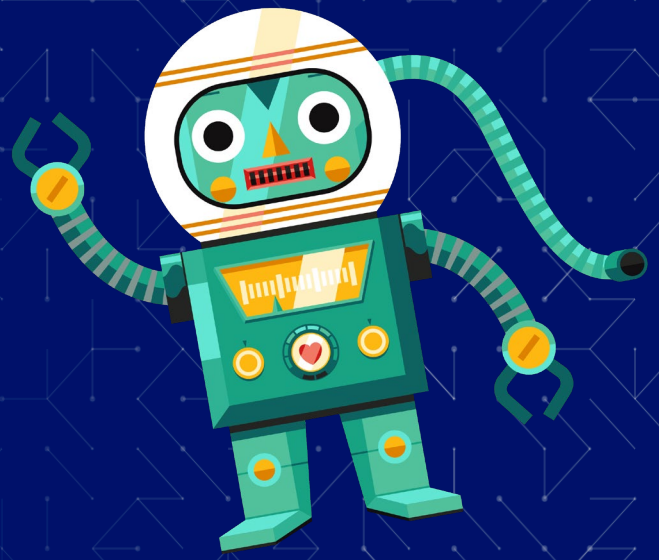


TECHORAMA

Why gRPC

Collaboard
Great minds think together

Collaboard (web.collaboard.app)

Massive R&D on real-time communication

Based on web-socket

Looking for alternatives

gRPC

Everything you will never find online

TECHORAMA

What’s gRPC

gRPC Stands for Remote Procedure Call

is a modern high-performance RPC framework

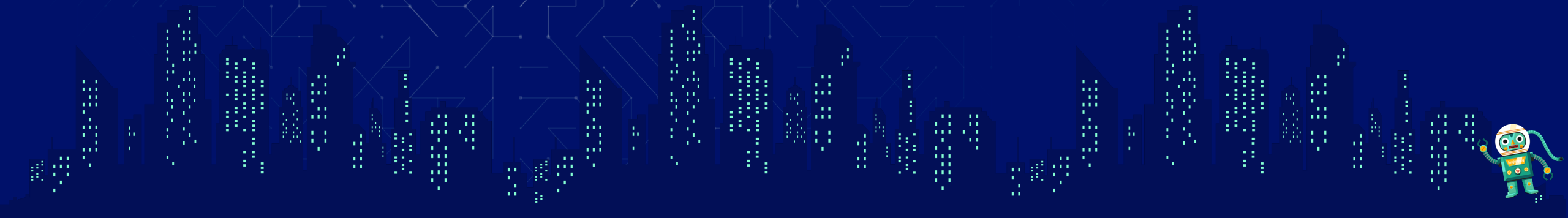
is based on HTTP/2, Protocol Buffers and other modern standard-based technologies

What’s gRPC

gRPC is Payload agnostic: is support protobuf but other serialization formats like flatbuffers, json etc are easily configurable

The architecture is extensible, most of the feature are provided by Filters: Auth, tracing, name resolution, LB, message size checks, RPC deadlines

Works with many transports, by default uses HTTP2 (other implementaions: QUIC, Cronet, Inprocess and more)



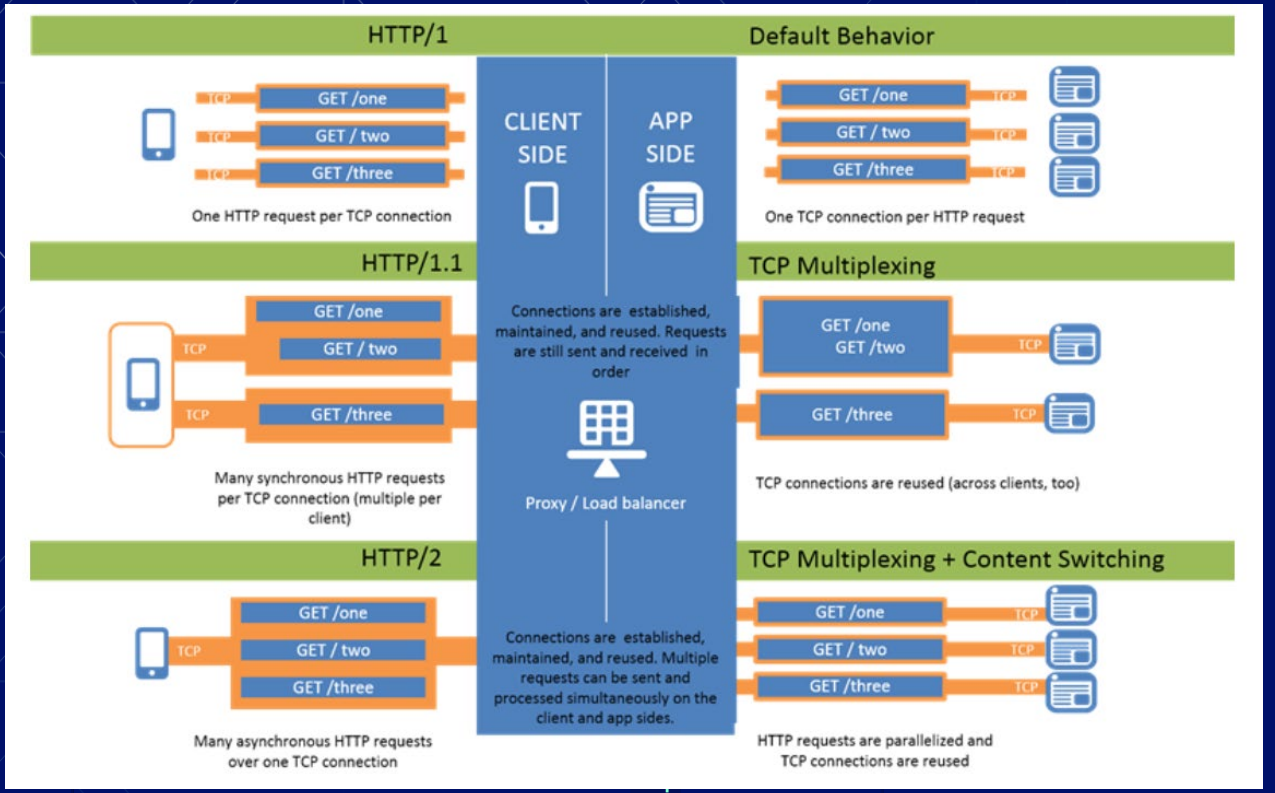
HTTP/2 Request/Response Multiplexing

Head of line blocking

Interleave multiple requests and responses in parallel without blocking on anyone

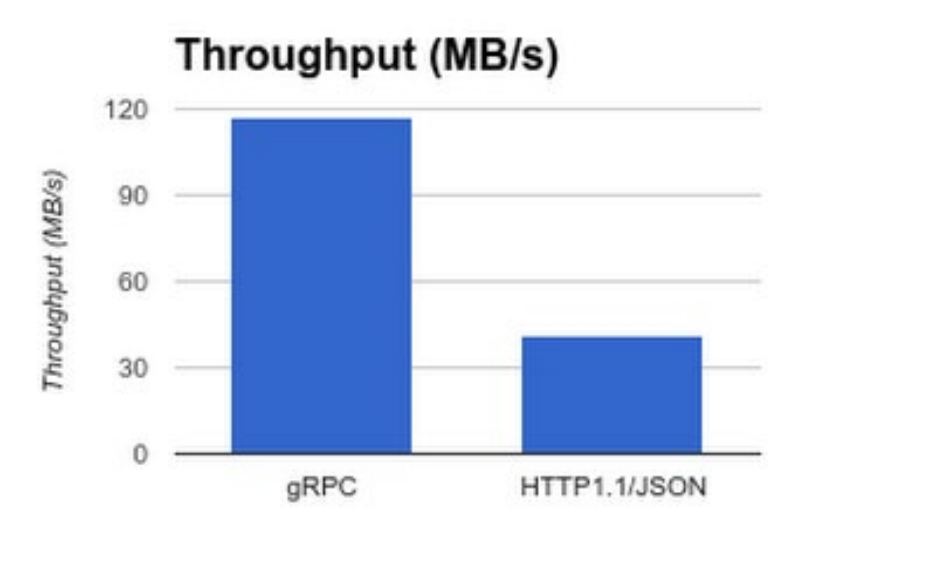
Use a **single TCP connection** to deliver multiple requests and responses in parallel

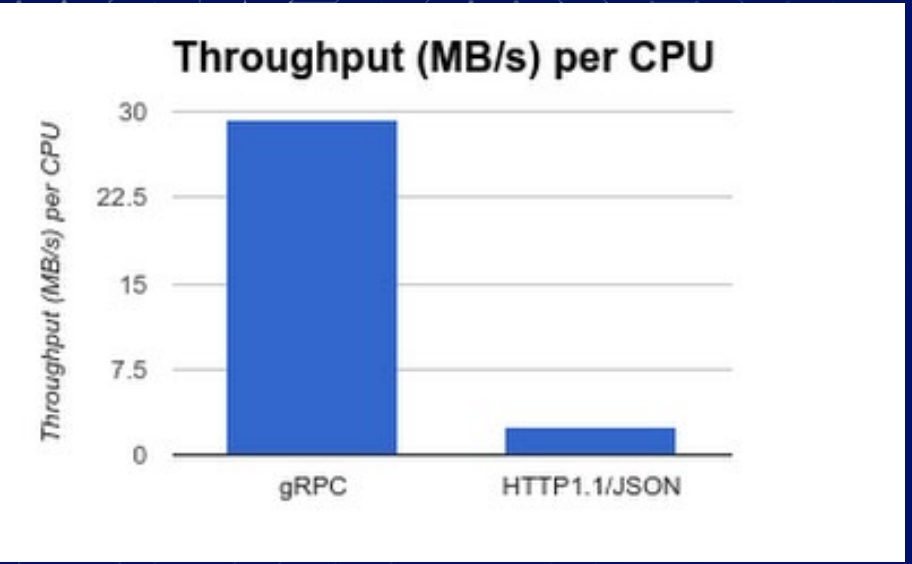
Enable flow-control, server push, etc



Source https://devcentral.f5.com/

Protocol Buffers / HTTP2





3x increase throughput
publishing 50KB messages at maximum throughput from a single n1-highcpu-16 Google Compute Engine Virtual Machine instance, using 9 gRPC channels.

