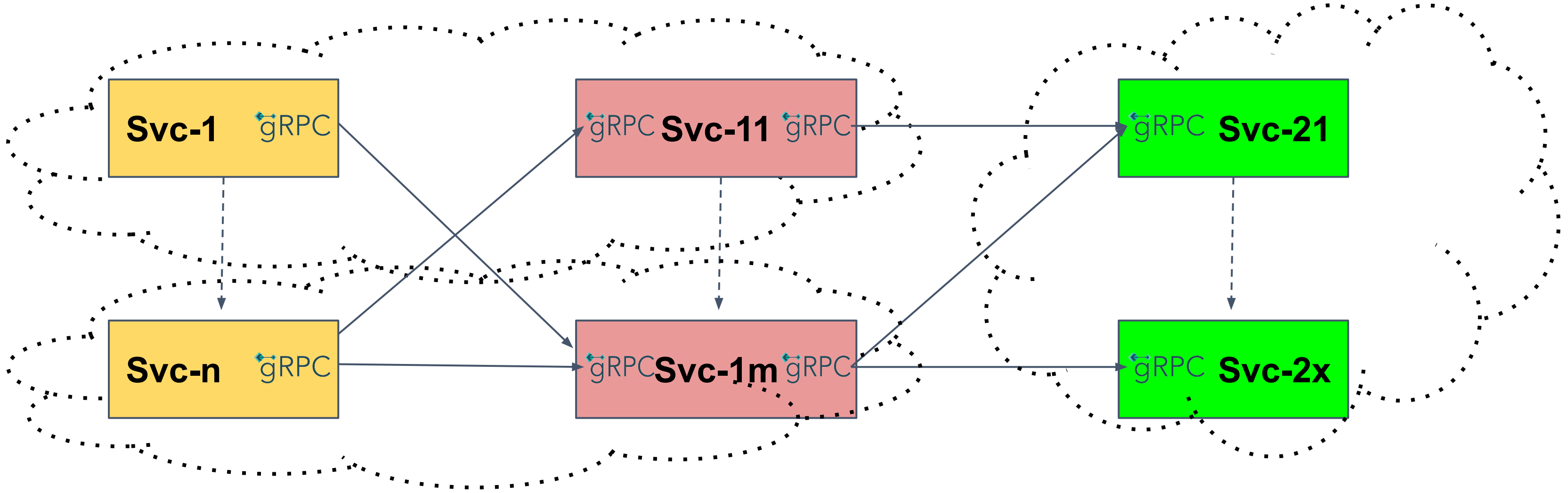
More Info/Resources:

- Intro to gRPC in KubeCon Europe2020
- gRPC http://grpc.io
- https://github.com/grpc-ecosystem
- Gitter Channel : https://gitter.im/grpc/grpc
- Twitter: @grpcio
- Mailing List : grpc-io@googlegroups.com

gRPC in Microservices & Service Mesh

New paradigm: convert a monolithic application into a mesh of microservices

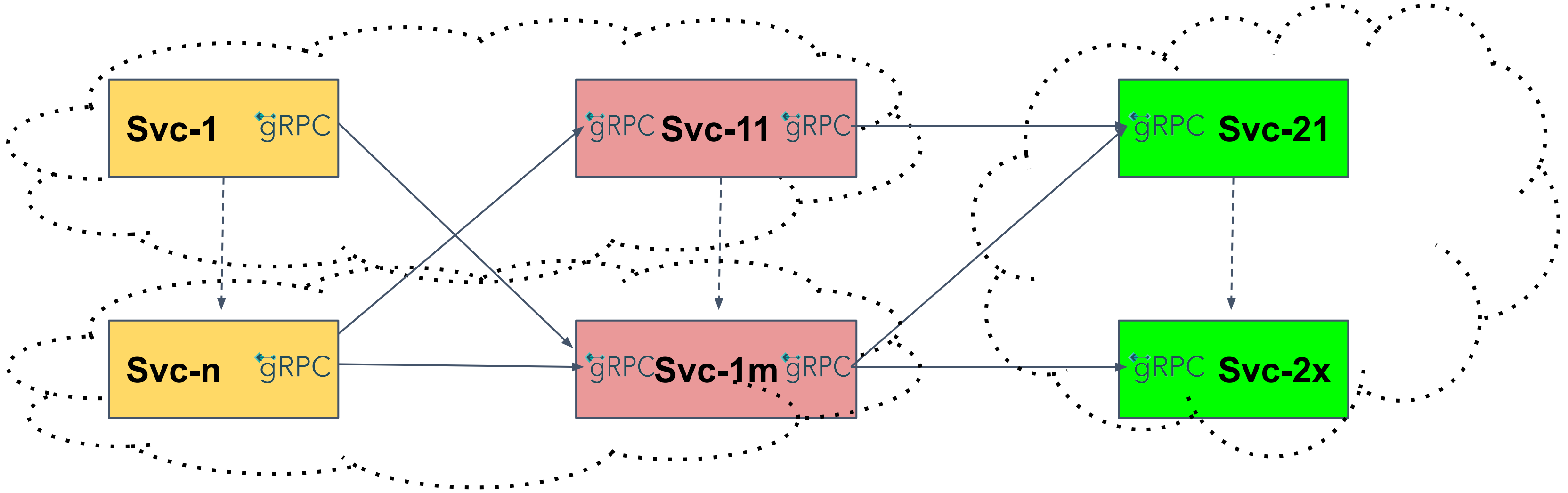
In-process calls become gRPC calls between microservices over the network



gRPC in Microservices & Service Mesh

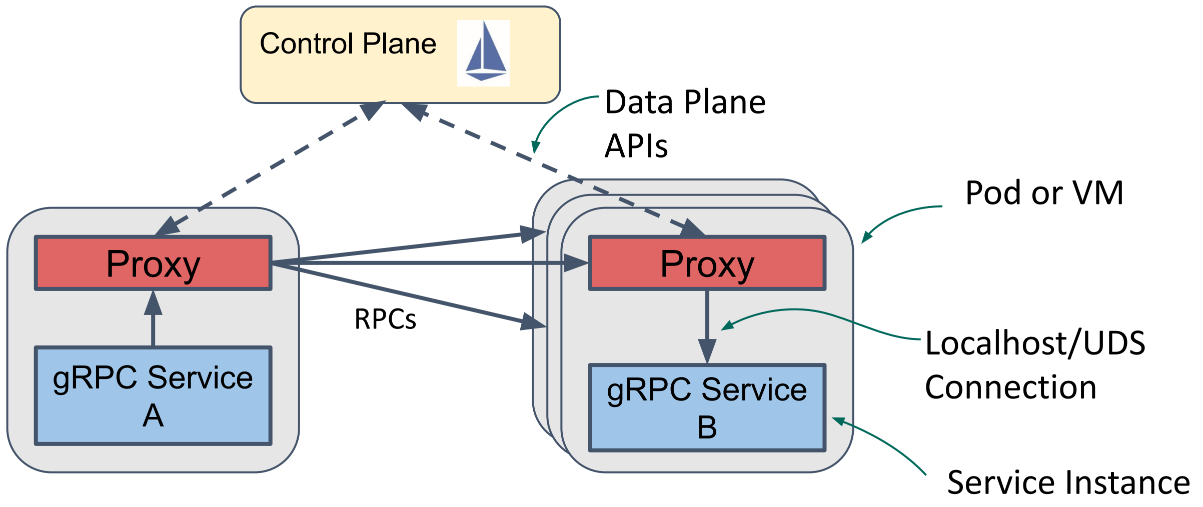
Scaling involves new VMs/clusters/networks and RPCs crossing the boundaries

Need to manage traffic and security now!



gRPC & Service Mesh

Service Mesh to the Rescue!



