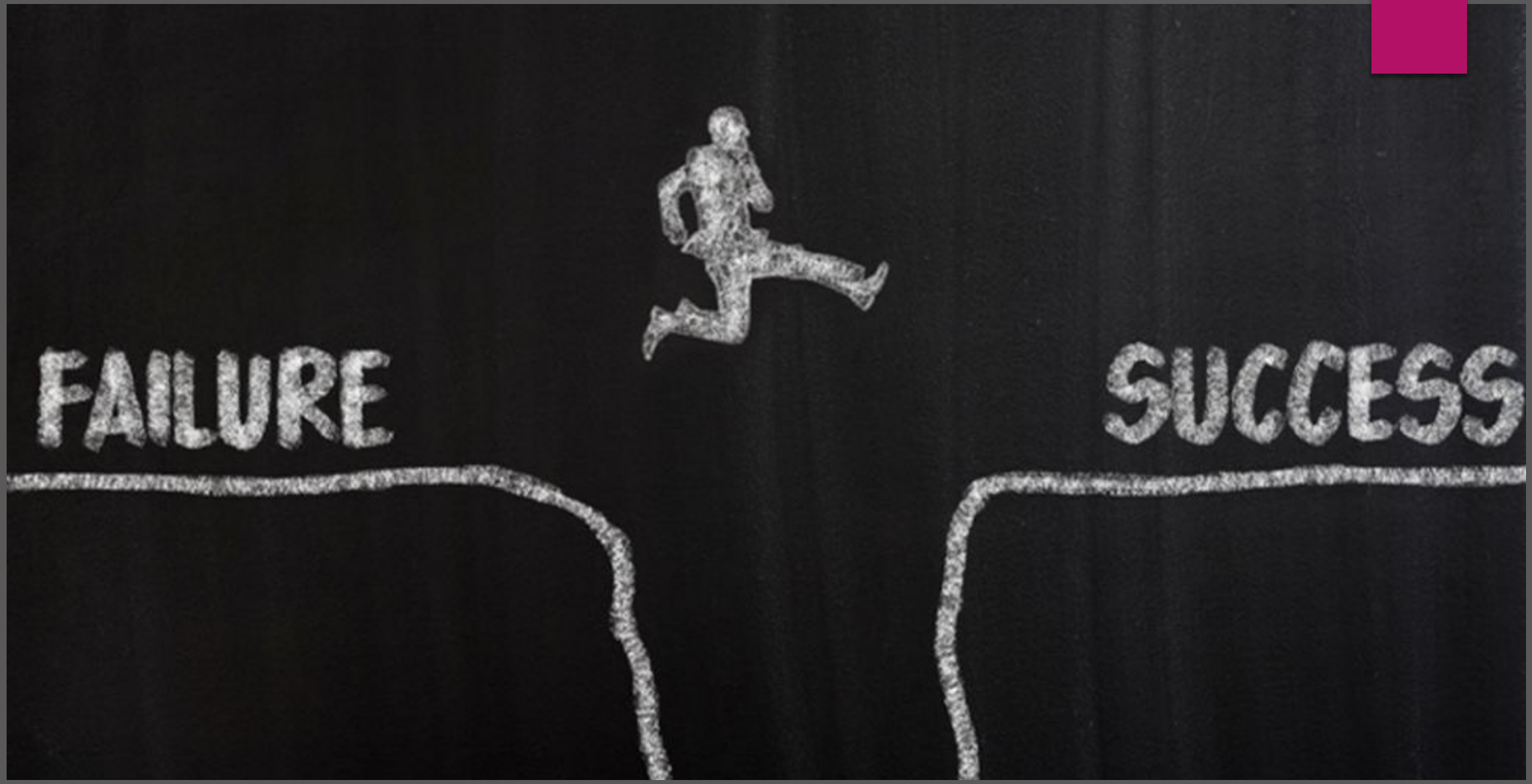
“Da grandi poteri derivano grandi responsabilità”

UGO LATTANZI
@IMPERUGO


FAILURE
SUCCESS

Una buona architettura **è un risultato di gruppo** tra l'architect, il cloud enginer, il data engineer, la security e i developer leaders


4GIFs.com

La “mini” check list

- Deve avere una visione a breve, medio e lungo termine
- Deve essere “pluggabile”
- Non deve premere su un team
- Deve dare il giusto peso alle cose
- Deve chiedersi “si può fare dopo?”
- Posso realizzarla con qualsiasi vendor?

