APACHE NIFI COM POSTGRESQL

Behind the scenes

PALESTRANTES

Gerdan Rezende dos Santos

Brasília - DF

Tecnisys Tecnologias Inovadoras

Especialista em Banco de Dados e Big Data

Ativo em várias comunidades de software livre, sendo um apaixonado por tecnologias open source

Revisor de funcionalidades do PostgreSQL e desenvolvedor/contribuidor do projeto Apache NiFi

Organizador e palestrante do PGDAY Brasília, já palestrou em eventos da IBM, Latinoware, PGDAY Brasília, entre outros

https://www.linkedin.com/in/gerdan

Davy Alvarenga Machado

Brasília - DF

Tecnisys Tecnologias Inovadoras

Interessado em dados de todos os tamanhos

Desenvolvedor Java e Android nas horas vagas e curioso por natureza

Revisor de funcionalidades do PostgreSQL e desenvolvedor/contribuidor do projeto Apache NiFi

Organizador e palestrante do PGDay Brasília

Editor e escritor do blog Sketchtease

https://br.linkedin.com/in/davy-alvarenga-machado

TECNISYS

Referência nacional em soluções open source atuando no mercado de TI a mais de 25 anos

Parceira de empresas líderes internacionalmente, tais como, EnterpriseDB, Hortonworks, Red Hat, SUSE, entre outras

Especialistas em Banco de Dados, Big Data, Sistemas Operacionais, Middlewares, entre outras áreas

Principal via estratégica de mercado da EnterpriseDB para o território brasileiro

Suporte ao PostgreSQL 24x7, com atendimento inicial em menos de 15 min via 0800, site ou e-mail

AGENDA

- HADOOP, DATA FLOW E APACHE NIFI
- CAPTURE DATA CHANGE (CDC)
- POSTGRESQL REPLICAÇÃO LÓGICA
- PROCESSADOR CAPTURECHANGEPOSTGRESQL
- LOGICAL REPLICATION API
- COPY API
- LIMITAÇÕES DA REPLICAÇÃO LÓGICA

APACHE HADOOP

Coleção de open source software frameworks para armazenamento e processamento distribuído de grandes conjuntos de dados

Escalável e tolerante a falhas

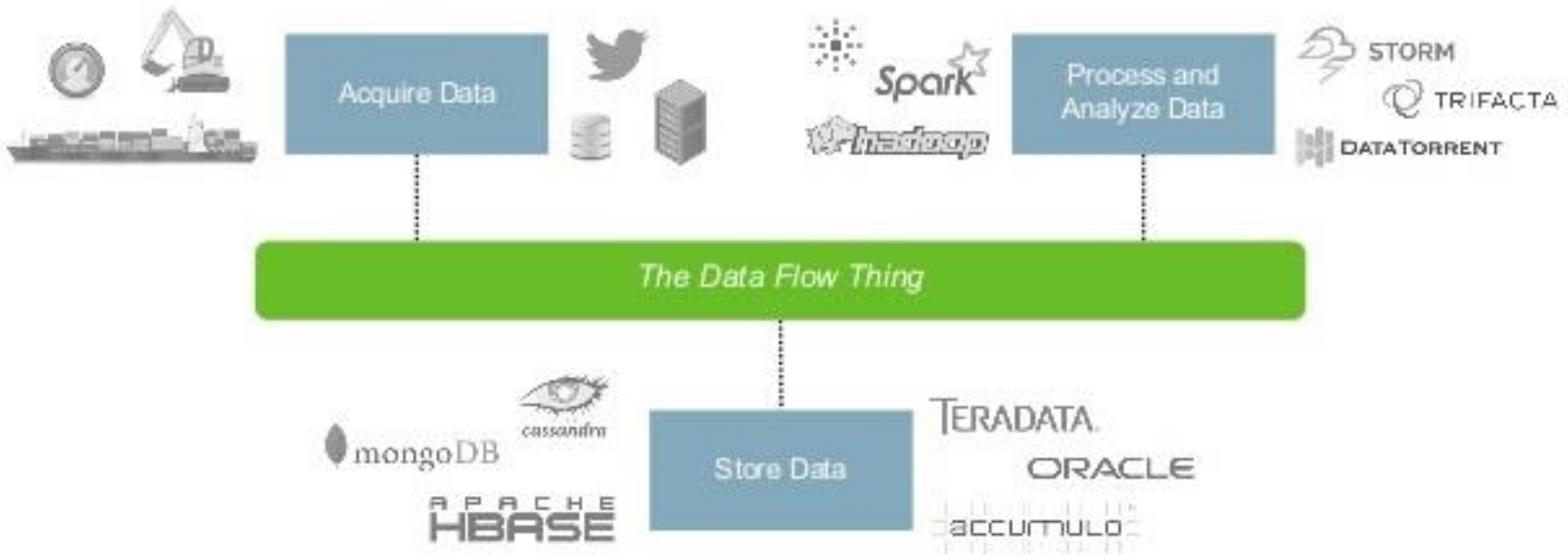
Hardware commodity

Processa todos os tipos de Big Data (all/semi/un structured)