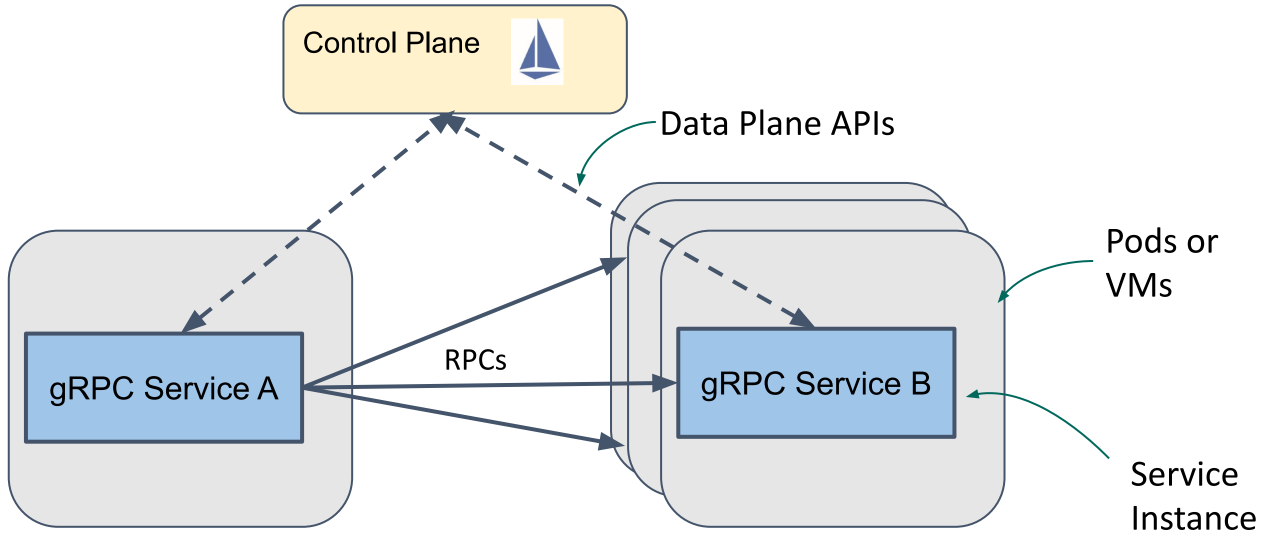
So What's a Service Mesh?

- Policies for Traffic Management and Security
- Control Plane (e.g. Istio) stores and manages policies
- In the proxy mode transparent proxies enforce/implement policies
- gRPC sends requests to the virtual IP of the service
- Proxy intercepts requests, applies service mesh policies and sends them out
- Server proxy receives requests, applies policies and forwards to local service instance

But with gRPC, there is the proxyless mode!

gRPC & Proxyless Service Mesh

Proxy eliminated by gRPC!



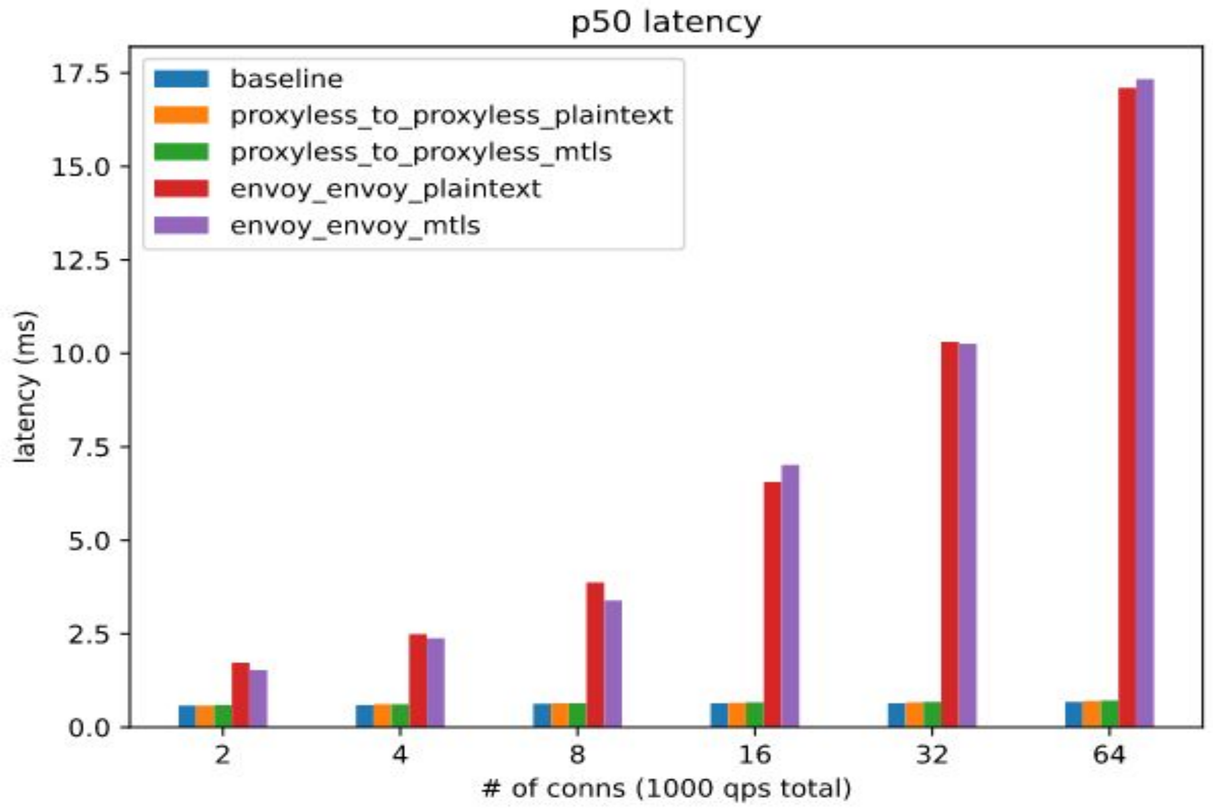
What's the Proxyless Service Mesh?

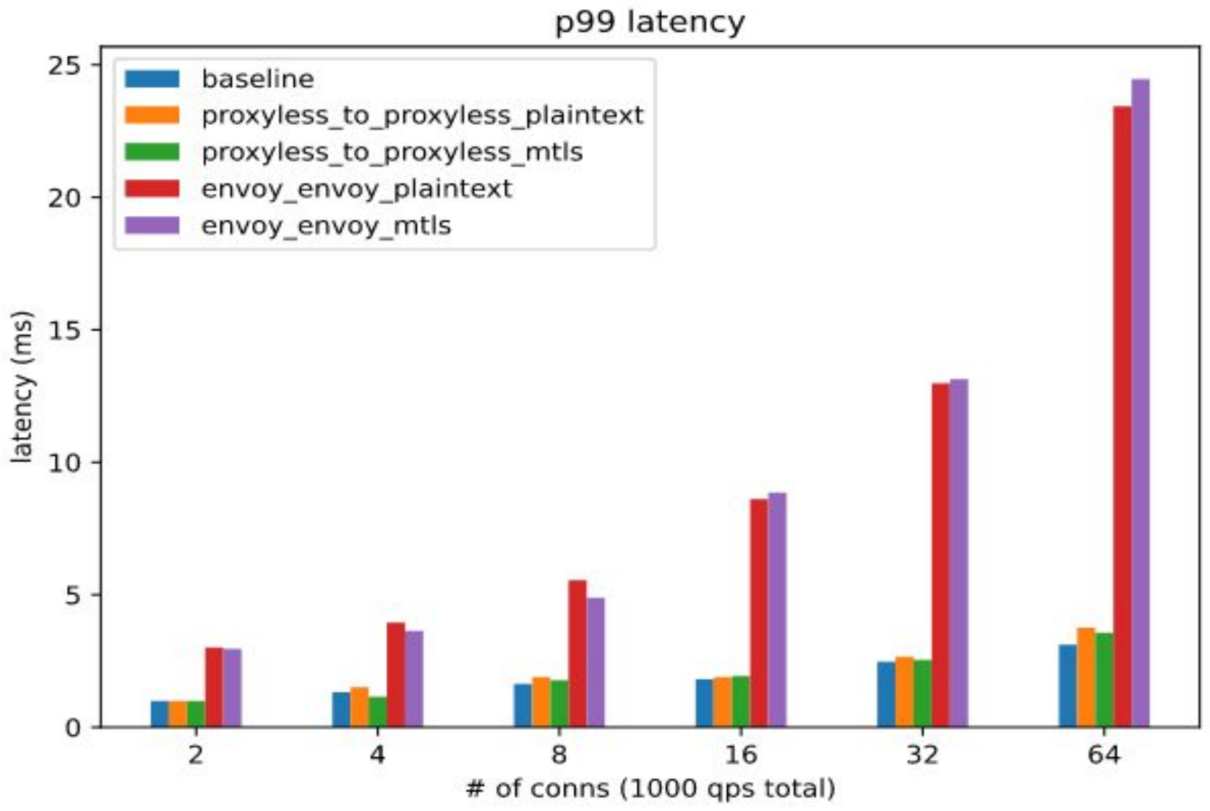
- The same control plane (e.g. Istio) stores and manages policies
- gRPC client applies service mesh policies to the outbound traffic
- gRPC server applies service mesh policies to incoming traffic
- Microservices talk to each other directly - no proxies!