Microservices, più sono micro e meglio sono.

Microservices

- Google: https://github.com/GoogleCloudPlatform/microservices-demo Currency Service 180 righe di codice

```
153  }
154
155  /**
156   * Endpoint for health checks
157   */
158  function check (call, callback) {
159    callback(null, { status: 'SERVING' });
160  }
161
162  /**
163   * Starts an RPC server that receives requests for the
164   * CurrencyConverter service at the sample server port
165   */
166  function main () {
167    logger.info(`Starting gRPC server on port ${PORT}...`);
168    const server = new grpc.Server();
169    server.addService(shopProto.CurrencyService.service, {getSupportedCurrencies, convert});
170    server.addService(healthProto.Health.service, {check});
171
172    server.bindAsync(
173      `0.0.0.0:${PORT}`,
174      grpc.ServerCredentials.createInsecure(),
175      function() {
176        logger.info(`CurrencyService gRPC server started on port ${PORT}`);
177        server.start();
178      },
179     );
180  }
181
182  main();
```

Microservices

The **single-responsibility principle** (**SRP**) is a computer programming principle that states that "A module should be responsible to one, and only one, actor."[1] The term actor refers to a group (consisting of one or more stakeholders or users) that requires a change in the module.

https://en.wikipedia.org/wiki/Single-responsibility_principle

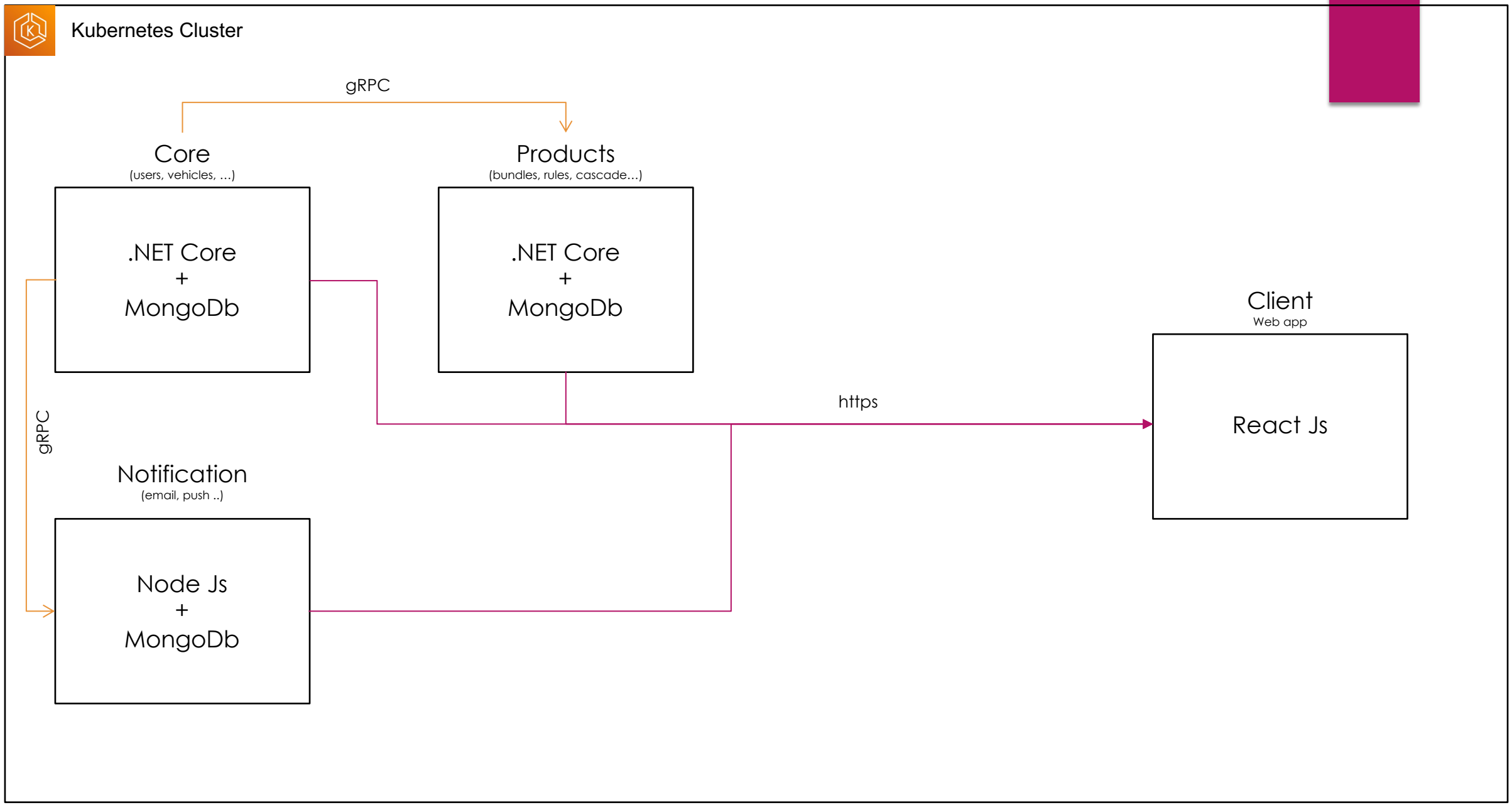
Microservices: “mini” check list

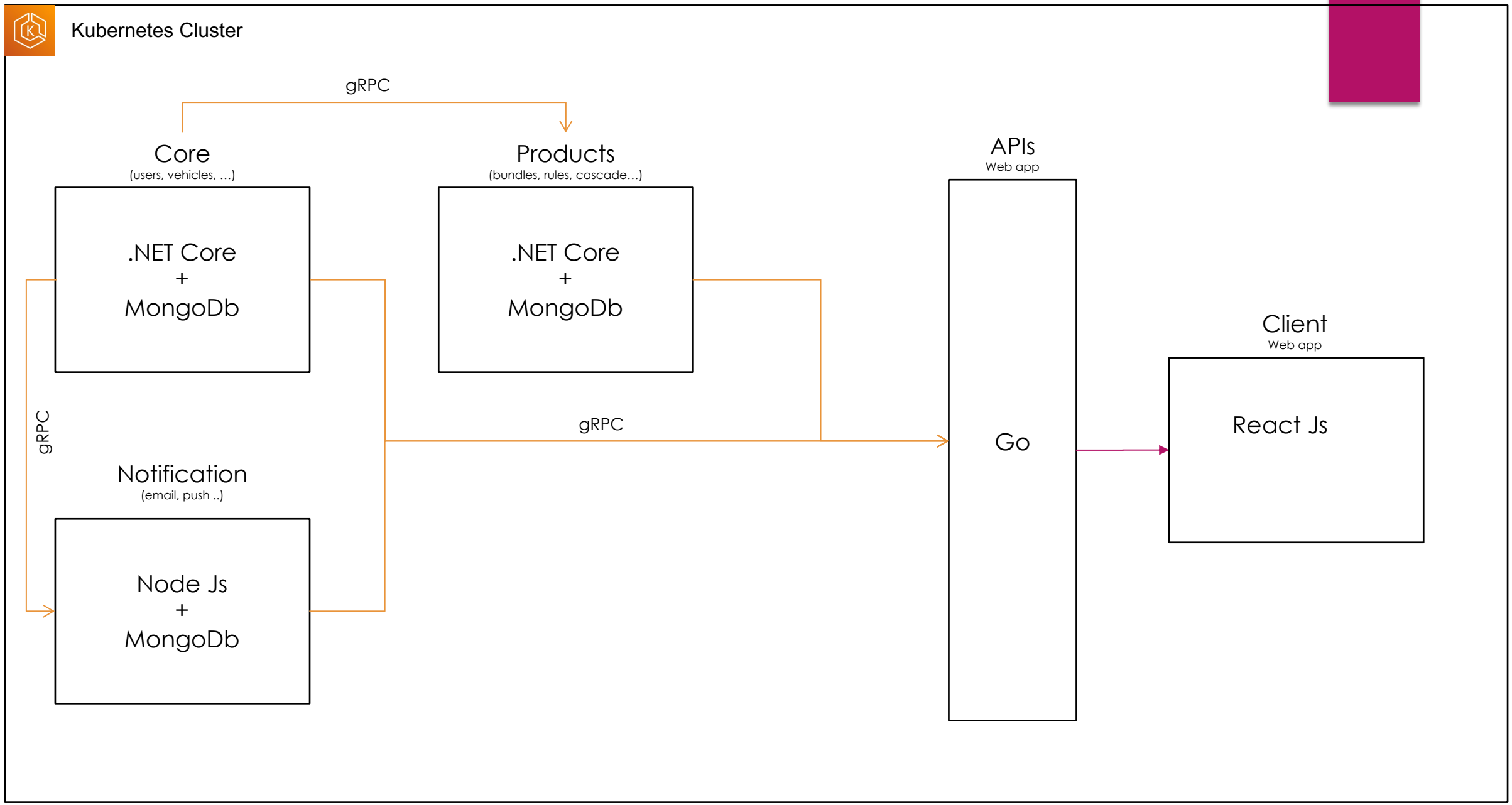
- Un database suo e solo suo
- Uno storage suo e solo suo
- Un’utenza sua e basta
- Espone il minimo indispensabile delle informazioni (meglio se solo in gRPC …)
- Lo può chiamare solo chi realmente autorizzato
- Deve essere piccolo per poter scalare (non solo a livello infrastrutturale)