11x difference per CPU
More impressive than the almost 3x increase in throughput, is that it took only 1/4 of the CPU resources

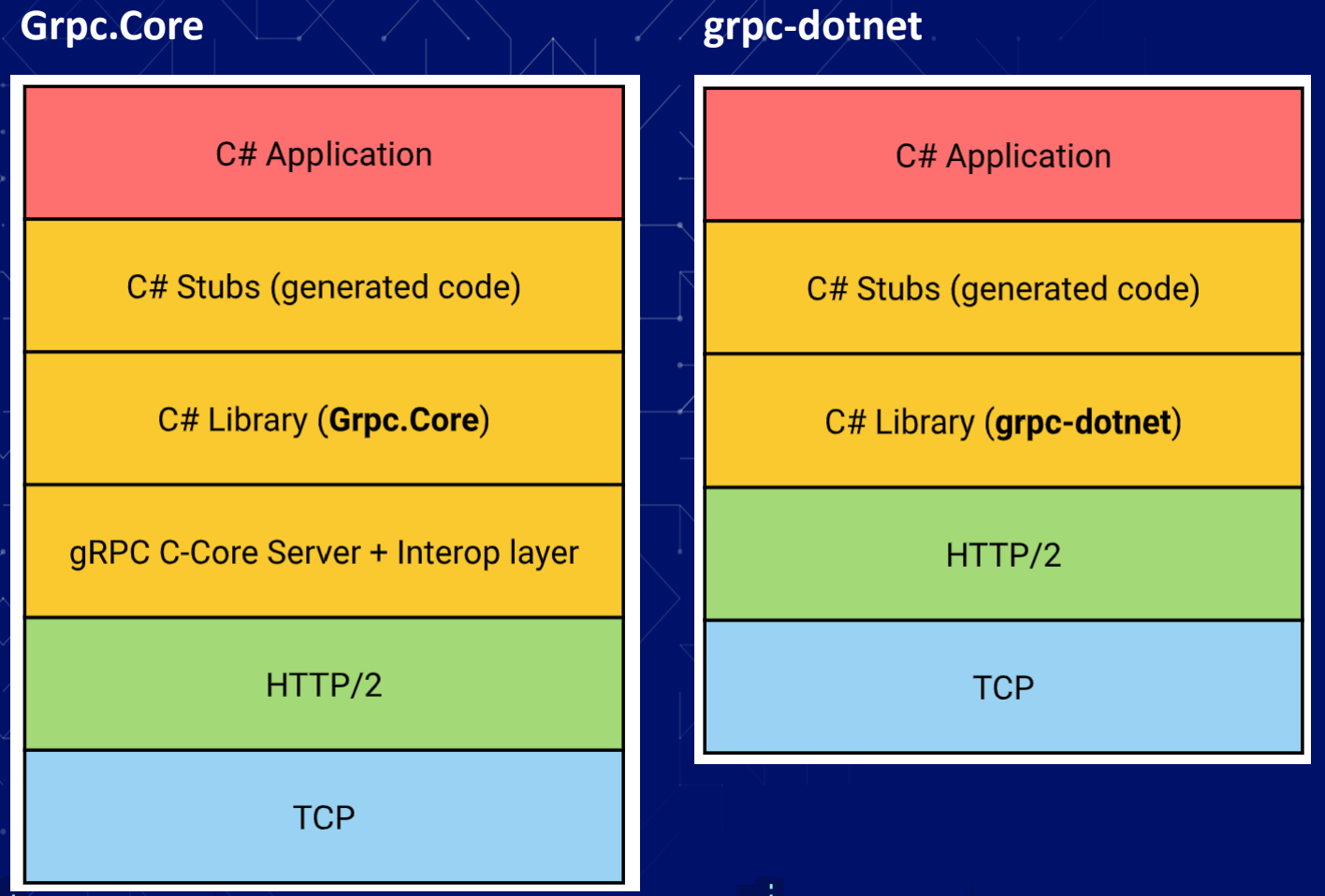
Source M.Atamel, R.Tsang @Google https://cloud.google.com/blog/products/gcp/announcing-grpc-alpha-for-google-cloud-pubsub

gRPC the state of the art

There are currently two official implementations of gRPC for .NET:

Grpc.Core
The original gRPC C# implementation based on the native gRPC Core library.
https://github.com/grpc/grpc/tree/master/src/csharp
https://www.nuget.org/packages/Grpc.Net.Client/

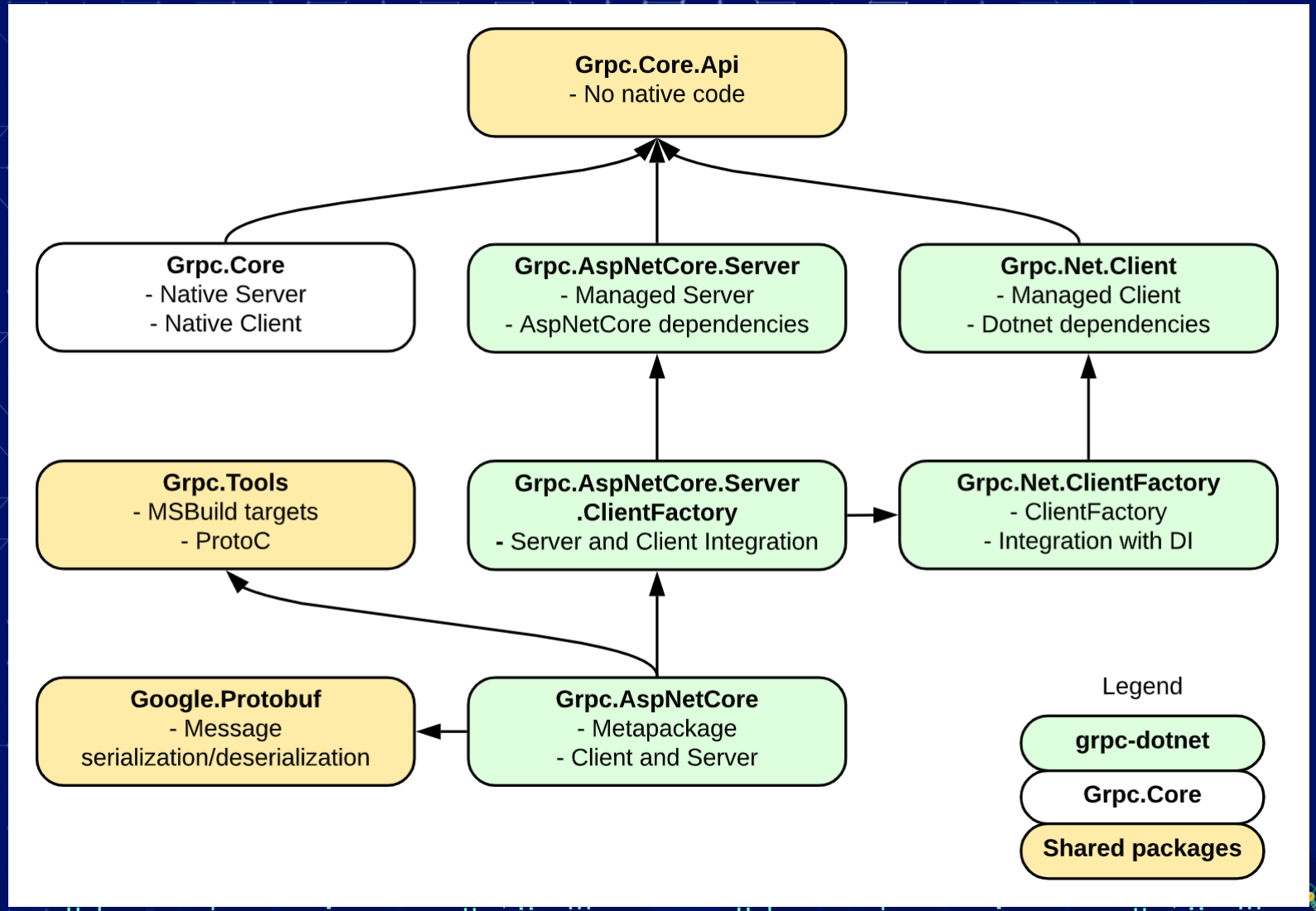
grpc-dotnet
The new implementation written entirely in C# with no native dependencies and based on the newly released .NET Core 3.0.
https://github.com/grpc/grpc-dotnet
https://www.nuget.org/packages/Grpc.AspNetCore.Server/



Source https://grpc.io/

gRPC the state of the art

all new .NET packages for gRPC and their relationship with the existing packages.



Source https://grpc.io/

The future of gRPC

grpc-dotnet is the new gRPC!!!!!!

The new implementation written entirely in C# with no native dependencies and based on the newly released .NET 5.0.

https://github.com/grpc/grpc-dotnet

https://www.nuget.org/packages/Grpc.AspNetCore.Server/

Grpc.Core effective May first 2021 will enter maintenance mode and won't be getting any new features

The original gRPC C# implementation based on the native gRPC C++ library

https://github.com/grpc/grpc/tree/master/src/csharp

https://www.nuget.org/packages/Grpc.Net.Client/

May 2021 maintenance mode

May 2022 deprecated

Source https://grpc.io/

The future of gRPC

grpc-dotnet is the new gRPC!!!!!!