Proxies eliminated for gRPC traffic - might still be needed for other traffic.

gRPC & Proxyless Service Mesh

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

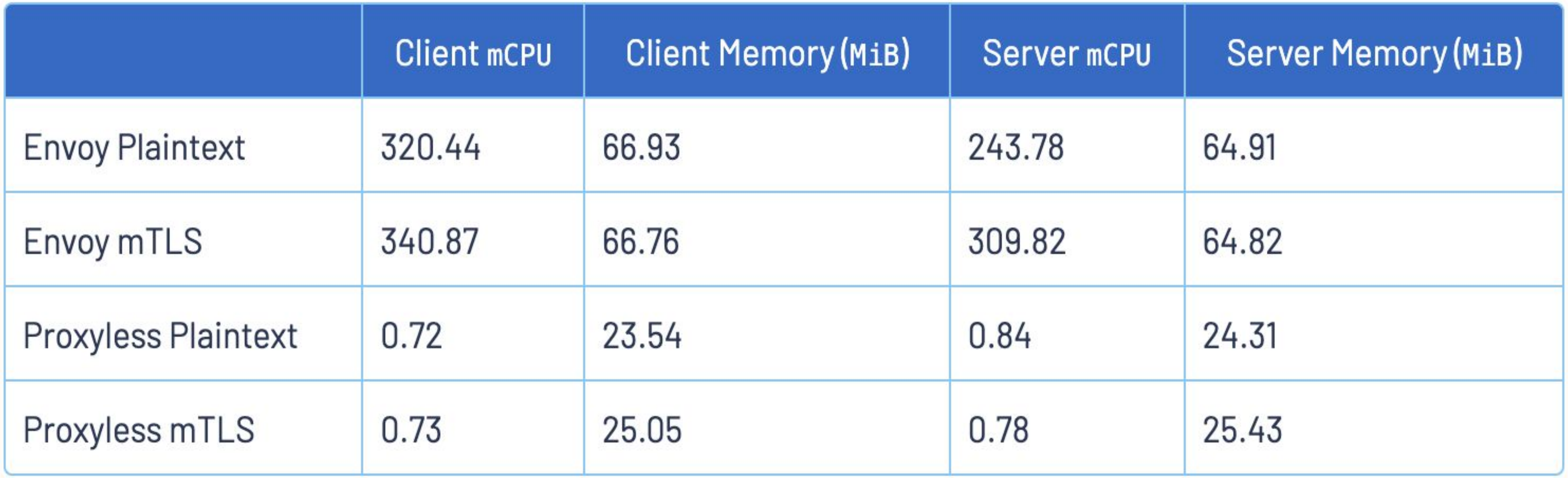




Source: https://istio.io/latest/blog/2021/proxyless-grpc/#latency

gRPC & Proxyless Service Mesh

- 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.

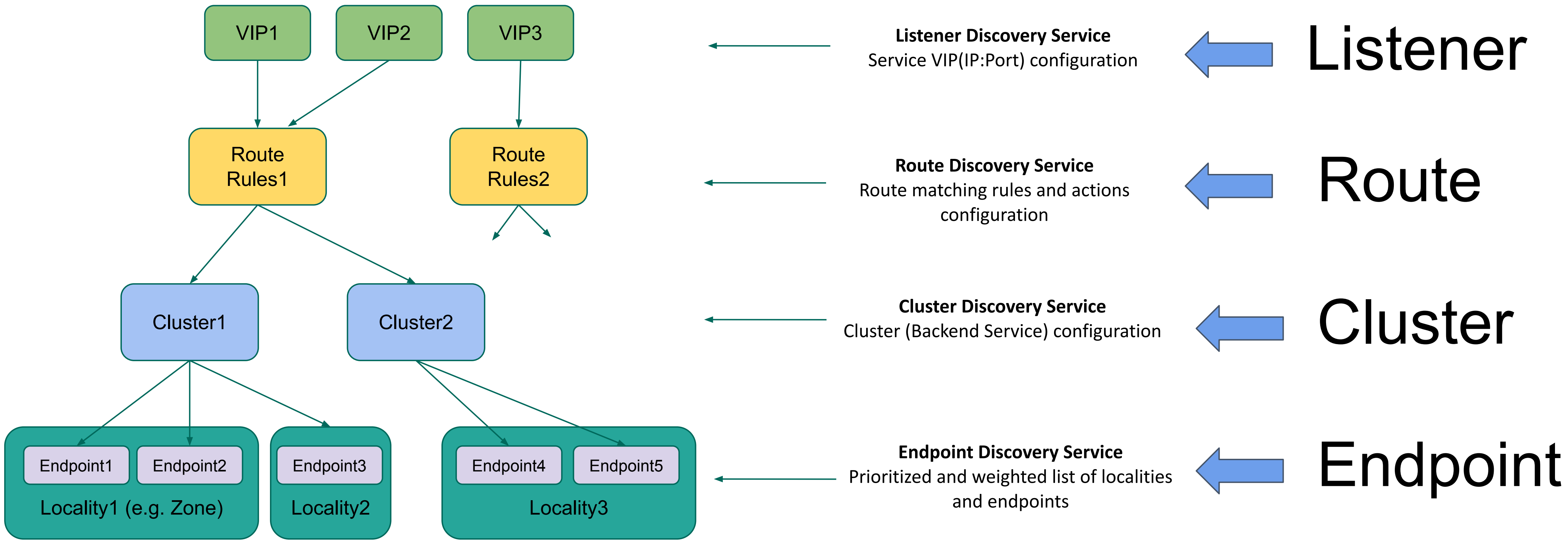


	Client mCPU	Client Memory (MiB)	Server mCPU	Server Memory (MiB)
Envoy Plaintext	320.44	66.93	243.78	64.91
Envoy mTLS	340.87	66.76	309.82	64.82
Proxyless Plaintext	0.72	23.54	0.84	24.31
Proxyless mTLS	0.73	25.05	0.78	25.43