Comunidade

Apache Software Foundation (ASF)

DATA FLOW

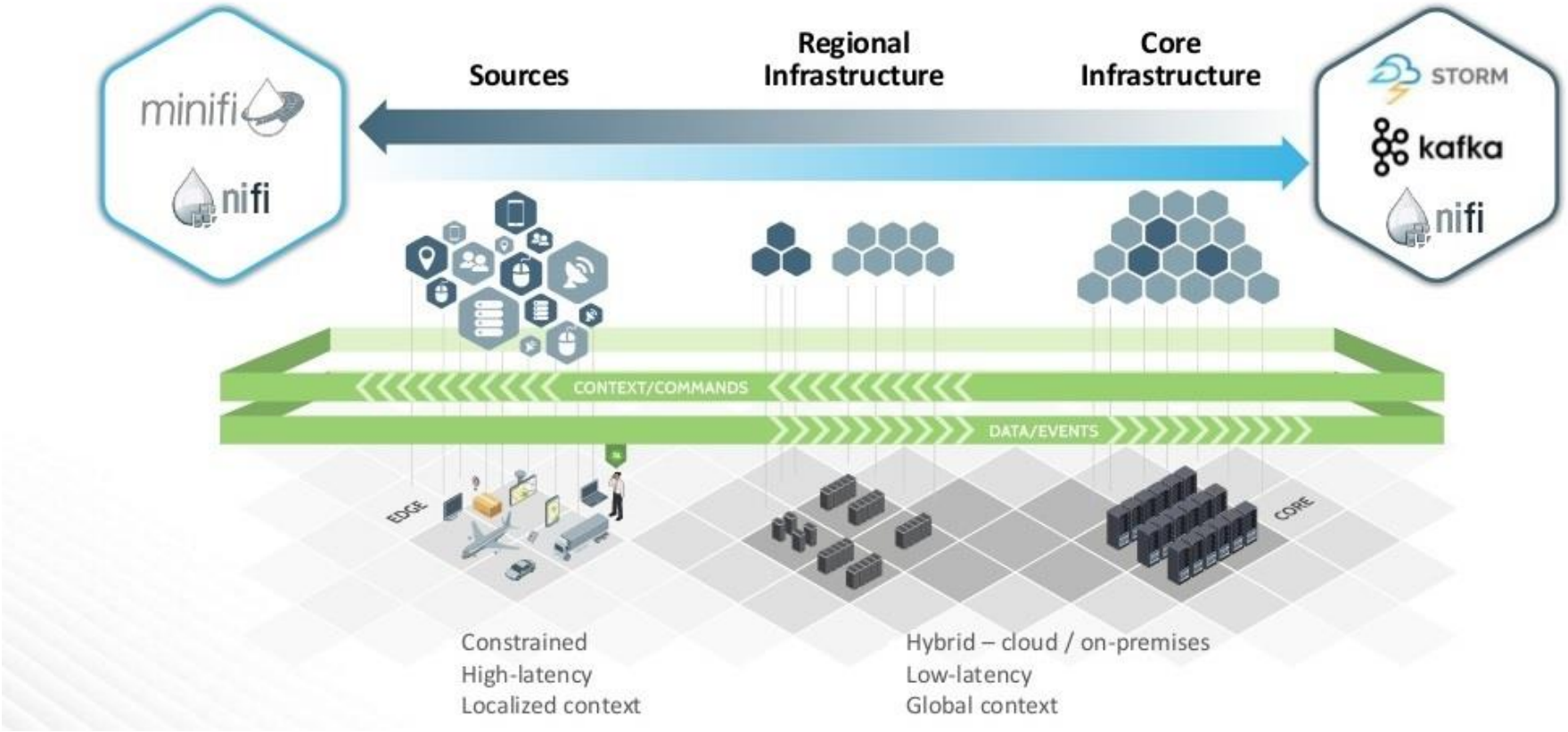

Acquire Data
Spark
Hadoop
Process and Analyze Data
STORM
TRIFACTA
DATATORRENT
The Data Flow Thing
mongoDB
cassandra
APACHE HBASE
Store Data
TERADATA
ORACLE
accumulo

DATA FLOW

It’s not just how quickly you move data – it’s about how quickly you can change behavior and seize new opportunities

Hortonworks Docs

APACHE NIFI


minifi
nifi
Sources
Regional Infrastructure
Core Infrastructure
STORM
kafka
nifi
CONTEXT/COMMANDS
DATA/EVENTS
EDGE
CORE
Constrained
High-latency
Localized context
Hybrid – cloud / on-premises
Low-latency
Global context