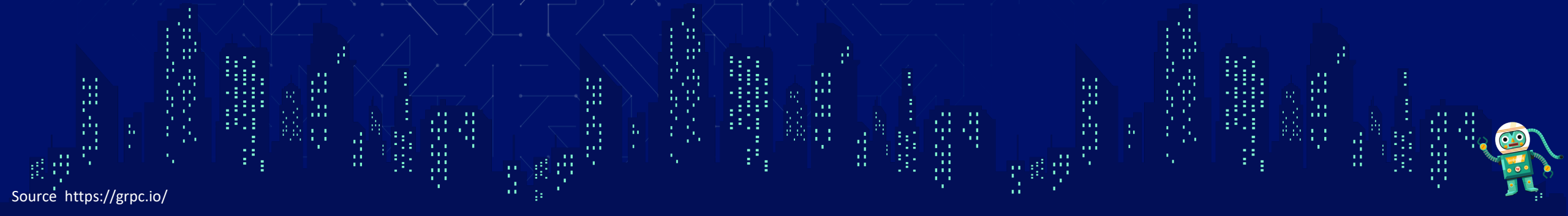
The new implementation written entirely in C# with no native dependencies and based on the newly released .NET Core 3.0.

https://github.com/grpc/grpc-dotnet

https://www.nuget.org/packages/Grpc.AspNetCore.Server/

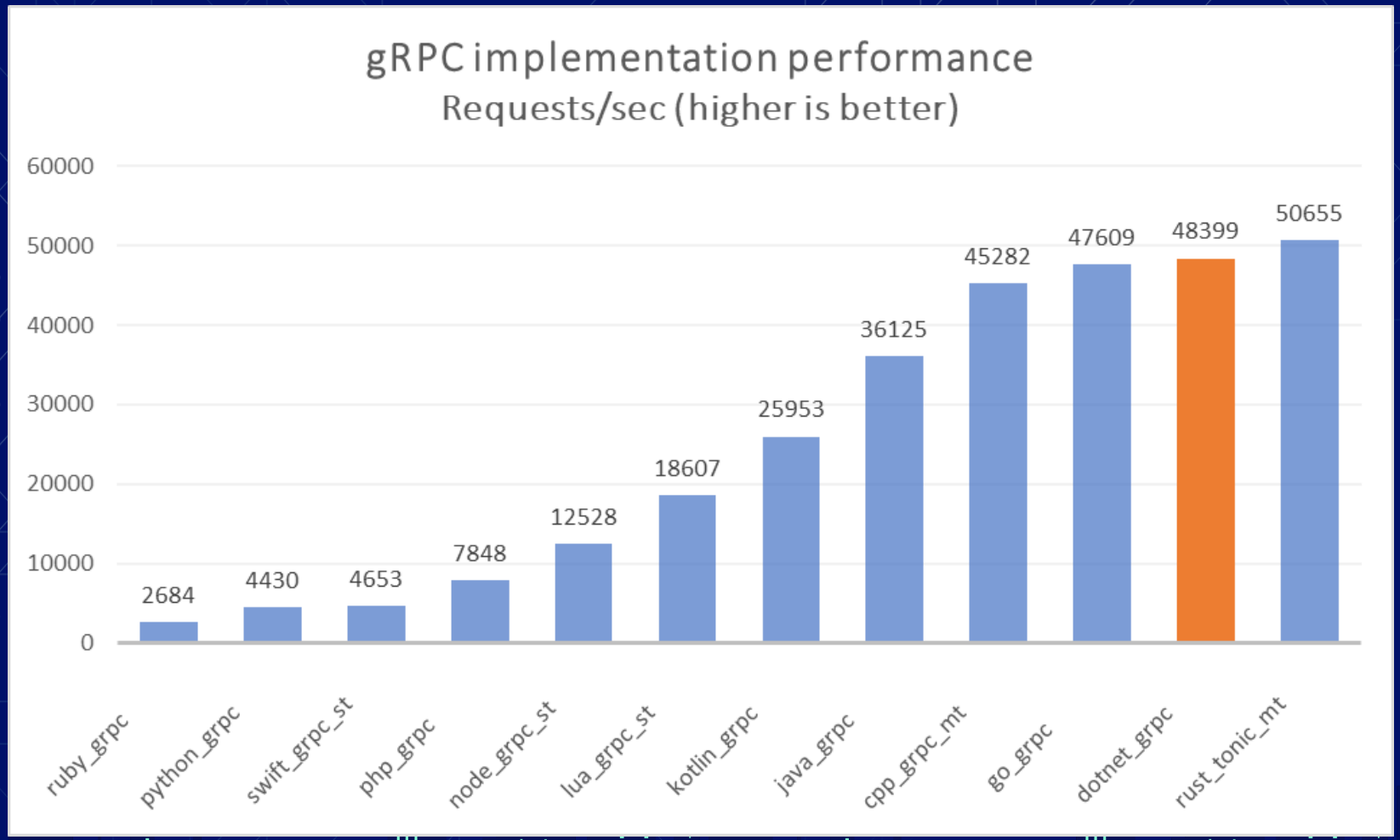
- It's a more modern implementation
- It's more aligned with where the C# / .NET community
  The implementation is much more agile and contribution friendly
- Internally based on well known primitives / APIs (ASP.NET core serving APIs and HTTP2 client) and it's implemented in pure C#
- Having a library that is implemented in pure C# is something that's generally favored by the .NET community