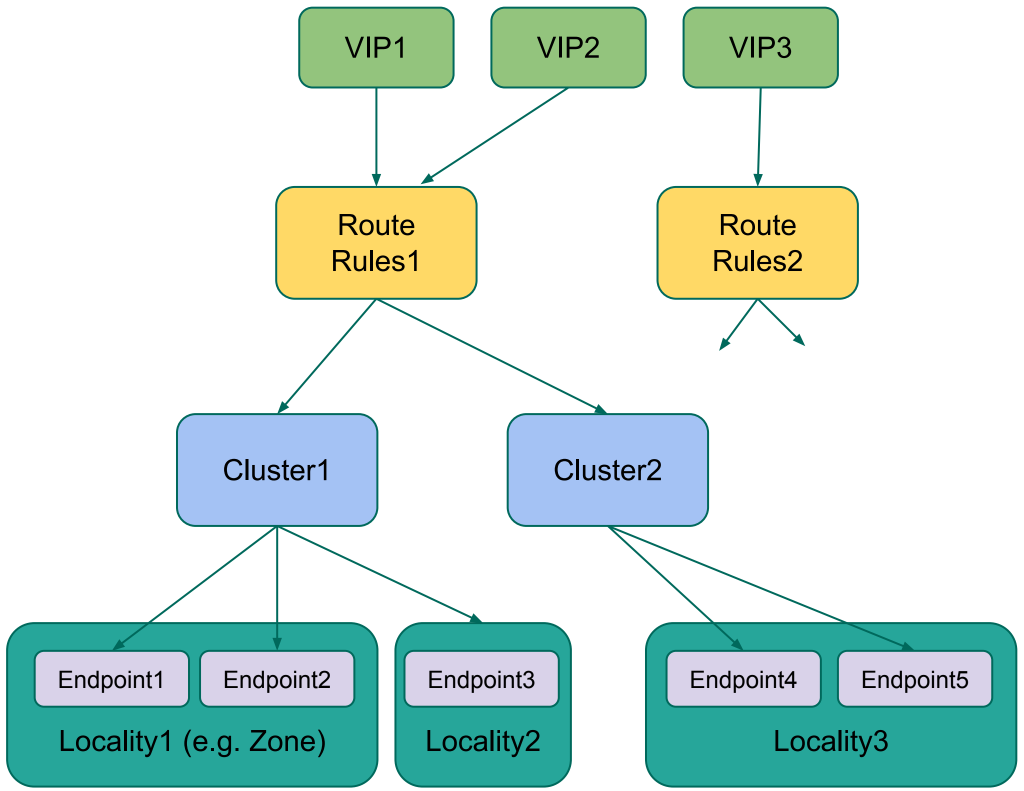
Source: https://istio.io/latest/blog/2021/proxyless-grpc/#istio-proxy-container-resource-usage

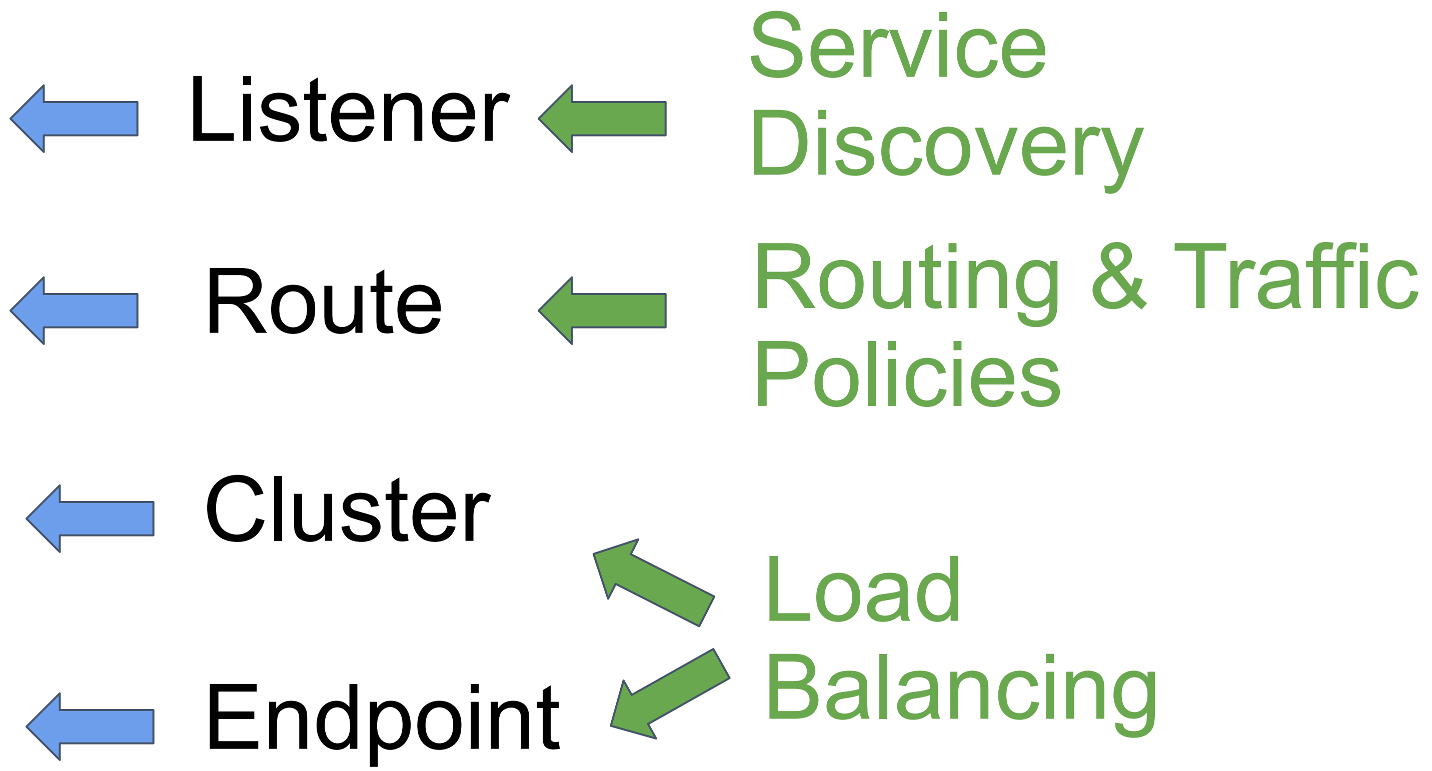
Service Mesh With xDS

- xDS Data Plane APIs from Envoy: Open & Extensible
- Right choice for gRPC Service Mesh!


VIP1
VIP2
VIP3
Route Rules1
Route Rules2
Cluster1
Cluster2
Endpoint1
Endpoint2
Endpoint3
Endpoint4
Endpoint5
Locality1 (e.g. Zone)
Locality2
Locality3
Listener Discovery Service
Service VIP(IP:Port) configuration
Listener
Route Discovery Service
Route matching rules and actions configuration
Route
Cluster Discovery Service
Cluster (Backend Service) configuration
Cluster
Endpoint Discovery Service
Prioritized and weighted list of localities and endpoints
Endpoint

