Multi-data center & multi-tenant durable messaging with Apache Pulsar

Ivan Kelly
@ivankelly

What is GDPR?

General Data Protection Regulation

Replaces Directive, mostly stricter

Controls what companies can do with personal data of EU citizens

Hefty fines for violations

Software won’t solve your GDPR problem

GDPR is mostly organizational:

- Need to have a Data Protection Officer
- Need to know what data you have
- Need to know how the data is being used
- Need to know who has access to it

Apache Pulsar

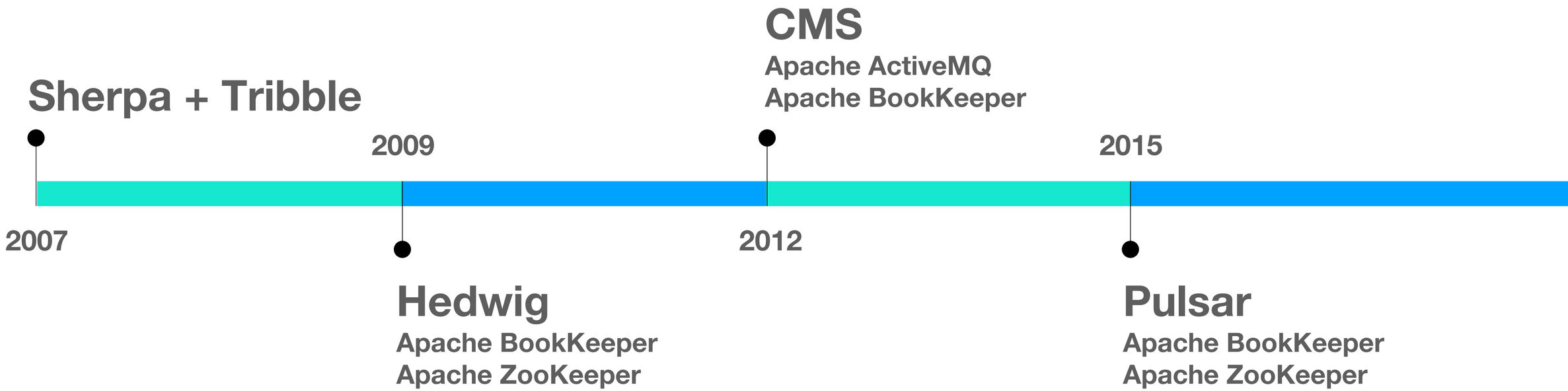
Large scale distributed messaging system

Provides both Queuing and PubSub semantics

Developed by Yahoo, in production for 3 years

Open sourced to Apache Incubator in September 2016

A brief history



Lessons learnt

Some stuff much easier if builtin from start

- Multi tenancy
- Geo replication

Don’t build single points of failure

Avoid direct access to coordination service

Separate different usage patterns

Data protection by design and by default

Who can access data?

What data can they access?

Encryption & Pseudo-anonymization

Authentication in Pulsar

TLS or Athenz*

Based on Role Token concept

Role Tokens

Arbitrary opaque string to identify a role

Built into the TLS cert

Special super user role tokens in broker config

Configuring TLS

Generate a Certificate Authority

```
/usr/lib/ssl/misc/CA.pl pl -newca
```

Generate a key and cert for each broker

```
openssl req -newkey rsa:2048 \
  -sha256 -nodes \
  -out broker-cert.csr -outform PEM
openssl pkcs8 -topk8 -nocrypt \
  -inform PEM -outform PEM \
  -in privkey.pem -out broker-key.pem
```

Sign cert with CA

```
openssl ca -out broker-cert.pem \
  -infiles broker-cert.csr
```

Configure in broker.conf

```
tlsEnabled=false
tlsCertificateFilePath=/path/to/cacert.pem
tlsKeyFilePath=/path/to/broker-key.pem
tlsTrustCertsFilePath=/path/to/broker-cert.pem
```

Configuring TLS Authentication

For each role

- Generate key and certificate, specifying Role Token as common name
- Sign each key with Certificate Authority