NOTE: the Google.Protobuf library for C# is already written purely in C#



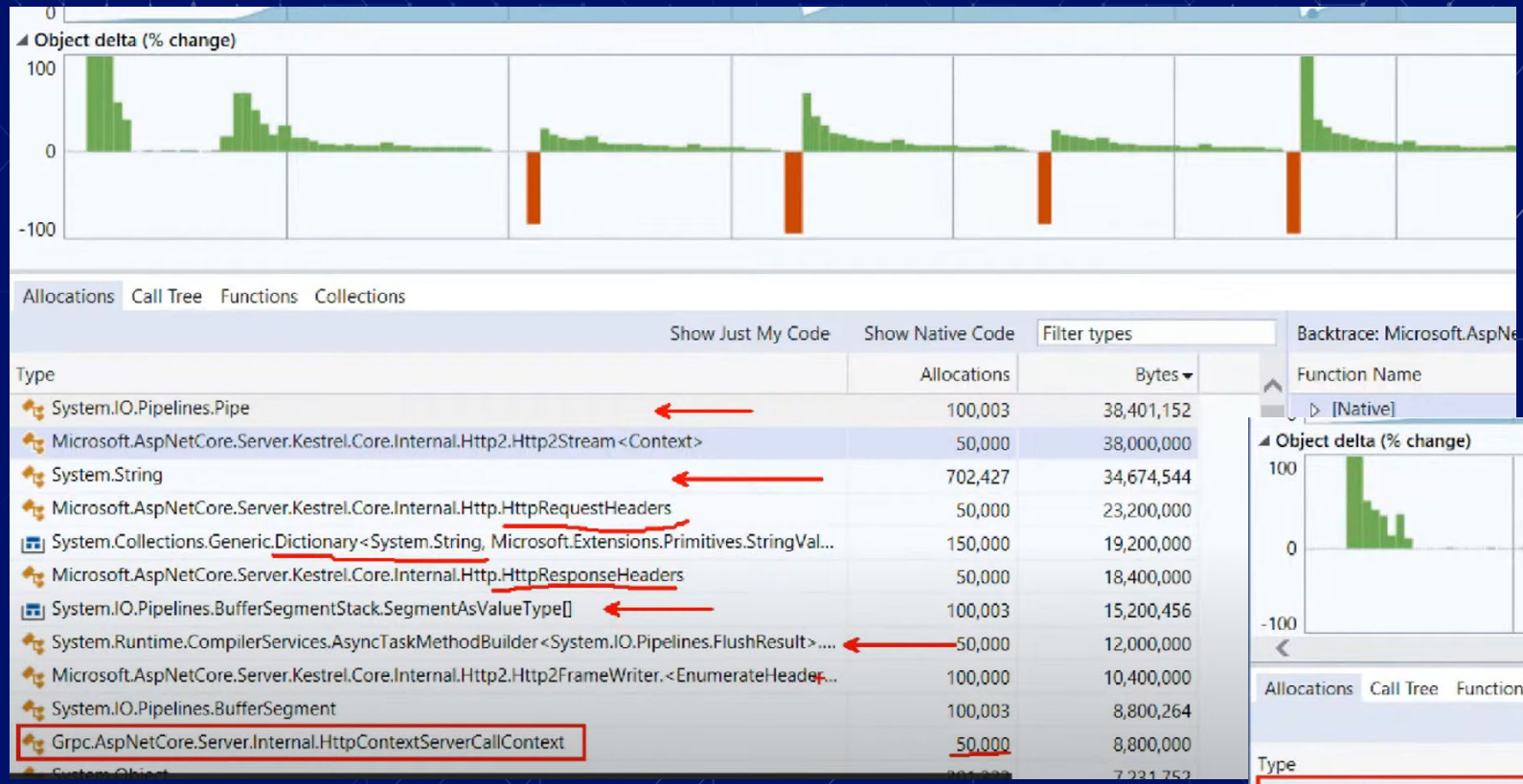
Source https://grpc.io/

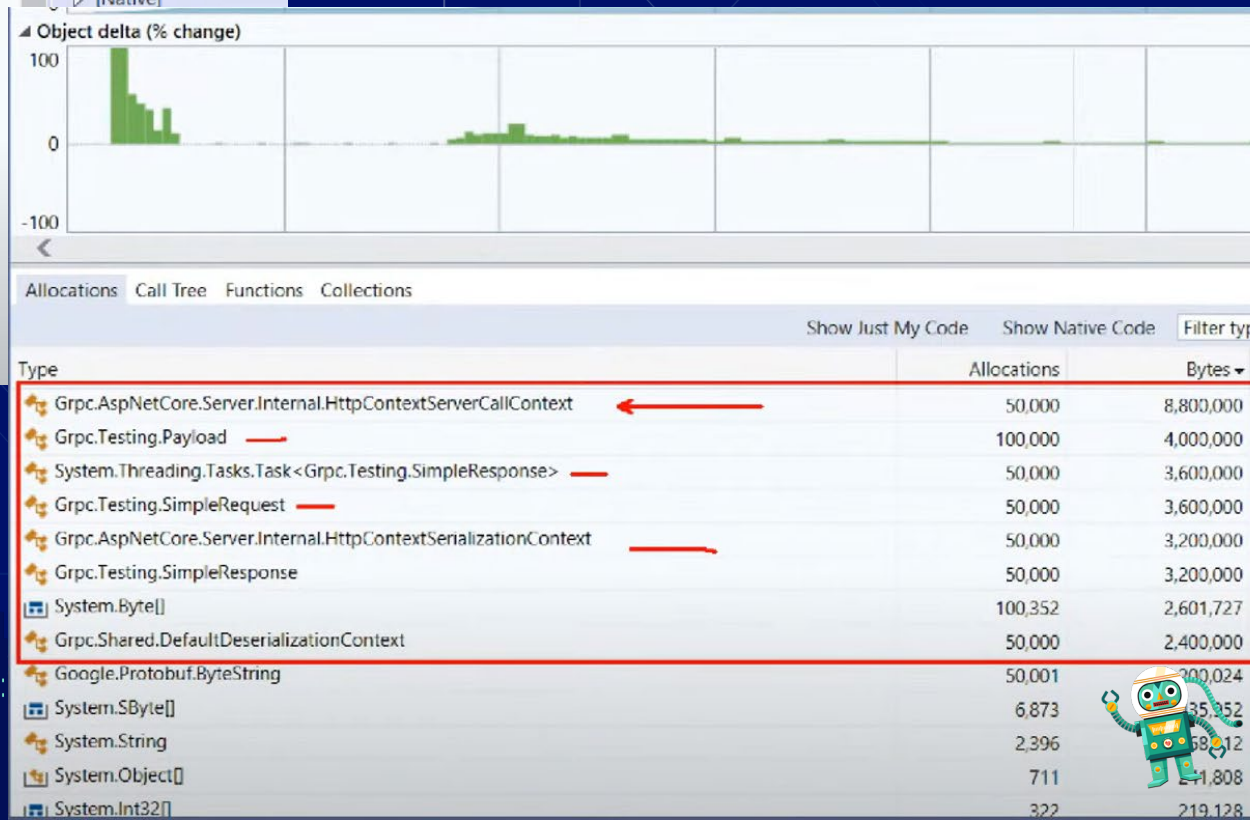
gRPC performance improvements in .NET 5



Source https://github.com/LesnyRumcajs/grpc_bench

gRPC performance improvements in .NET 5





HTTP/2 allocations in Kestrel
Pooling of HTTP2 streams
reduced by 92%!!

Source https://devblogs.microsoft.com

gRPC performance improvements in .NET 5

Added HPack response compression
Using Wireshark, we can see the impact of header compression on response size for this example gRPC call. .NET Core 3.x writes 77 B, while .NET 5 is only 12 B.

.NET Core 3.x - HTTP/2 response headers

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 01   ........ ......`.
0010  20 51 00 71 06 80 00 00  00 00 00 00 00 00 00 00    Q.q..... ........
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 01 13 88  c8 e0 92 50 ee 5d 03 0d   ........ ...P.]..
0040  d8 f1 50 18 27 f5 3f 80  00 00 00 00 44 01 04 00   ..P.'.?. ....D...
0050  00 00 01 88 00 04 64 61  74 65 1d 57 65 64 2c 20   ......da te.Wed,
0060  31 34 20 4f 63 74 20 32  30 32 30 20 30 35 3a 31   14 Oct 2 020 05:1
0070  33 3a 31 32 20 47 4d 54  00 0c 63 6f 6e 74 65 6e   3:12 GMT ..conten
0080  74 2d 74 79 70 65 10 61  70 70 6c 69 63 61 74 69   t-type.a pplicati
0090  6f 6e 2f 67 72 70 63 00  00 07 00 00 00 00 00 01   on/grpc. ........
00a0  00 00 00 00 02 0a 00                               .......
```

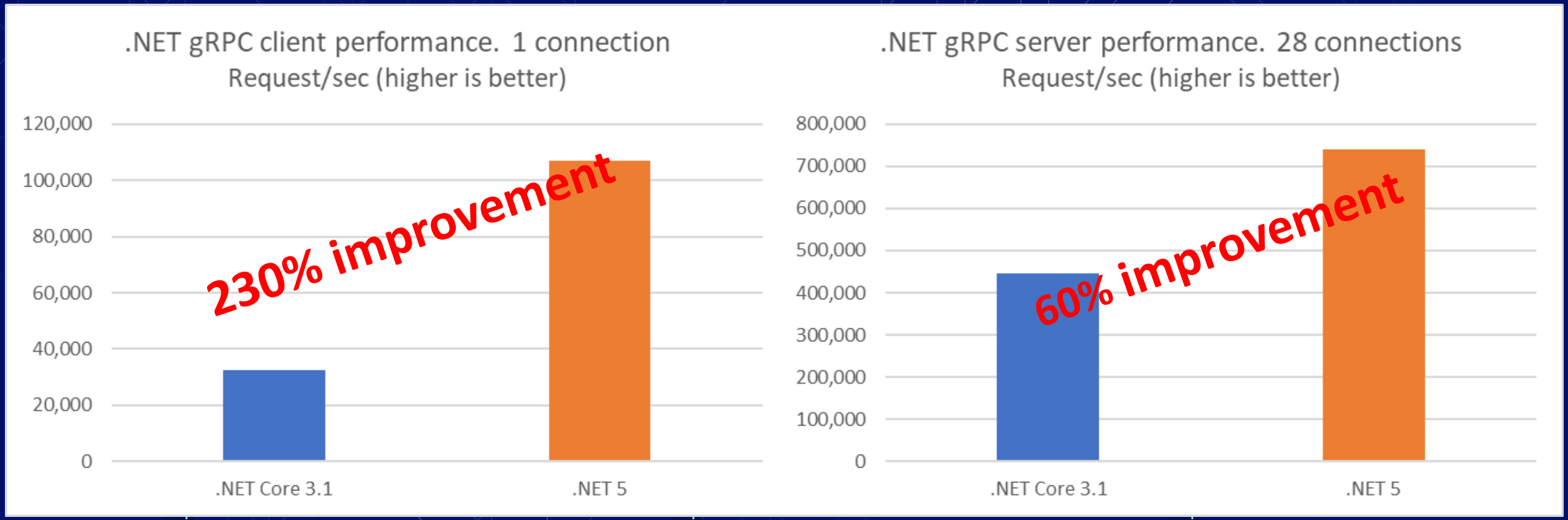
.NET 5 - HTTP/2 response headers

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 07   ........ ......`.
0010  0f 15 00 3a 06 80 00 00  00 00 00 00 00 00 00 00   ...:.... ........
0020  00 00 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 01 13 88  f9 17 9d aa 6c 8f 61 18   ........ ....l.a.
0040  35 c8 50 18 27 f4 f0 ec  00 00 00 00 03 01 04 00   5.P.'... ........
0050  00 00 03 88 c0 bf 00 00  07 00 00 00 00 00 03 00   ........ ........
0060  00 00 00 02 0a 00 00 00  01 01 05 00 00 00 03 be   ........ ........
```

Protobuf message serialization uses Span<T>
gRPC for .NET serializes Protobuf messages directly to Kestrel's request and response buffers. Intermediary array allocations and byte copies have been eliminated from gRPC message serialization

Source https://devblogs.microsoft.com

gRPC performance improvements in .NET 5

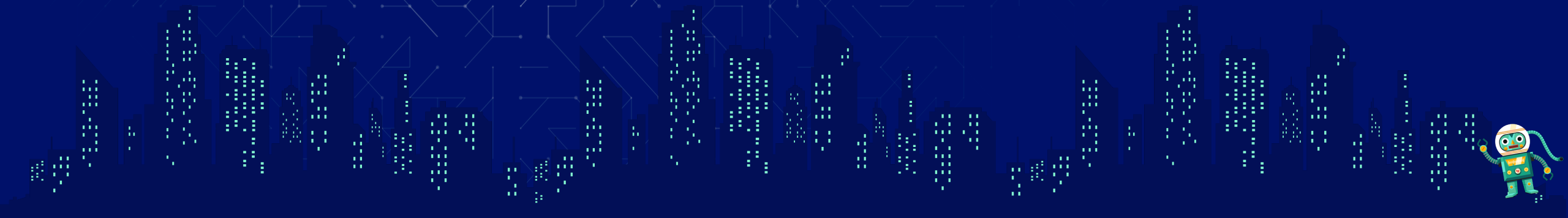


Source https://devblogs.microsoft.com

gRPC .NET 5 – New platform primitives

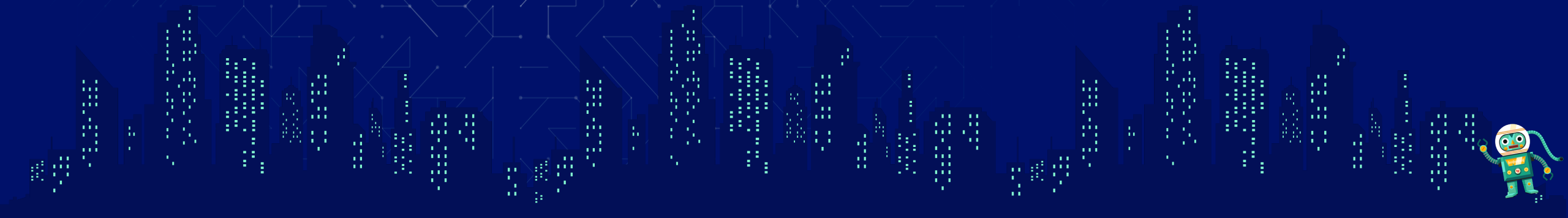
- Enhanced setup for TLS
- Difficult to setup distributed tracing
- OS native crypto stack
  - Support for HSMs (Hardware security module) and Secure Enclave

```
// Using Grpc.Core
var channelCredentials = new SslCredentials(File.ReadAllText("roots.pem"));
var channel = new Channel("https://myservice.example.com", channelCredentials);
var client = new Greeter.GreeterClient(channel);

// Using grpc-dotnet
var channel = GrpcChannel.ForAddress("https://myservice.example.com");
var client = new Greeter.GreeterClient(channel);
```



gRPC in .NET 6

- gRPC retries
- gRPC client on .NET Standard
- HTTP/3
- Client-side load balancing



gRPC – Two more challenges to solve

IIS / App Service – Web Clients

It's true that IIS cannot current host a gRPC service because http.sys doesn't support trailing headers.
gRPC relies on trailing headers to communicate vital information, like call status.
In Azure, Windows based App Services use IIS and have the same http.sys limitation.