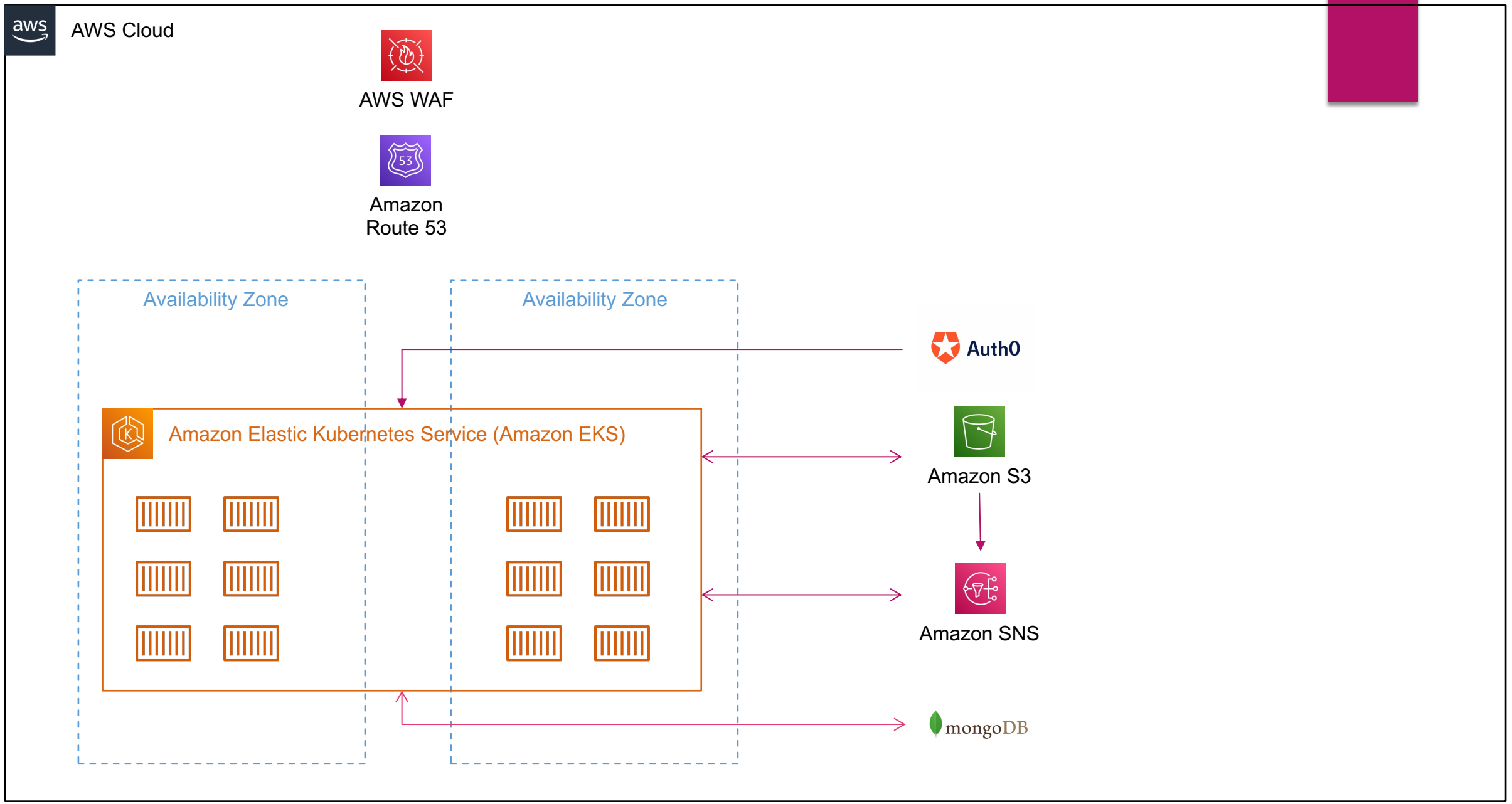
Se vuoi fare il figo solleva eventi.