Traffic Management in the Service Mesh





Security in the Service Mesh

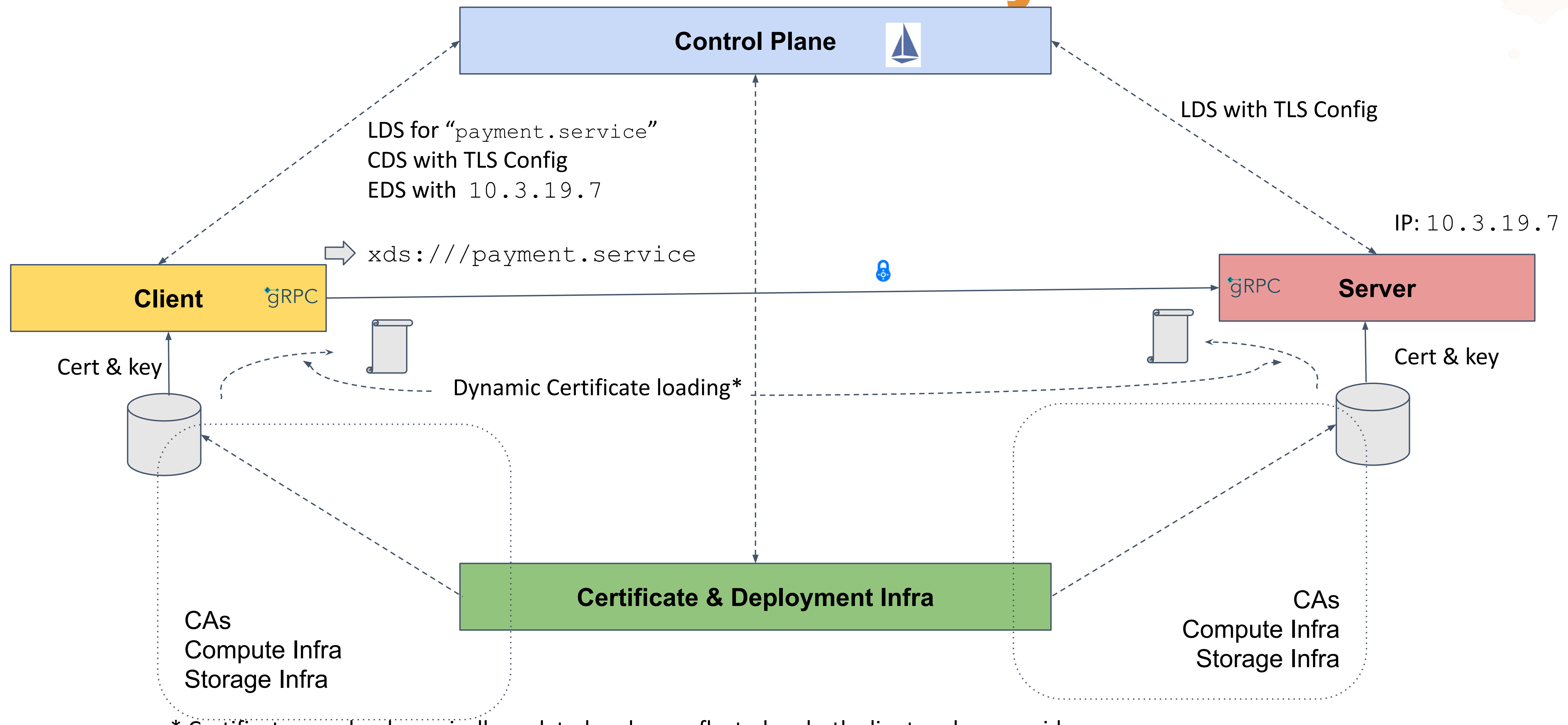
Why is Security So Important?

- Remember the paradigm shift of breaking a monolith into microservices?
- In-process calls are now gRPC calls between microservices over the network
- This network traffic needs to be authenticated, encrypted and authorized

And who is going to do that?

gRPC for Microservices: Service-mesh & Observability

Service Mesh with Security



* Certificates can be dynamically updated and are reflected on both client and server side

Service Mesh with Security

- Security Infrastructure provides certificates and keys
- Control plane configures mTLS in CDS (client side) or LDS (server side)
- gRPC uses provided certs and transport_socket configuration to create mTLS connections
- mTLS gives you encryption + authentication + server authorization
- Server uses “authorization policy” aka RBAC to authorize RPCs based on client identities

Using gRPC in Proxyless Service Mesh

Java example from [A29-xds-tls-security.md#java]()

`XdsChannelCredentials` on the channel (client side):

```
    ChannelCredentials credentials
              = XdsChannelCredentials.create(InsecureChannelCredentials.create());
    ManagedChannel channel = Grpc.newChannelBuilder(target, credentials).build();
```

`XdsServerCredentials` on the server side:

```
    ServerCredentials credentials
          = XdsServerCredentials.create(InsecureServerCredentials.create());
    Server server = XdsServerBuilder.forPort(port, credentials)
          .addService(new HostnameGreeter(hostname)).build().start();
```

Fallback Credentials

gRPC in Service Mesh: Wrap-up

Info/Resources:

- KubeCon presentation: Service Mesh With GRPC And xDS in May’21 - Video recording
- A27: xDS-Based Global Load Balancing
- Istio Blog: gRPC Proxyless Service Mesh
- KubeCon presentation: GRPC Proxyless Service Mesh With Security in Oct’21 - Video recording

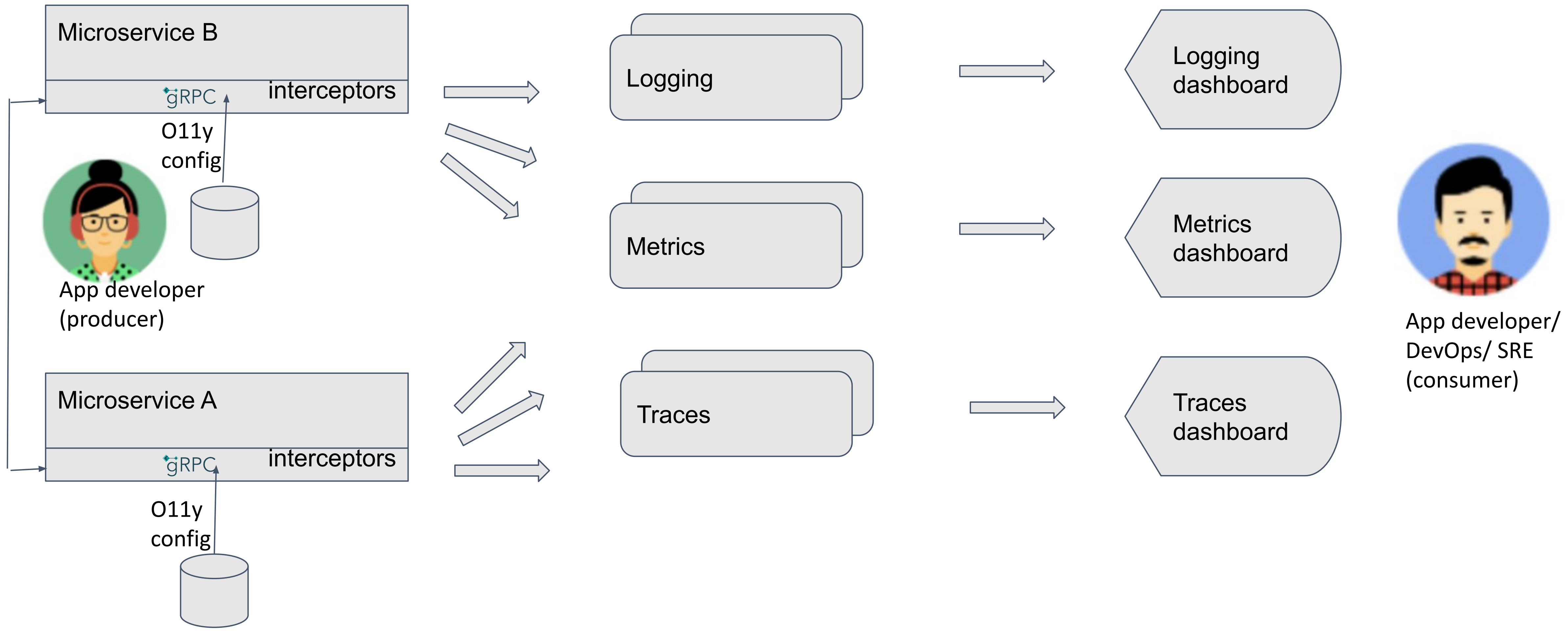
gRPC Microservices & Observability

- Using gRPC, a monolith now split into microservices spread over diverse infrastructure
- Behavior of the “system” now dependent on individual microservices, network, compute & other infrastructure
- If an issue arises, how can we debug and fix it? Needed to increase reliability and efficiency of this new paradigm
- We need an “observable” system where internal state is visible or can be inferred
- Can gRPC provide this “observability” into your microservices?