The App Service front-end terminates incoming TLS connections before forwarding unencrypted messages to your App Service worker. Your gRPC service will refuse to communicate over http and return only HTTP 426 Upgrade Required errors.

gRPC services with C#

07/09/2020 • 3 minutes to read • +3

This document outlines the concepts needed to write gRPC apps in C#. The topics covered here apply to both C-core -based and ASP.NET Core-based gRPC apps.

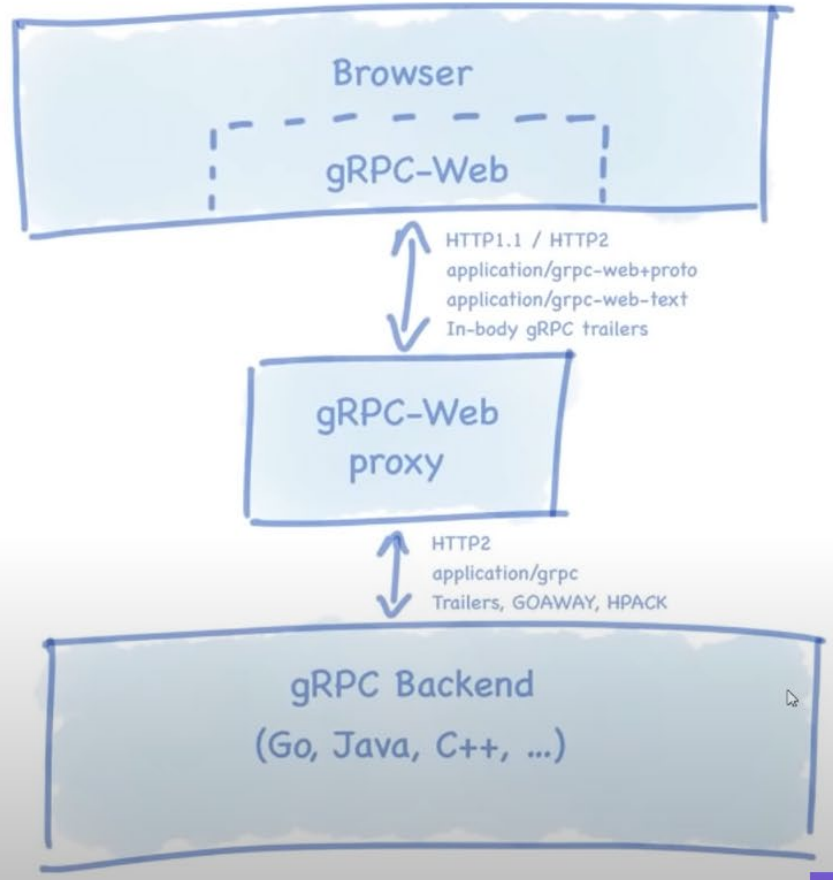
⚠ **Warning**

ASP.NET Core gRPC has extra requirements for being used with Azure App Service or IIS. For more information on where gRPC can be used, see **gRPC on .NET supported platforms**.

gRPC – Web clients

gRPC-Web

- Transform gRPC requests
  - Move HTTP/2 trailers into response body
  - Encode messages in base64
- Envoy Proxy transforms requests going to server
- Proxy works with all gRPC implementations
- gRPC-Web JavaScript client



gRPC-Web in .NET

gRPC-web server support
- ASP.NET middleware
- Converts incoming gRPC-Web to gRPC
- In-process proxy

gRPC-Web client support
DelegatingHandler
Converts outgoing gRPC to gRPC-Web

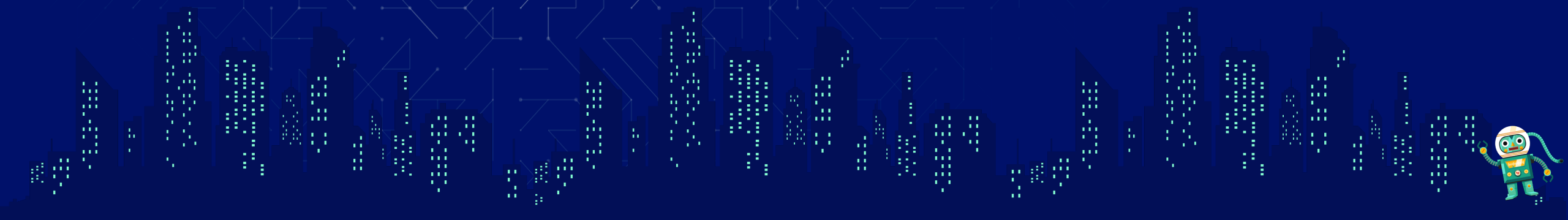
```
app.UseRouting();

app.UseGrpcWeb();

app.UseEndpoints(endpoints =>
{
    endpoints
        .MapGrpcService<GreeterService>()
        .EnableGrpcWeb();
});
```

```
var httpHandler = new GrpcWebHandler(
    GrpcWebMode.GrpcWebText,
    new HttpClientHandler());

var c = GrpcChannel.ForAddress(
    backendUrl,
    new GrpcChannelOptions
    {
        HttpHandler = httpHandler
    });
```

gRPC Load Balancing - why

Build scalable services
Improve throughput, decreases latency
Avoid overloading of a single back end
Improved tolerance for back-end failures
Allows updating service on the fly

LB is key importance in microservice architecture



gRPC Load Balancing: L4 vs L7

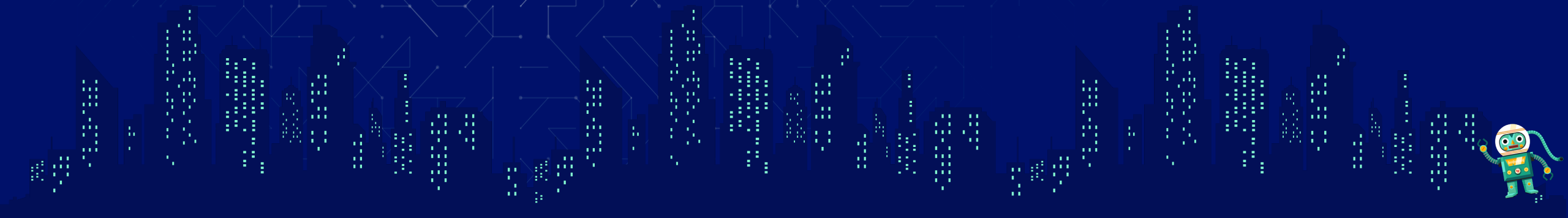
Level is based on OSI network model levels and it describes at what level the LB happen

- L4 Connection based
- L7 Stream based happening at application layer

With L4 load balancer the balancing is happening very time a new TCP/IP connection is created – works fine with HTTP1.1/REST APIs

gRPC uses HTTP2: every RPC is a separate stream in the same TCP/IP connection
gRPC traffic needs an L7 LB

Potential problem: Kubernetes LB is only L4 (= in service types clusterip and loadbalancer)

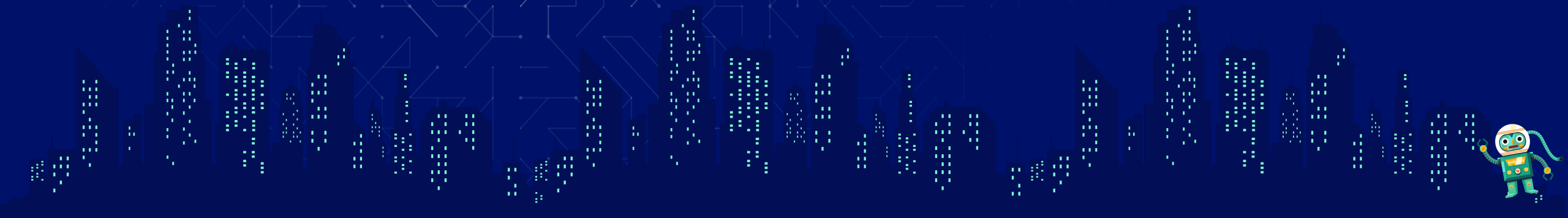


gRPC Load Balancing: Client vs Proxy

Proxy LB

+ simple client, untrusted clients are fine
- Higher latency & overhead
- “Sidecar” deployment possible on Kubernetes

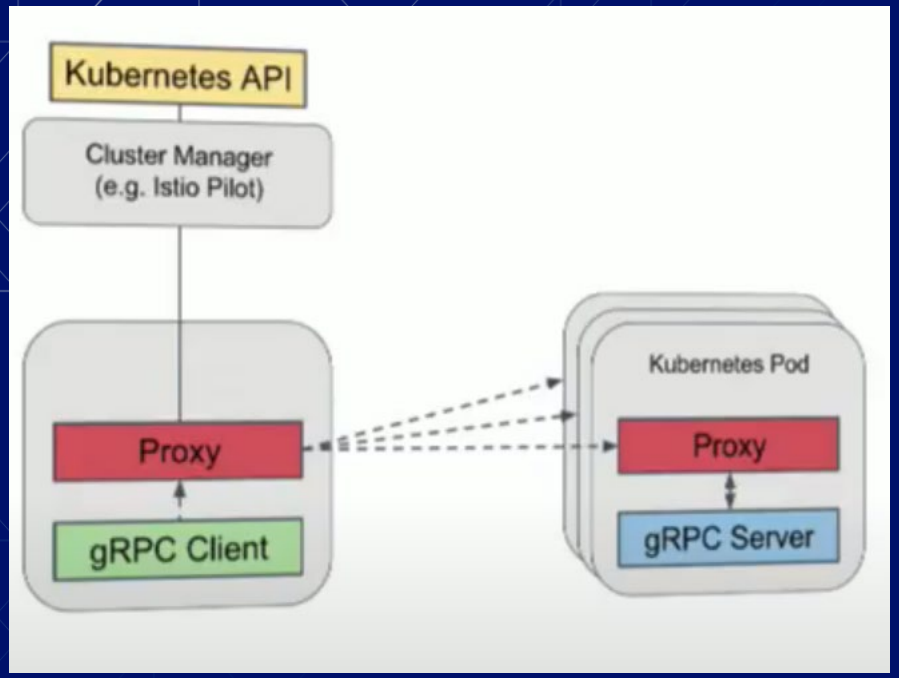
- Client LB
- + low latency, log overhead, no proxy management
- - usually only good for simple LB logic
- gRPC implements RoundRobin and “grpclb” lookaside



gRPC Load Balancing – Service Mesh

Service Mesh provides different functionalities, once is LB

- Proxy deployed as a service side-car
- LB informed by proxy
- Many additional features available