La “mini” check list

- Il client ha veramente necessità di aspettare che il servizio faccia tutto il necessario prima di potersi sganciare?
- Sicuro che non interessi a nessun il dato che è appena arrivato?


Kubernetes Cluster
gRPC
Core
(users, vehicles, ...)
Products
(bundles, rules, cascade...)
.NET Core
+
MongoDb
.NET Core
+
MongoDb
Client
Web app
https
React Js
gRPC
Notification
(email, push ..)
Node Js
+
MongoDb


Kubernetes Cluster
gRPC
Core
(users, vehicles, ...)
Products
(bundles, rules, cascade...)
APIs
Web app
.NET Core
+
MongoDb
.NET Core
+
MongoDb
Client
Web app
gRPC
gRPC
Notification
(email, push ..)
Go
React Js
Node Js
+
MongoDb


AWS Cloud
AWS WAF
Amazon
Route 53
Availability Zone
Availability Zone
Amazon Elastic Kubernetes Service (Amazon EKS)
Auth0
Amazon S3
Amazon SNS
mongoDB

Kubernetes Cluster

Core
(users, vehicles, ...)

.NET Core
+
Mongo Db
+
Redis

Products
(bundles, rules, cascade...)

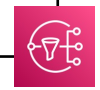
.NET Core
+
Mongo Db
+
Redis

Orders
Users, Products, Budles

.NET Core
+
Mongo Db
+
Redis

WebHook
Users, Products, Budles

Node Js



Amazon SNS

Notification
(email, push ..)