APACHE NIFI

UMA BREVE HISTÓRIA

2006

NiagaraFiles (NiFi) foi desenvolvido pela Agência de Segurança Nacional dos Estados Unidos (NSA)

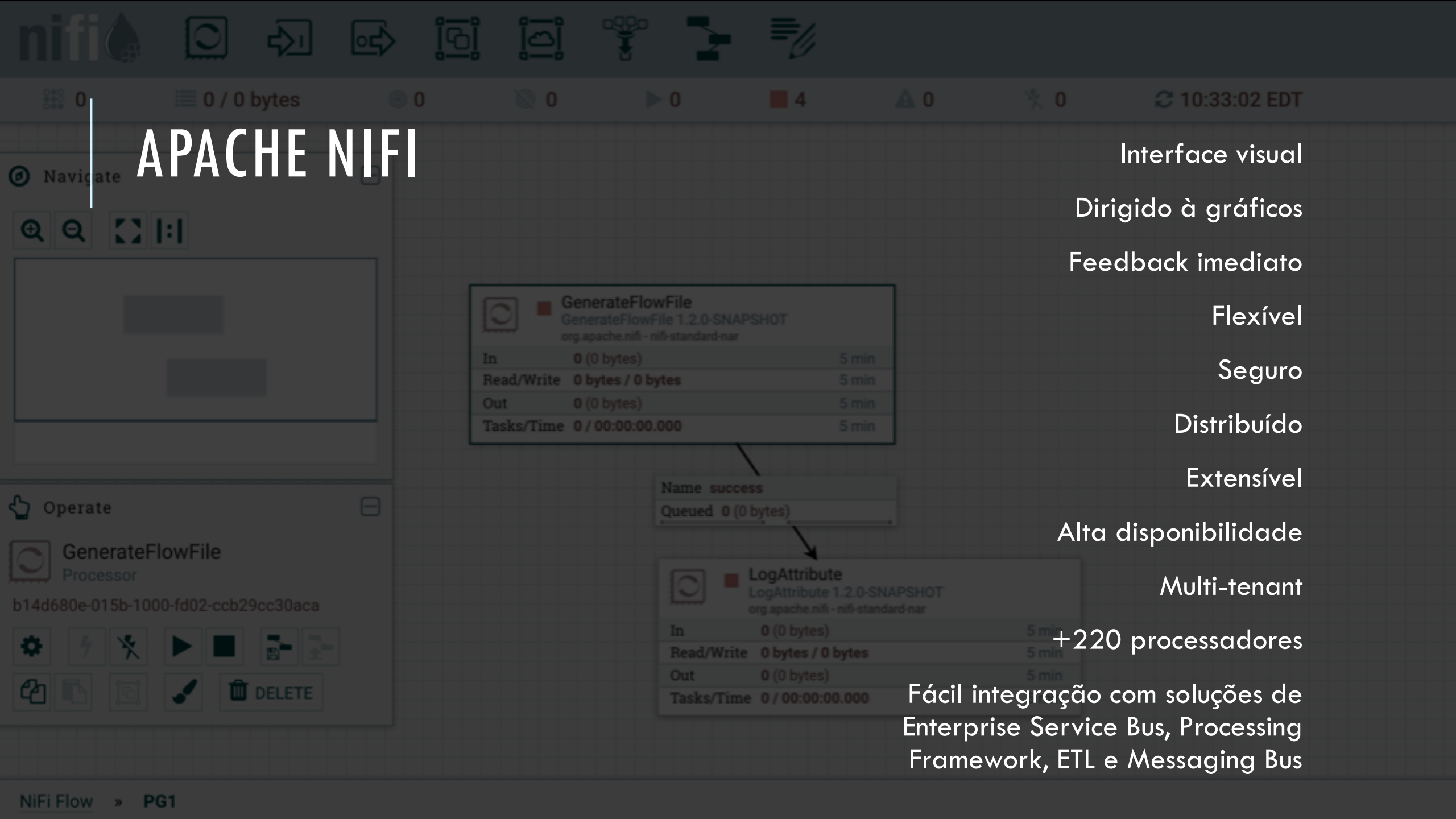
2014

NiFi foi doado para a Apache Software Foundation (ASF)

2015

NiFi alcança o status de projeto top-level na ASF

APACHE NIFI



Interface visual

Dirigido à gráficos

Feedback imediato

Flexível

Seguro

Distribuído

Extensível

Alta disponibilidade

Multi-tenant

+220 processadores

Fácil integração com soluções de Enterprise Service Bus, Processing Framework, ETL e Messaging Bus

CAPTURE DATA CHANGE (CDC)

Categoria de processadores NiFi responsáveis pela captura de eventos de alteração de um banco de dados em tempo real

Baseado em logs transacionais (não em triggers)