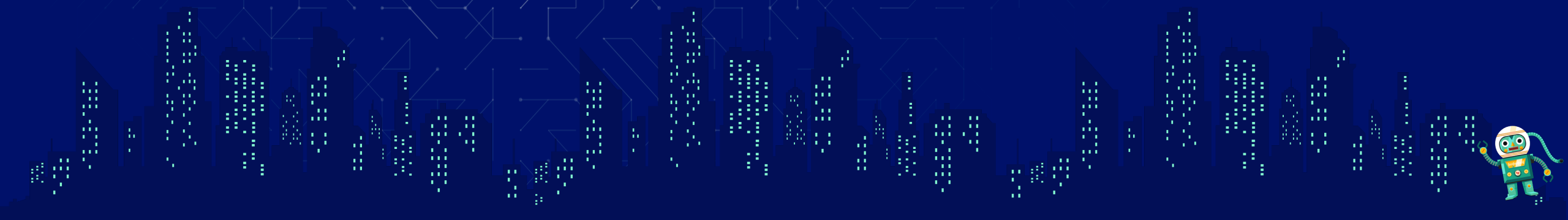
gRPC Load Balancing Options

Proxy LB

- Envoy (can be configured dynamically. It can talk with the cluster manager)
- NGINX
- Proxies that support both HTTP/2 and LB should work

Proxy LB in a Service Mesh

- Envoy / Istio
- Linkerd

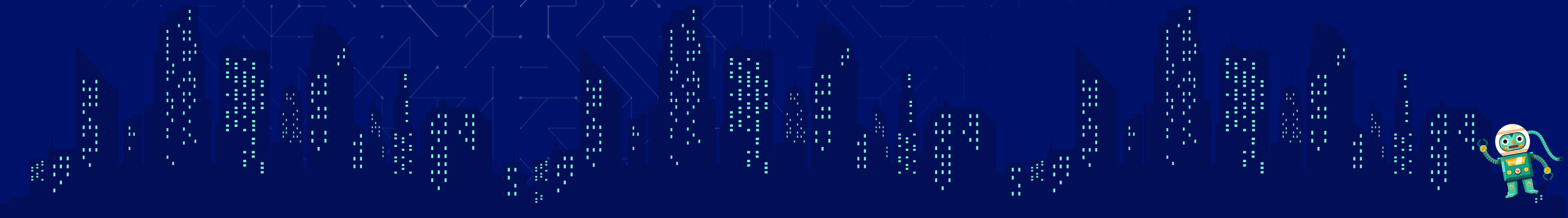


gRPC Load Balancing

- Envoy uses the Universal data plane API do discover endpoints
- gRPC recently added Universal data plane API support

Two possible deployment models

- Envoy proxy dose the lookaside load balancing
- gRPC client consumes data plane API directly (as a grpclb alternative)

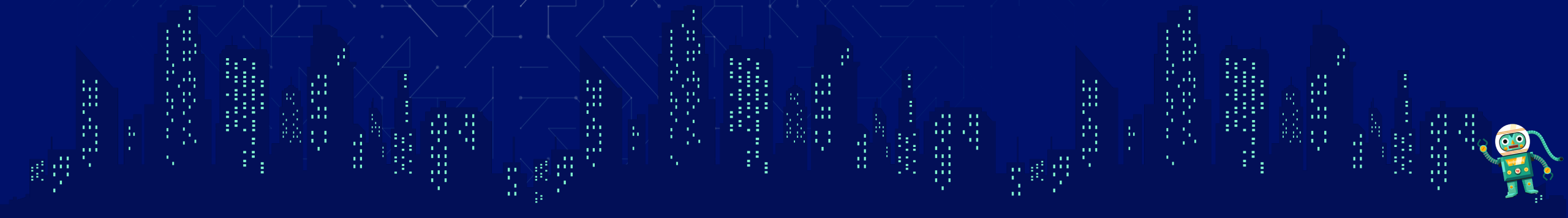


gRPC Security

gRPC provides a general framework to do auth and security (Metadata, Tokes)

The basic principles are very similar to securing HTTP traffic

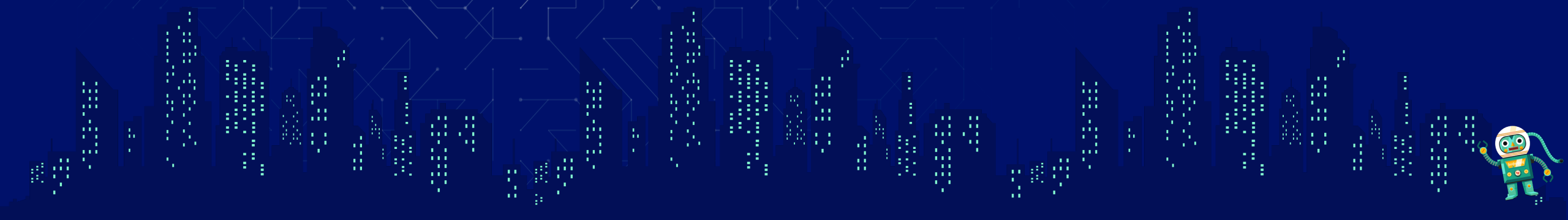
HTTP/2	gRPC Equivalent
Connection-level security	Use “Secure” channel
Request Headers	Initial metadata



gRPC Security

- Per-RPC based
  - Tokens in metadata
  - Needs an underlying secure transport
- Channel level (connection level)
  - TLS, Mutual TLS

- Transparent
  - Trusted proxy that does encryption/auth for you



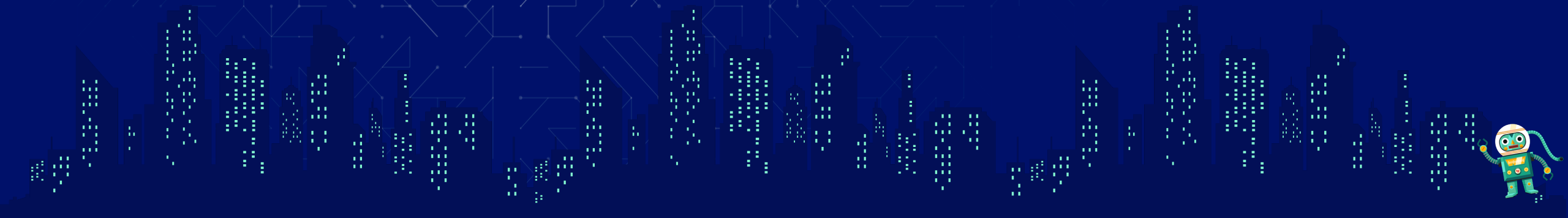
gRPC Security – Auth with tokens

Tokens that can be sent in request initial metadata

- Known ad “bearer tokens”
- OAuth2
- JWT (JSON Web Token)

Sending token with each RPC usually not a performance problem

- Token are reused for multiple request
- HPACK header compression (client sends a reference to the compressed table)

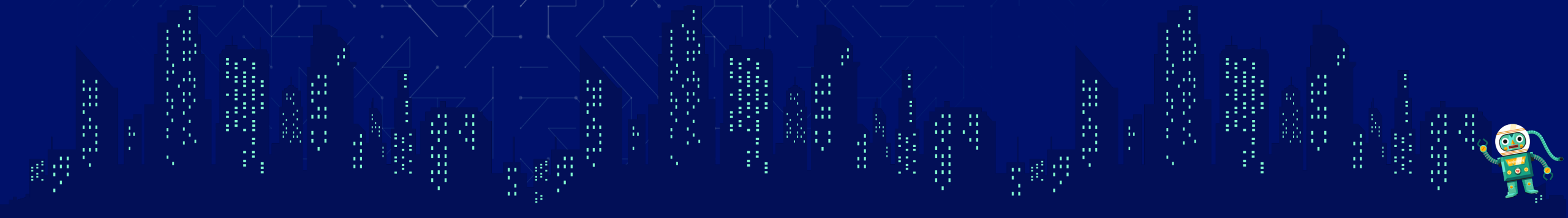


gRPC Security – Securing Channel

- gRPC Secure Channel
  - Exists in all gRPC implementations
  - Provides integrity and privacy, TLS is the default

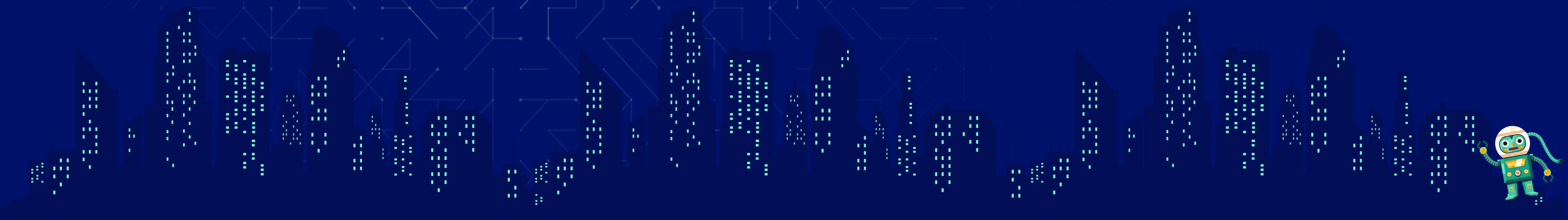
  TLS
  - Client makes sure it's talking to the right server (server has a certificate)
  - Any client can connect to the server