Node Js
+
Mongo Db

Search
Users, Products, Budles

Go Lang
+
Elastic Search

Amazon SNS

ETL
Users, Products, Budles

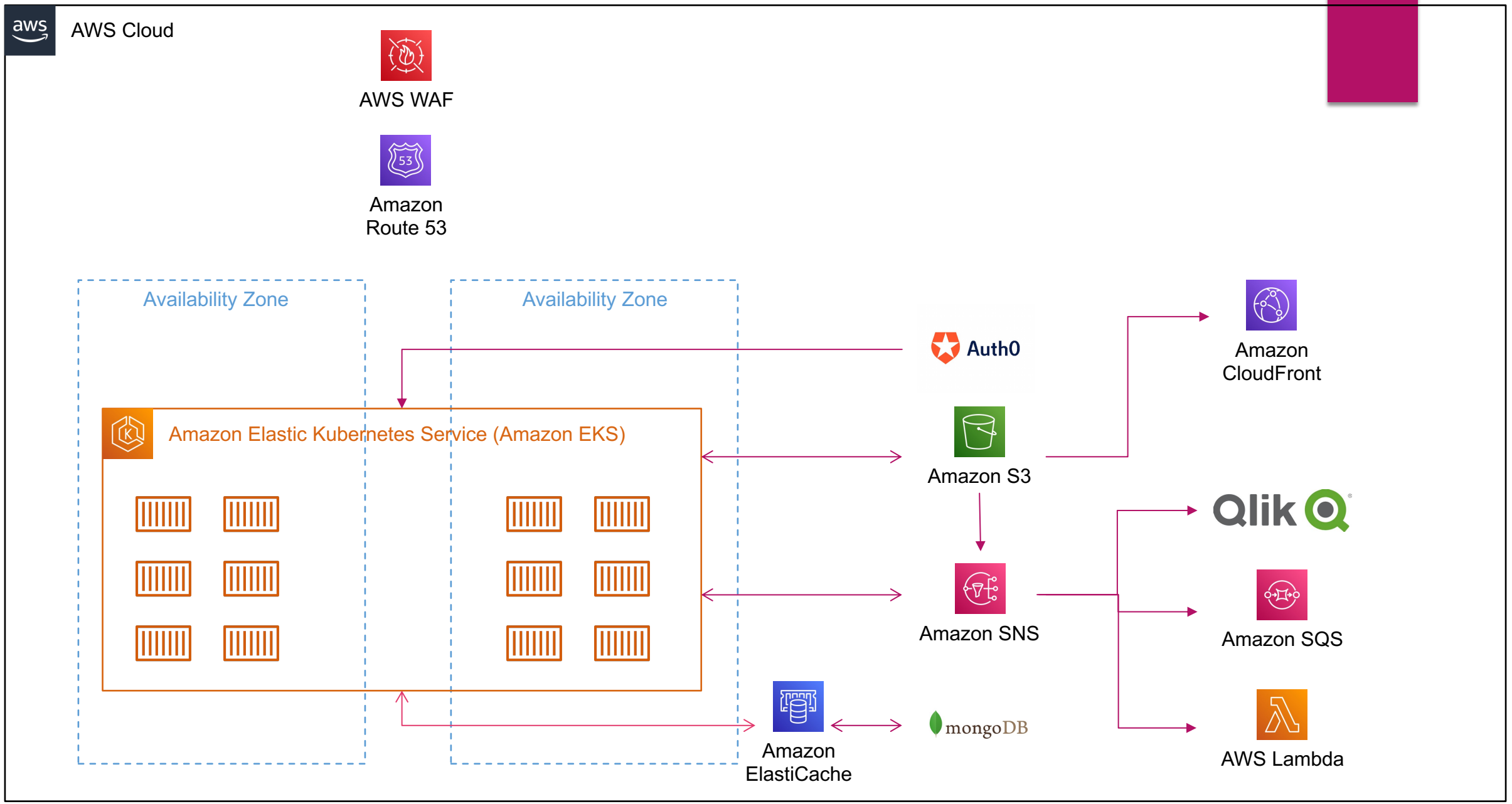
Node Js

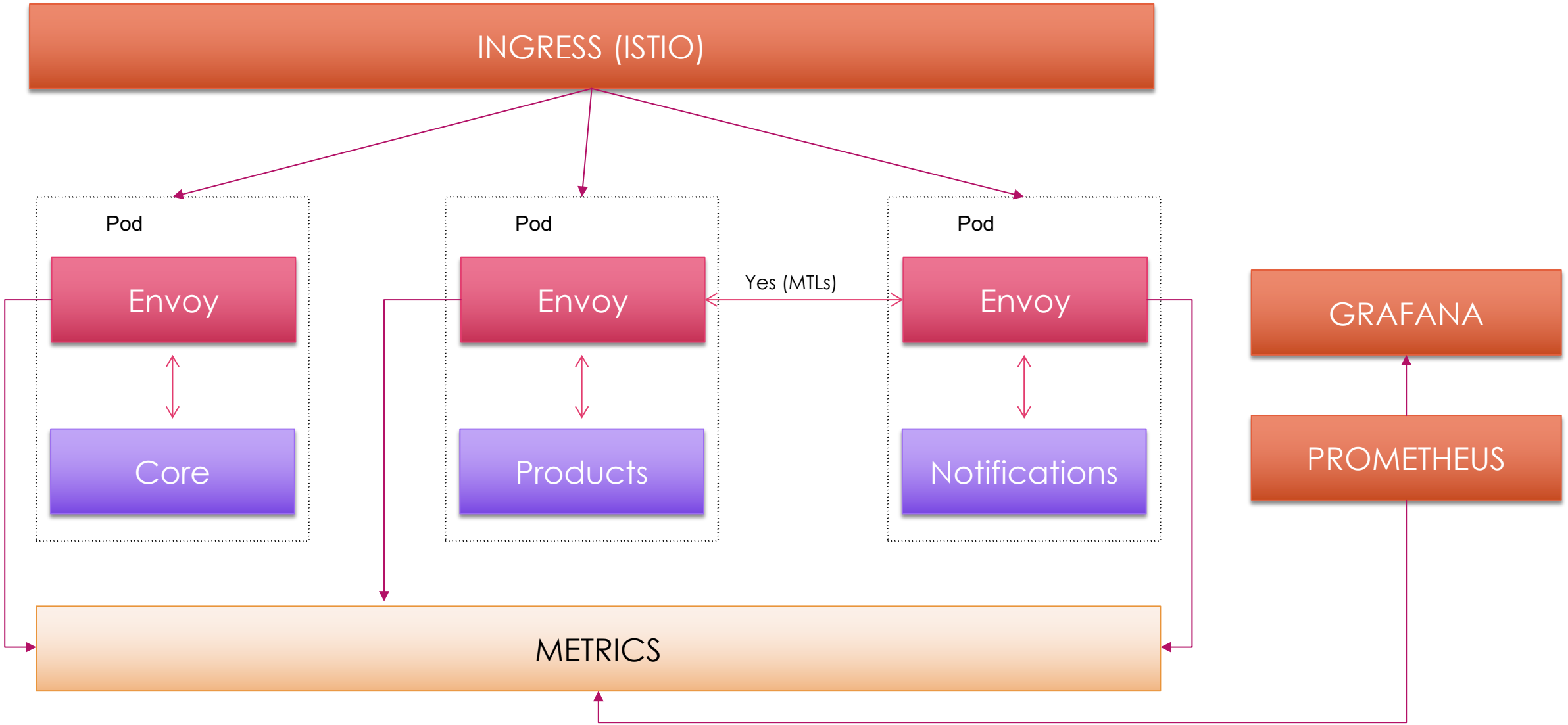
Amazon SNS

Audit
Users, Products, Budles

Node Js
+
Mongo Db

Amazon SNS


AWS Cloud
AWS WAF
Amazon
Route 53
Availability Zone
Availability Zone
Amazon Elastic Kubernetes Service (Amazon EKS)
Auth0
Amazon S3
Amazon SNS
Amazon
ElastiCache
mongoDB
Amazon
CloudFront
Qlik
Amazon SQS
AWS Lambda


INGRESS (ISTIO)
Pod
Envoy
Core
Pod
Envoy
Products
Yes (MTLs)
Pod
Envoy
Notifications
GRAFANA
PROMETHEUS
METRICS