Configure in client

```
PulsarClient client = PulsarClient.builder()
    .serviceUrl(SERVICE_URL)
    .authentication("org.apache.pulsar.client.impl.auth.AuthenticationTls",
                    ImmutableMap.of("tlsCertFile", “/path/to/role-cert.pem”,
                                    "tlsKeyFile", “/path/to/role-key.pem”)
    .build();
```

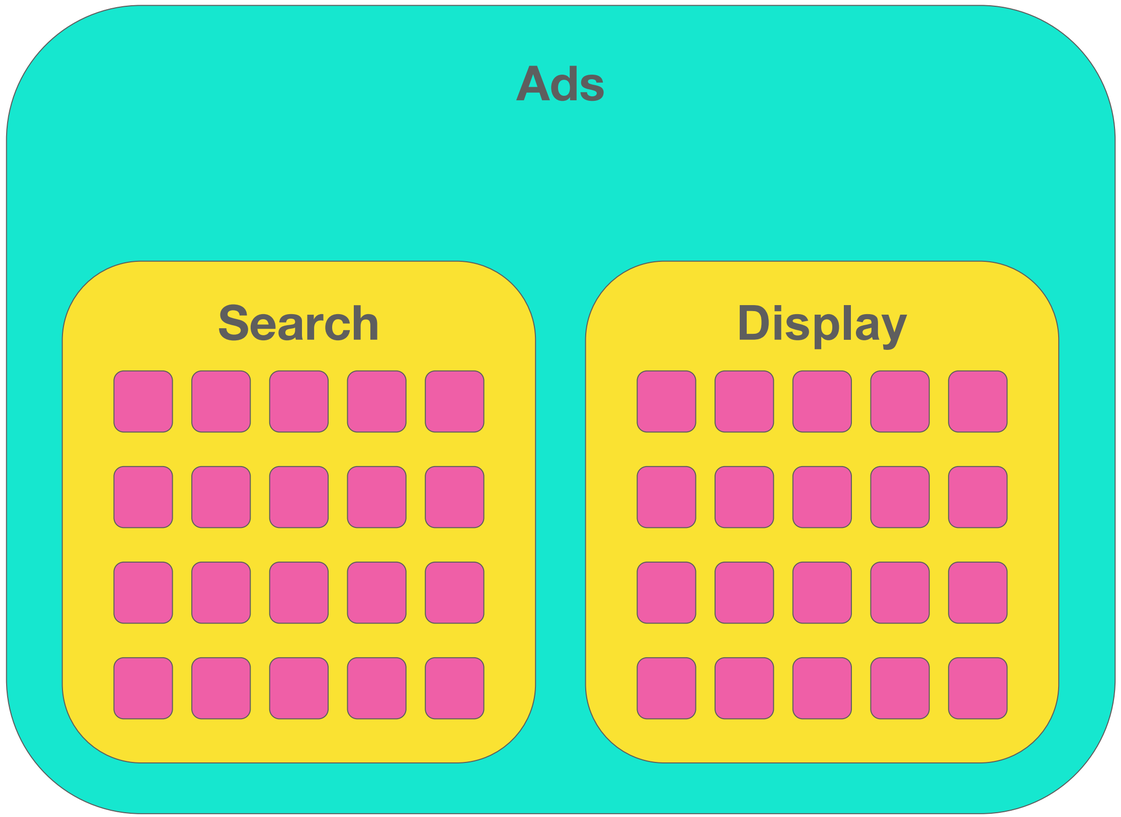
Authentication != Authorization

Authentication alone allows you to publish/consume from all topics

Authorization required to control fine grain access

Pulsar data model facilitates authorization

Pulsar Data Model



Tenants

Created by superuser
Admin role assigned on creation

Namespaces

Administered by admin role
ACLs at this level

Topics

Put messages in these

Enable authorization in broker

authorizationEnabled=true

superUserRoles=admin

Setup tenant with authorization

As superuser

```
pulsar-admin tenants create --clusters test \
    --admin-role strata.admin strata-tenant
```