gRPC Observability

- 3 traditional pillars of observability: logs, metrics and traces
- Use gRPC's "interceptor" framework to collect the raw data for the 3 pillars
- Integrate with exporters and analytics backends to provide end-to-end observability

gRPC Observability aka O11y



gRPC Observability in GCP

- gRPC library enhanced with necessary plugins for logging, metrics & traces
- Raw data exported via Stackdriver exporters to Google CloudOps backend
- Admin console to enable/administer feature
- Consumer dashboard to visualize

gRPC Observability with Java

- grpc-gcp-observability artifact to be added to your application build
- grpc-gcp-observability pulls in other required dependencies e.g. Stackdriver exporter
- Call observability init() from the app
- You get observability when application run in Google Cloud and with appropriate config

gRPC Observability with Java

```
...
import io.grpc.gcp.observability.GcpObservability;
...

// Main application class
...

public static void main(String[] args) {
...
   // call GcpObservability.grpcInit() to initialize & get observability
   try (GcpObservability observability = GcpObservability.grpcInit()) {

...
   } // observability.close() called implicitly
...
}
```

`grpcInit()` called by app before gRPC channels/servers creation. Reads Observability configuration and sets up channel/server providers

`close()` deallocates resources & removes special channel/server providers

gRPC Observability GCP Config

```
{
    "enable_cloud_logging": true,
    "enable_cloud_monitoring": true,
    "enable_cloud_trace": true,
    "destination_project_id": "your-project-here",
    "log_filters": [{
        "pattern": "*",
        "header_bytes": 4096,
        "message_bytes": 4096
    }],
   "event_types": [
        "GRPC_CALL_REQUEST_HEADER",
        "GRPC_CALL_RESPONSE_HEADER",
        "GRPC_CALL_TRAILER"
    ],
   "global_trace_sampling_rate": 0.5
}
```

Each pillar can be enabled/disabled separately

Allows cross-project logging

Filters to selectively log only certain RPCs

Filters to selectively log only certain events in an RPC

Probabilistic sampler specifying probability of 0 to 1

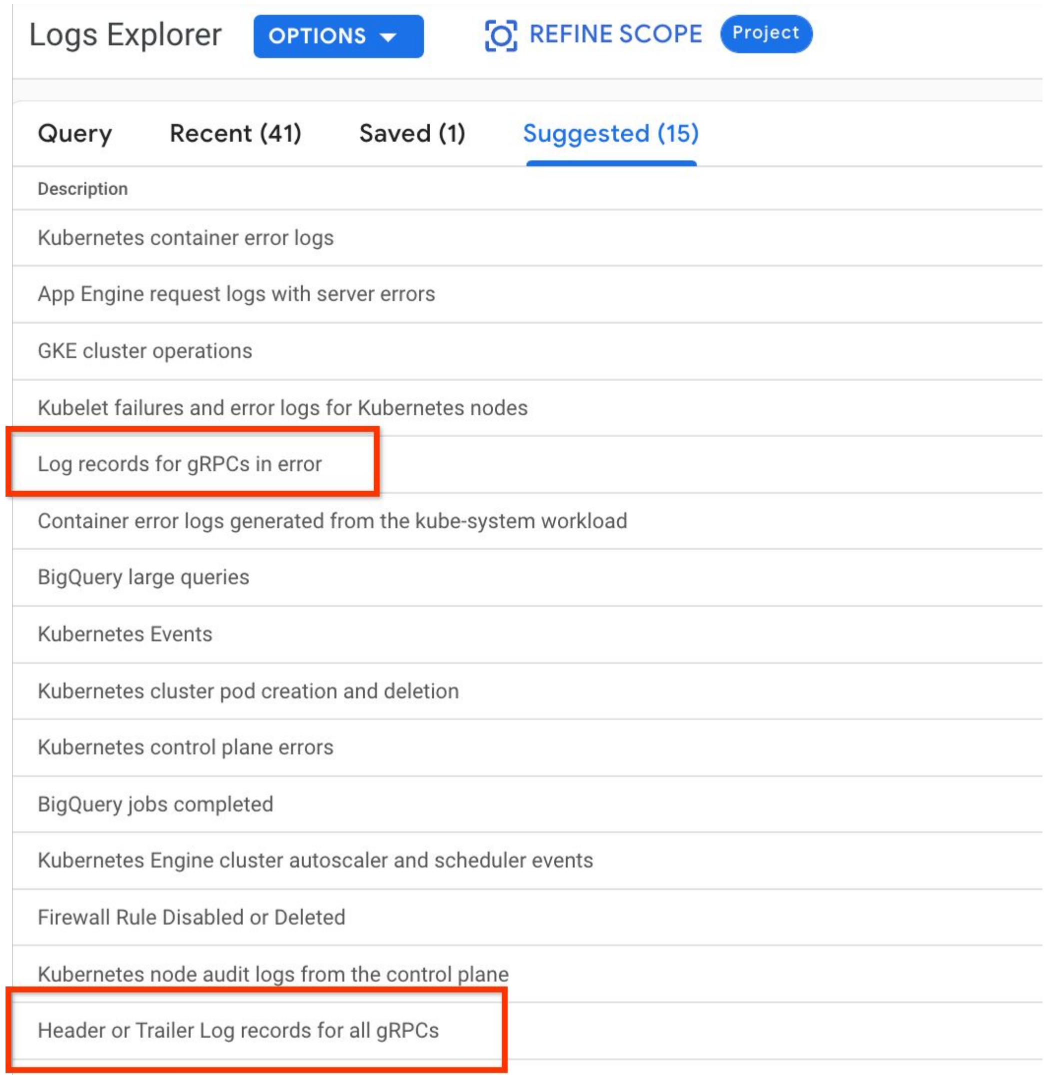
gRPC Observability in GCP

- Location tags automatically added: deployment-specific key-value pairs, such as location of workload. e.g. GCE node name, GKE namespace
- Custom tags based on user config: e.g.

```
GRPC_OBSERVABILITY_DATACENTER=SAN_JOSE_DC
GRPC_OBSERVABILITY_APP_ID=24512
```