  Mutual TLS (mTLS)
  - Both server and client authenticate each other
  - Common in microservice scenario (server to server communication)



gRPC Security – mTLS in practice

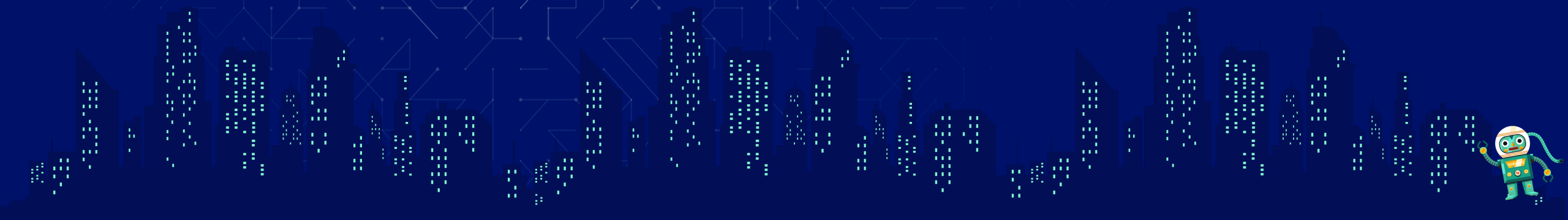
- Mutual TLS works great, but certificate distribution may be trivial
- Solution: Authentication done by service mesh (e.g. Istio)
  https://istio.io/latest/docs/concepts/security/



gRPC Security – Kubernetes Secrets

- Mutual TLS works great, but certificate distribution may be trivial
- Solution: Authentication done by service mesh (e.g. Istio)
  https://istio.io/latest/docs/concepts/security/

HSM – Hardware Security Module



PROTOBUF CI/CD

The challenges of generated code from .proto files

Packaging the generated code into libraries

Organizing the proto files

Proto file versioning

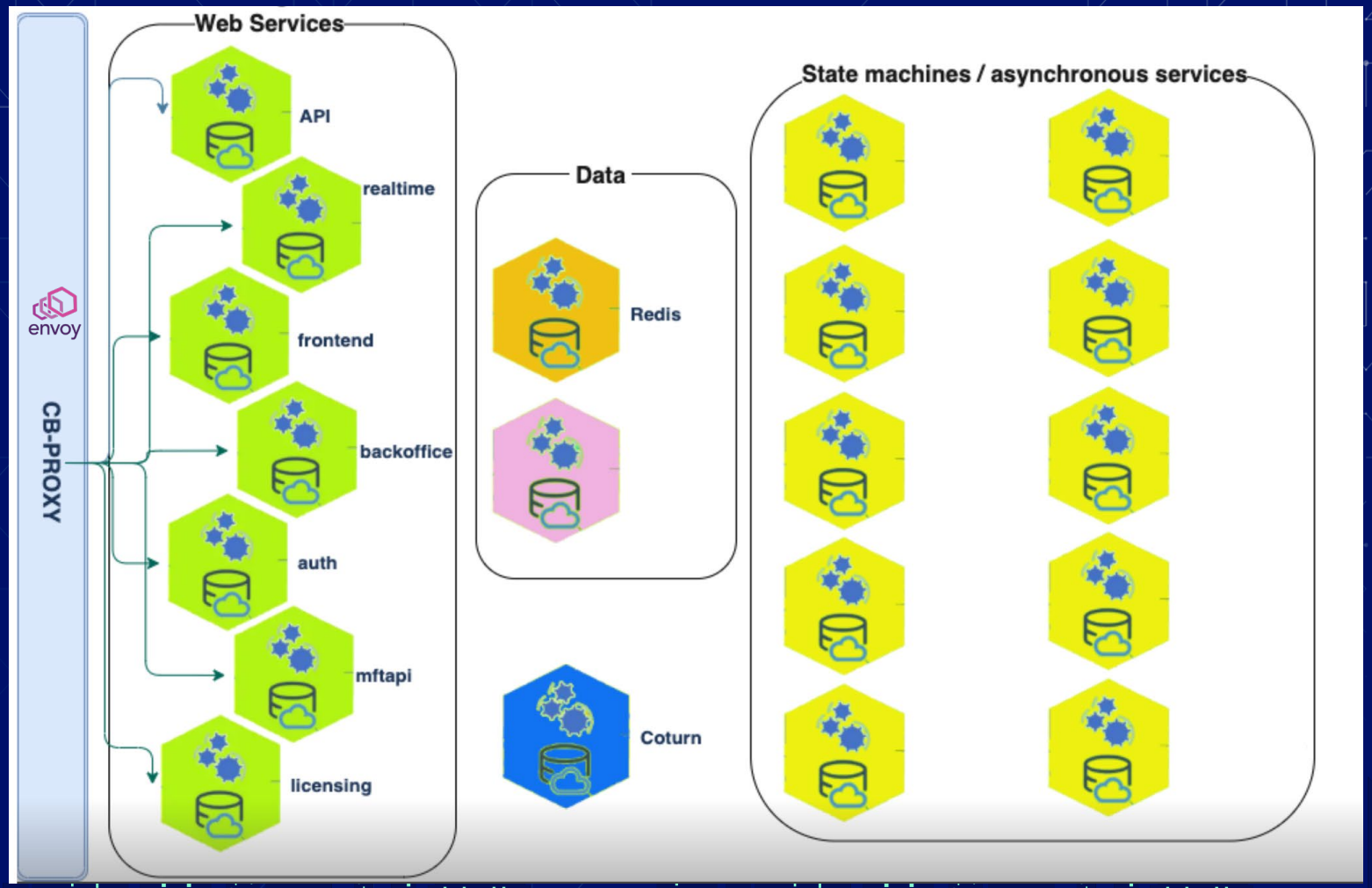
Generating libraries from proto files

Protobuf Language guide
https://developers.google.com/protocol-buffers/docs/proto3


```proto
syntax = "proto3";

import "google/protobuf/empty.proto";

package sensorsystem;

// service definition
service SensorService {
  // unary
  rpc GetAvailableSensors (AvailableSensorsRequest) returns (AvailableSensorsResponse);

  // Client streaming
  rpc SendSensorData (stream SensorData) returns (SensorDataResponse);
  rpc SendSensorDataNoResponse (stream SensorData) returns (google.protobuf.Empty);

  // Server streaming
  rpc ReceiveTemperatureUpdates (TemperatureRequest) returns (stream TemperatureData);

}

// The request message
message AvailableSensorsRequest {
  string username = 1;
  string message = 2;
}

// The response message
message AvailableSensorsResponse {
  string message = 1;
  string devices = 2;
}

message SensorData {
  string data1 = 1;
  int32 data2 = 2;
}

message SensorDataResponse {
  string message = 1;
}

message Device {
  string username = 1;
  string message = 2;
  bytes content = 3;
}

message TemperatureRequest {
  int32 deviceid = 1;
}

message TemperatureData {
  string devicelocation = 1;
  int32 temperature = 2;
}
```

How do we use gRPC

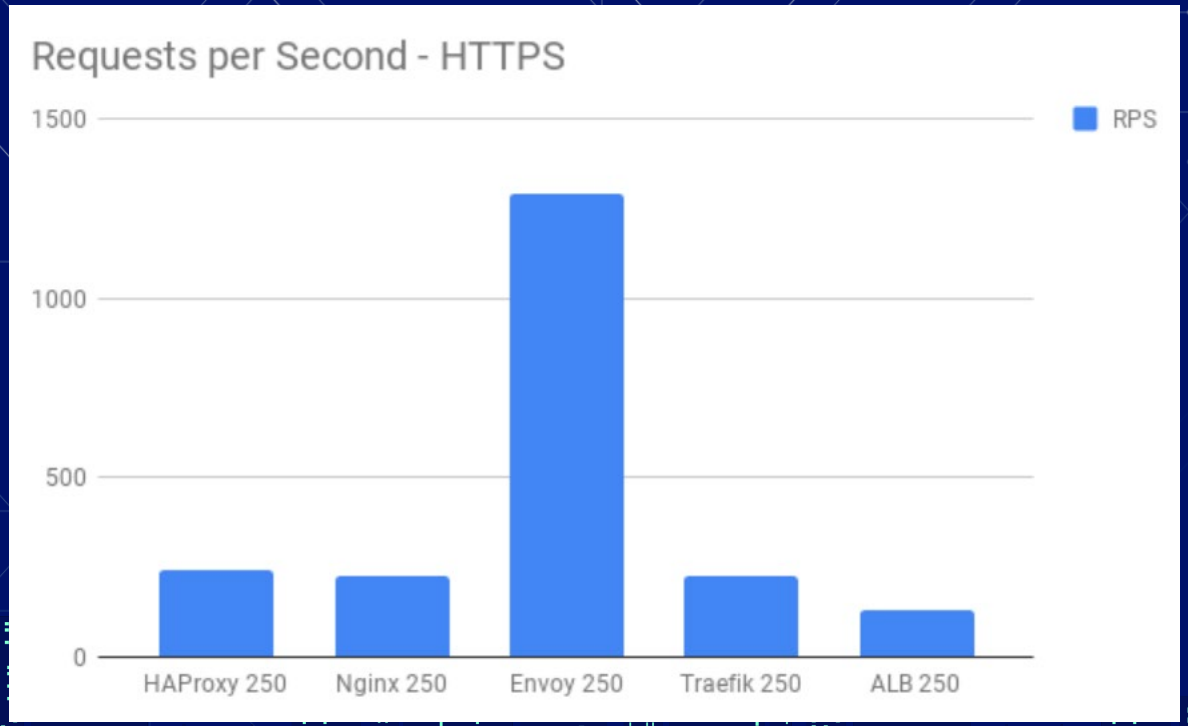
Collaboard
Great minds think together
https://web.collaboard.app