As strata.admin

```
pulsar-admin namespaces create \
    --clusters test strata-tenant/demo

pulsar-admin namespaces grant-permission \
    --role strata.user.foobar \
    --actions produce strata-tenant/demo
```

As strata.user.foobar

```
pulsar-client produce \
    -m foo -n 1 \
    persistent://strata-tenant/demo/topic1
```

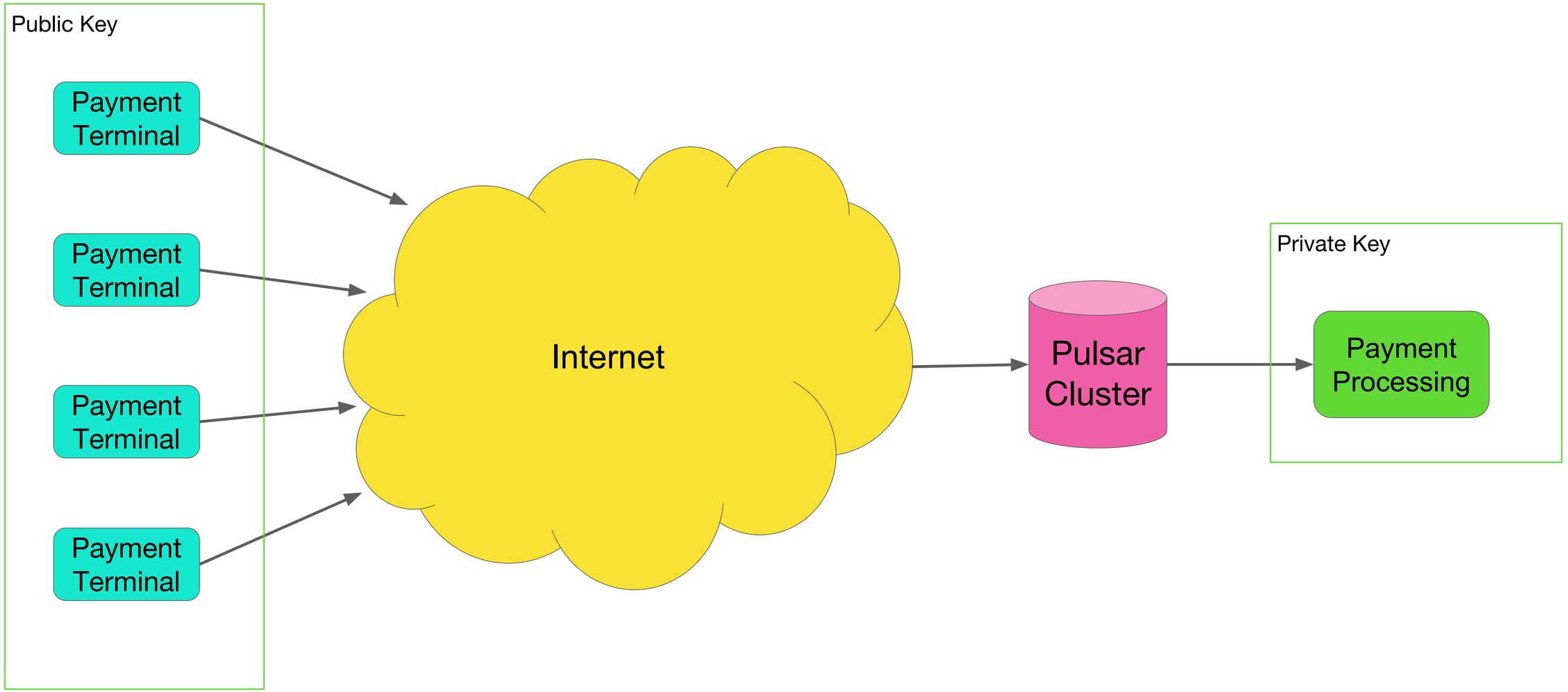
End-to-End encryption

Sometimes we want to limit access to only the final recipient

Pulsar provides end-to-end encryption:

- Producer encrypts with public key
- Consumer decrypts with private key

Use case: End-to-end encryption



Using End-to-End encryption

Implement one interface

```
public interface CryptoKeyReader {
    EncryptionKeyInfo getPublicKey(String keyName, Map<String, String> metadata);
    EncryptionKeyInfo getPrivateKey(String keyName, Map<String, String> metadata);
}
```

Configure the producer

```
client.newProducer()
    .topic("foobar")
    .cryptoKeyReader(new MyCryptoKeyReader())
    .addEncryptionKey(keyName).create()
```

Configure the consumer

```
client.newConsumer()
    .topic("foobar").subscriptionName("sub1")
    .cyptoKeyReader(new MyCryptoKeyReader())
    .subscribe()
```

Pseudo anonymization

Depersonalize the data as early as possible

Process the data before long term storage

Pulsar provides Pulsar Functions

Pulsar Functions

Lightweight stream processing

New in Pulsar 2.0

Currently supports Java and Python

Java

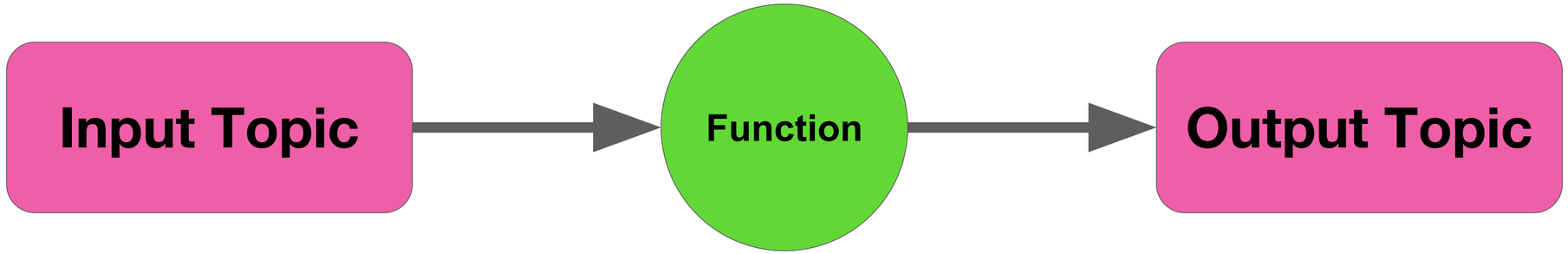
```java
import java.util.function.Function;

public class Anon implements Function<String,String> {
    @Override
    public String apply(String input) {
        return input.replace("ivan", "anonymous");
    }
}
```

Python

```python
def process(input):
    return input.replace("ivan", "anonymous")
```

Pulsar Functions



Java

```
pulsar-admin functions create --jar anon.jar --className Anon \
    --fqfn strata-tenant/demo/anonymize \
    --inputs persistent://strata-tenant/demo/input \
    --output persistent://strata-tenant/demo/output
```

Python

```
pulsar-admin functions create --py anon.py --className anon \
    --fqfn strata-tenant/demo/anonymize \
    --inputs persistent://strata-tenant/demo/input \
    --output persistent://strata-tenant/demo/output
```

Transfer of data internationally

GDPR allows transfer of data across borders

But only to countries whose data protections are deems adequate

Currently adequate territories

Andorra

Argentina

Canada

Faroe Islands

Guernsey

Israel

Isle of Man

Jersey

New Zealand

Switzerland

Uruguay

United States

Use case: Geo replication

3 data centres

- US
- Ireland
- Singapore

2 data sets

- EU user data
- non-EU user data

Create tenant that CAN have data in 3 places

```
pulsar-admin tenants create \
    --allowed-clusters us,ie,sg \
    --role-admin strata.admin geo-repl-tenant
```

Create namespace for each data set

```
pulsar-admin namespaces create \
    --clusters us,ie geo-repl-tenant/eu-users
pulsar-admin namespaces create \
    --clusters us,ie,sg geo-repl-tenant/non-eu-users
```

Geo replication in pulsar

Facilitated by data model

Managed at both tenant and namespace levels

@streamlio @ivankelly

Blog: https://streaml.io/blog

YouTube: https://goo.gl/qnWXBT