Environment variables with prefix "GRPC_OBSERVABILITY_" specify custom tags

gRPC Observability in GCP



Description

Kubernetes container error logs

App Engine request logs with server errors

GKE cluster operations

Kubelet failures and error logs for Kubernetes nodes

Log records for gRPCs in error

Container error logs generated from the kube-system workload

BigQuery large queries

Kubernetes Events

Kubernetes cluster pod creation and deletion

Kubernetes control plane errors

BigQuery jobs completed

Kubernetes Engine cluster autoscaler and scheduler events

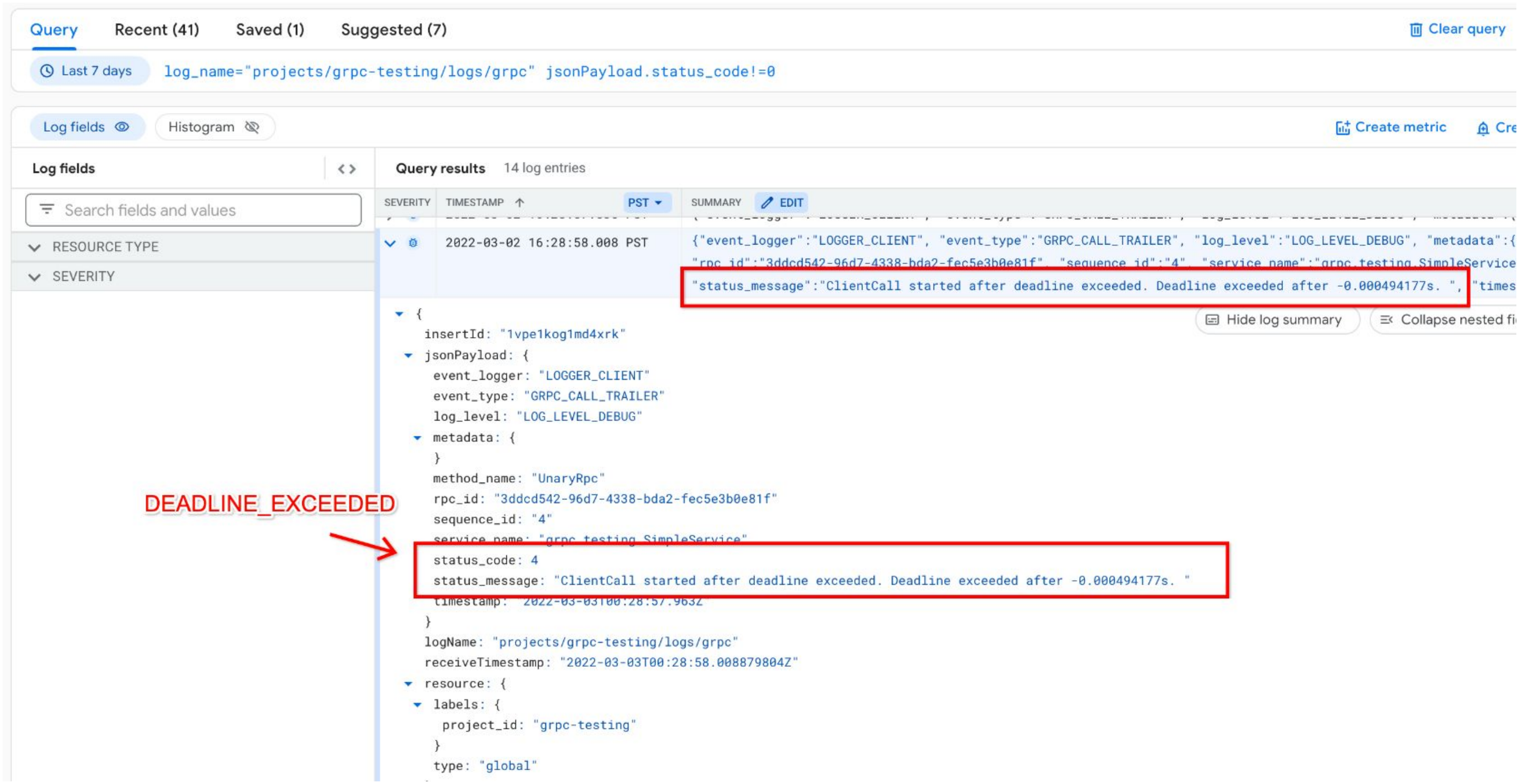
Firewall Rule Disabled or Deleted

Kubernetes node audit logs from the control plane

Header or Trailer Log records for all gRPCs

Logging available in private preview

gRPC Observability Logging Preview

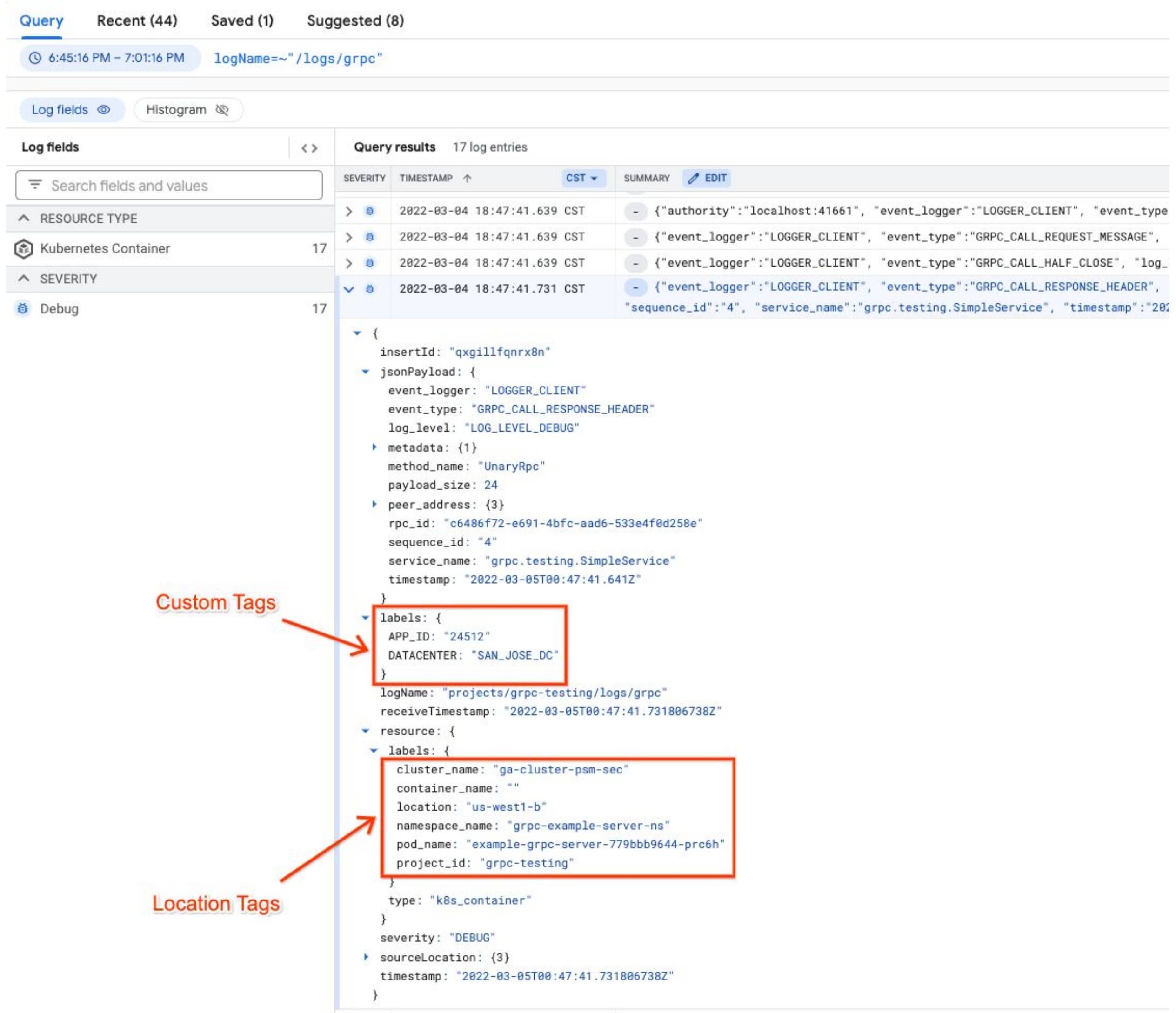


```
log_name="projects/grpc-testing/logs/grpc" jsonPayload.status_code!=0
```

```
2022-03-02 16:28:58.008 PST  {"event_logger":"LOGGER_CLIENT", "event_type":"GRPC_CALL_TRAILER", "log_level":"LOG_LEVEL_DEBUG", "metadata":{
                             "rpc_id":"3ddcd542-96d7-4338-bda2-fec5e3b0e81f", "sequence_id":"4", "service_name":"grpc.testing.SimpleService
                             "status_message":"ClientCall started after deadline exceeded. Deadline exceeded after -0.000494177s. ", "times

{
 insertId: "1vpe1kog1md4xrk"
 jsonPayload: {
  event_logger: "LOGGER_CLIENT"
  event_type: "GRPC_CALL_TRAILER"
  log_level: "LOG_LEVEL_DEBUG"
  metadata: {
  }
  method_name: "UnaryRpc"
  rpc_id: "3ddcd542-96d7-4338-bda2-fec5e3b0e81f"
  sequence_id: "4"
  service_name: "grpc.testing.SimpleService"
  status_code: 4
  status_message: "ClientCall started after deadline exceeded. Deadline exceeded after -0.000494177s. "
  timestamp: "2022-03-03T00:28:57.963Z"
 }
 logName: "projects/grpc-testing/logs/grpc"
 receiveTimestamp: "2022-03-03T00:28:58.008879804Z"
 resource: {
  labels: {
   project_id: "grpc-testing"
  }
  type: "global"
```