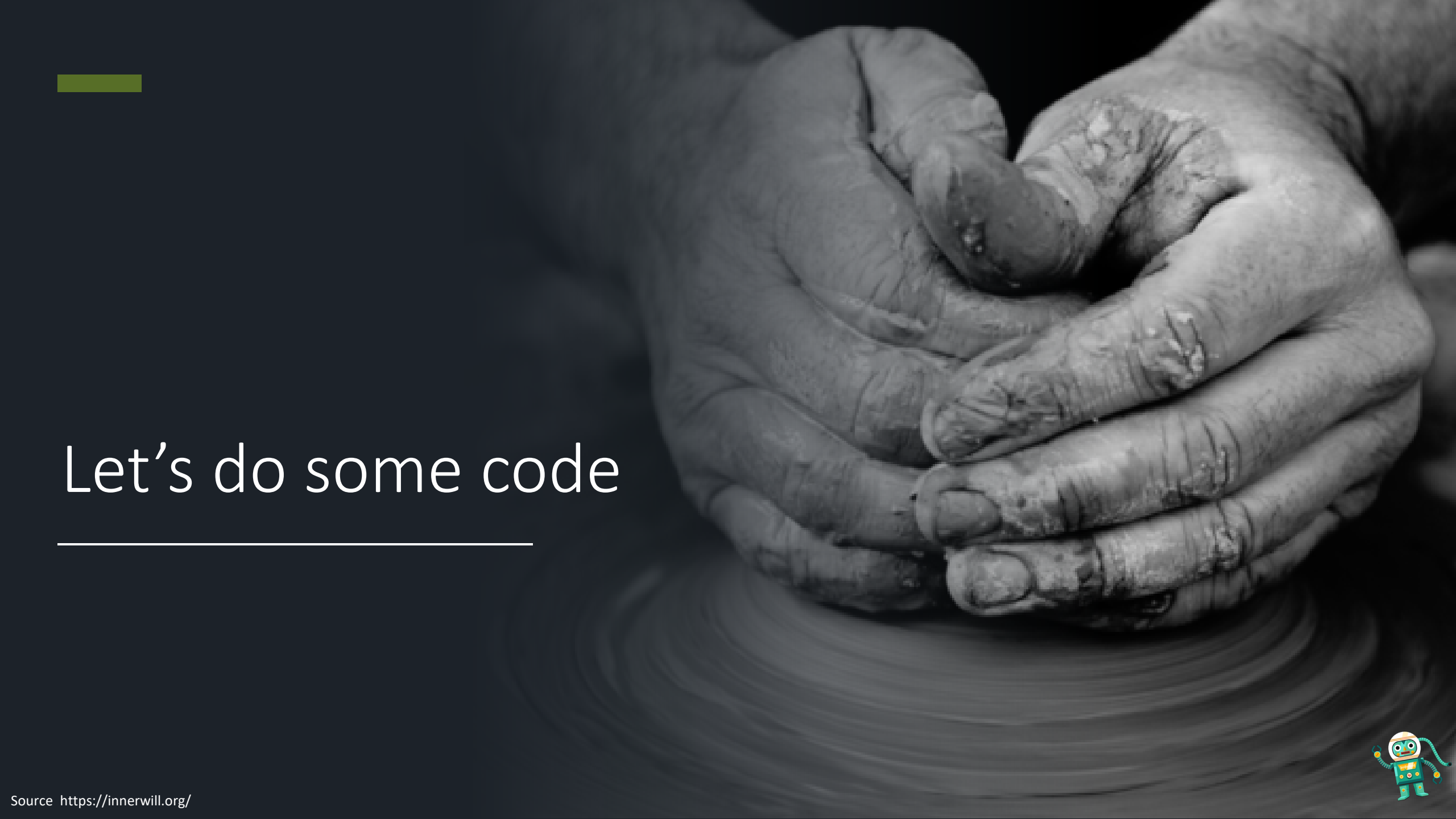
Envoy Performances

https://www.loggly.com/blog/benchmarking-5-popular-load-balancers-nginx-haproxy-envoy-traefik-and-alb/

https://dropbox.tech/infrastructure/how-we-migrated-dropbox-from-nginx-to-envoy



Let’s do some code

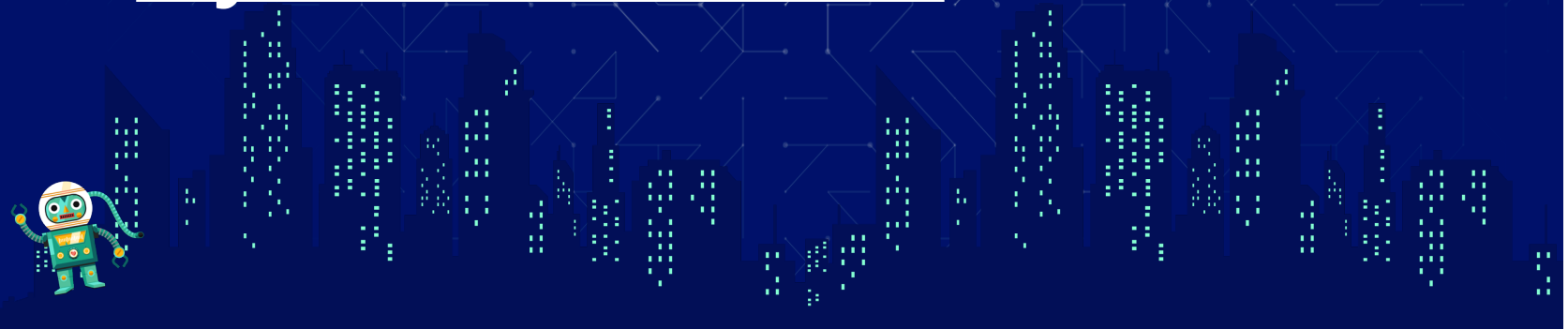


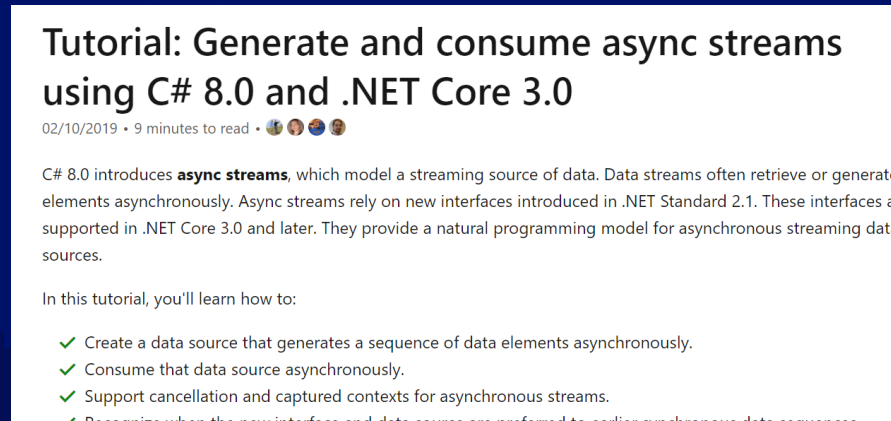
Source https://innerwill.org/

Async Streaming and gRPC with async enumerables IAsyncEnumerable<T> and Span<T>

Generate and consume async streams using C# 8.0 and .NET Core 3.0

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/generate-consume-asynchronous-stream





Tutorial: Generate and consume async streams using C# 8.0 and .NET Core 3.0

02/10/2019 • 9 minutes to read •

C# 8.0 introduces **async streams**, which model a streaming source of data. Data streams often retrieve or generat
elements asynchronously. Async streams rely on new interfaces introduced in .NET Standard 2.1. These interfaces a
supported in .NET Core 3.0 and later. They provide a natural programming model for asynchronous streaming dat
sources.

In this tutorial, you'll learn how to:

- ✓ Create a data source that generates a sequence of data elements asynchronously.
- ✓ Consume that data source asynchronously.
- ✓ Support cancellation and captured contexts for asynchronous streams.

Got questions ?

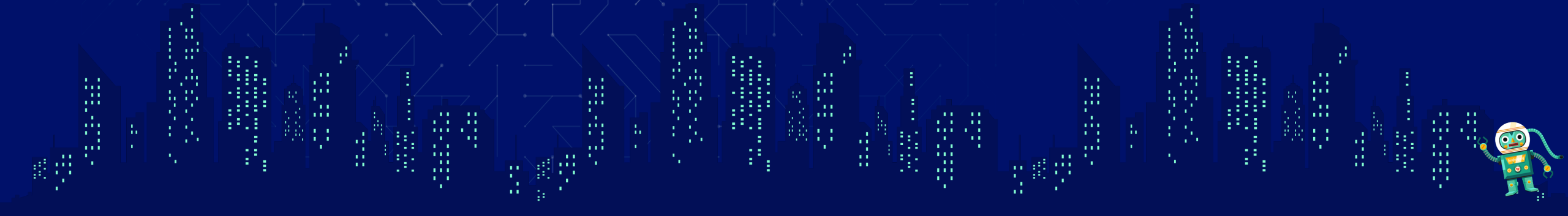
Sassion repo: https://github.com/gsantopaolo/grpc-streaming

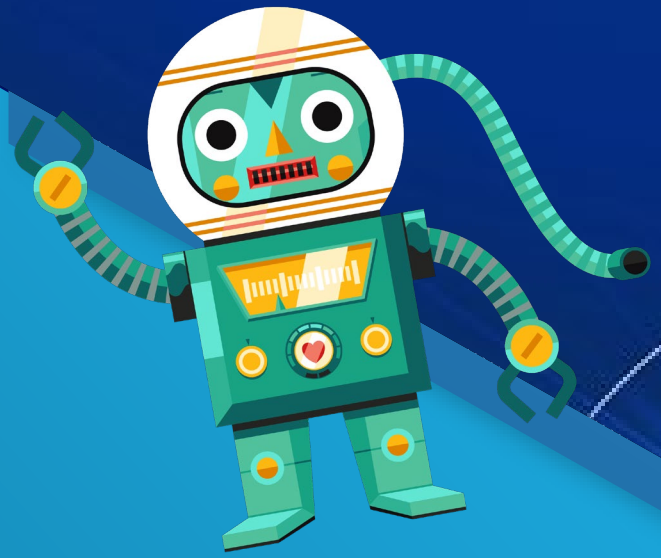
gRPC not supported in App Service and IIS: https://docs.microsoft.com/en-us/aspnet/core/grpc/aspnetcore?view=aspnetcore-3.0&tabs=visual-studio

gRPC-Web for .NET now available: https://devblogs.microsoft.com/aspnet/grpc-web-for-net-now-available/

Async stream: https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/generate-consume-asynchronous-stream

Protobuf Language guide https://developers.google.com/protocol-buffers/docs/proto3





Thank You!

@gsantopaolo

TECHORAMA