gRPC Observability Logging Preview



```
logName=~"/logs/grpc"
```

```
2022-03-04 18:47:41.639 CST  - {"authority":"localhost:41661", "event_logger":"LOGGER_CLIENT", "event_type
2022-03-04 18:47:41.639 CST  - {"event_logger":"LOGGER_CLIENT", "event_type":"GRPC_CALL_REQUEST_MESSAGE",
2022-03-04 18:47:41.639 CST  - {"event_logger":"LOGGER_CLIENT", "event_type":"GRPC_CALL_HALF_CLOSE", "log_
2022-03-04 18:47:41.731 CST  - {"event_logger":"LOGGER_CLIENT", "event_type":"GRPC_CALL_RESPONSE_HEADER",
"sequence_id":"4", "service_name":"grpc.testing.SimpleService", "timestamp":"202
{
  insertId: "qxgillfqnrx8n"
  jsonPayload: {
   event_logger: "LOGGER_CLIENT"
   event_type: "GRPC_CALL_RESPONSE_HEADER"
   log_level: "LOG_LEVEL_DEBUG"
   metadata: {1}
   method_name: "UnaryRpc"
   payload_size: 24
   peer_address: {3}
   rpc_id: "c6486f72-e691-4bfc-aad6-533e4f0d258e"
   sequence_id: "4"
   service_name: "grpc.testing.SimpleService"
   timestamp: "2022-03-05T00:47:41.641Z"
  }
  labels: {
   APP_ID: "24512"
   DATACENTER: "SAN_JOSE_DC"
  }
  logName: "projects/grpc-testing/logs/grpc"
  receiveTimestamp: "2022-03-05T00:47:41.731806738Z"
  resource: {
   labels: {
    cluster_name: "ga-cluster-psm-sec"
    container_name: ""
    location: "us-west1-b"
    namespace_name: "grpc-example-server-ns"
    pod_name: "example-grpc-server-779bbb9644-prc6h"
    project_id: "grpc-testing"
   }
   type: "k8s_container"
  }
  severity: "DEBUG"
  sourceLocation: {3}
  timestamp: "2022-03-05T00:47:41.731806738Z"
}
```

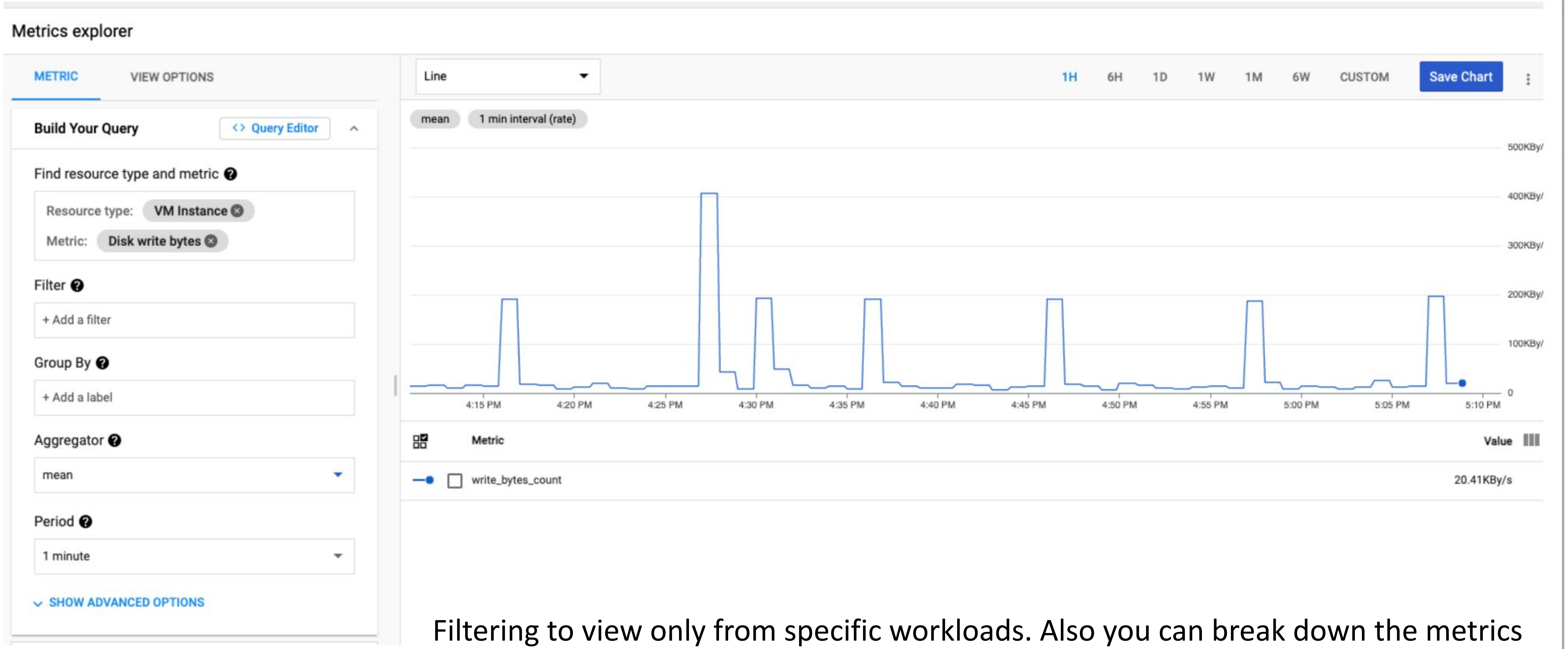
Custom Tags

Location Tags

gRPC Observability: Metrics & Trace

- Private Preview of Metrics & Traces coming soon - before end of Q2’22 for Java and Go
- Integrated with Google Cloud Monitoring and Google Cloud Trace
- Incorporate metrics views into Monitoring dashboards and charts
- Trace Overview shows recent traces: select individual trace to see breakdown of traffic

gRPC Observability: Metrics



Metrics explorer

METRIC · VIEW OPTIONS

Build Your Query · Query Editor

Find resource type and metric

Resource type: VM Instance

Metric: Disk write bytes

Filter: + Add a filter

Group By: + Add a label

Aggregator: mean

Period: 1 minute

SHOW ADVANCED OPTIONS

Line · 1H · 6H · 1D · 1W · 1M · 6W · CUSTOM · Save Chart

mean · 1 min interval (rate)

Metric	Value
write_bytes_count	20.41KBy/s

Filtering to view only from specific workloads. Also you can break down the metrics by grpc_client_method or grpc_server_method

gRPC for Microservices: Wrap-up

Resources

Visit http://grpc.io/meet to “meet” with gRPC Maintainers

Other Resources

- http://grpc.io
- Istio Blog: gRPC Proxyless Service Mesh

gRPC for Microservices: Service-mesh & Observability

Questions?