Captura single-thread, Processamento multi-thread

Ordenação dos eventos (EnforceOrder)

Diversas aplicações

Até o momento, temos apenas o processador CaptureChangeMySQL

POSTGRESQL
REPLICAÇÃO LÓGICA

Modelo Publish/Subscribe

Eventos publicados em tempo real

Eventos aplicados na ordem de confirmação (COMMIT)

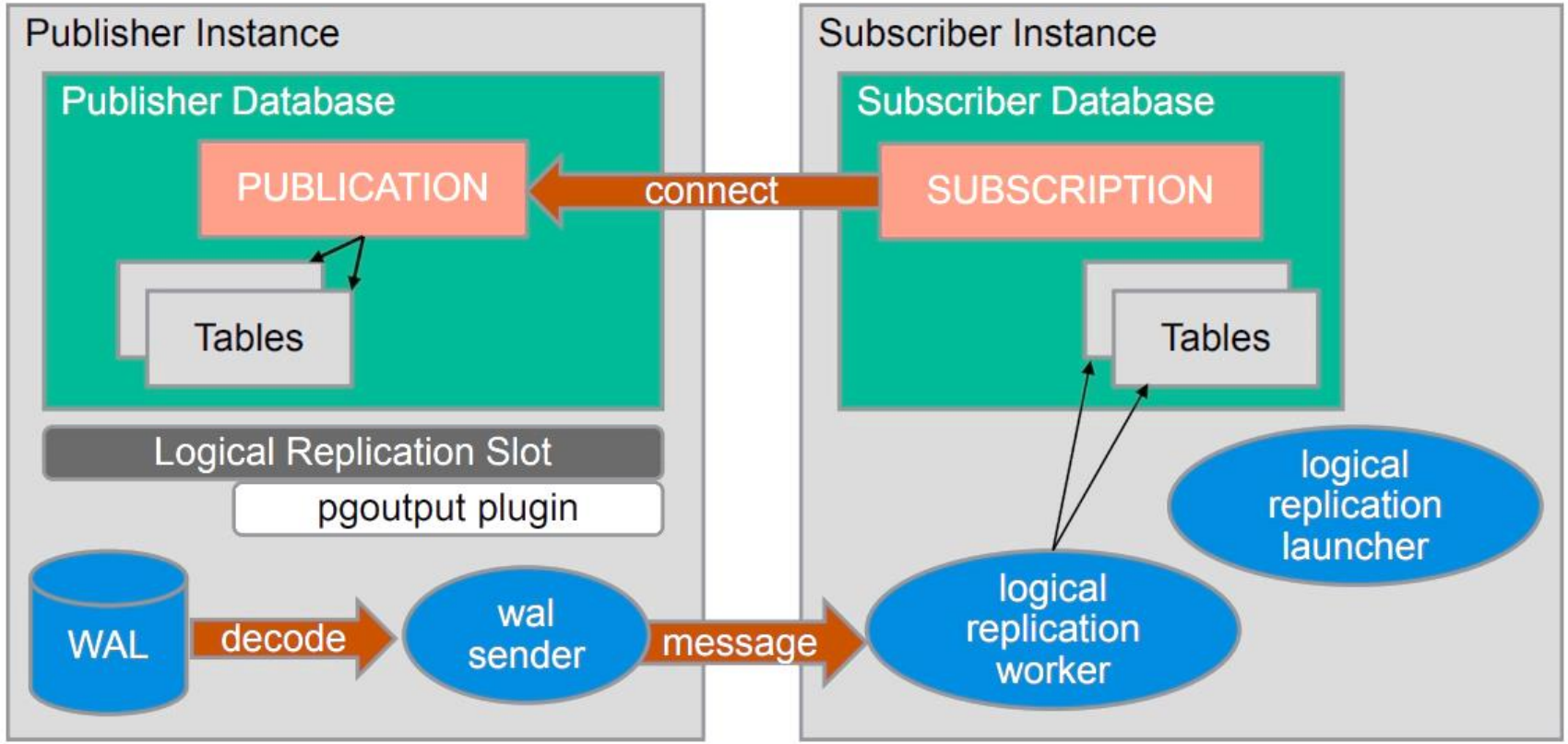
REPLICA IDENTITY (DEFAULT, USING INDEX, FULL e NOTHING)

Slot de Replicação
- Um para cada subscrição (SUBSCRIPTION)
- Nome da subscrição por padrão

pgoutput
- Plugin de decodificação lógica padrão

Replicação síncrona e assíncrona

REPLICAÇÃO LÓGICA
ARQUITETURA



REPLICAÇÃO LÓGICA
ARQUITETURA

walsender no publisher inicia a decodificação lógica do WAL

pgoutput transforma e filtra os registros transacionais

Protocolo de Replicação
- Parâmetros e formato das mensagens

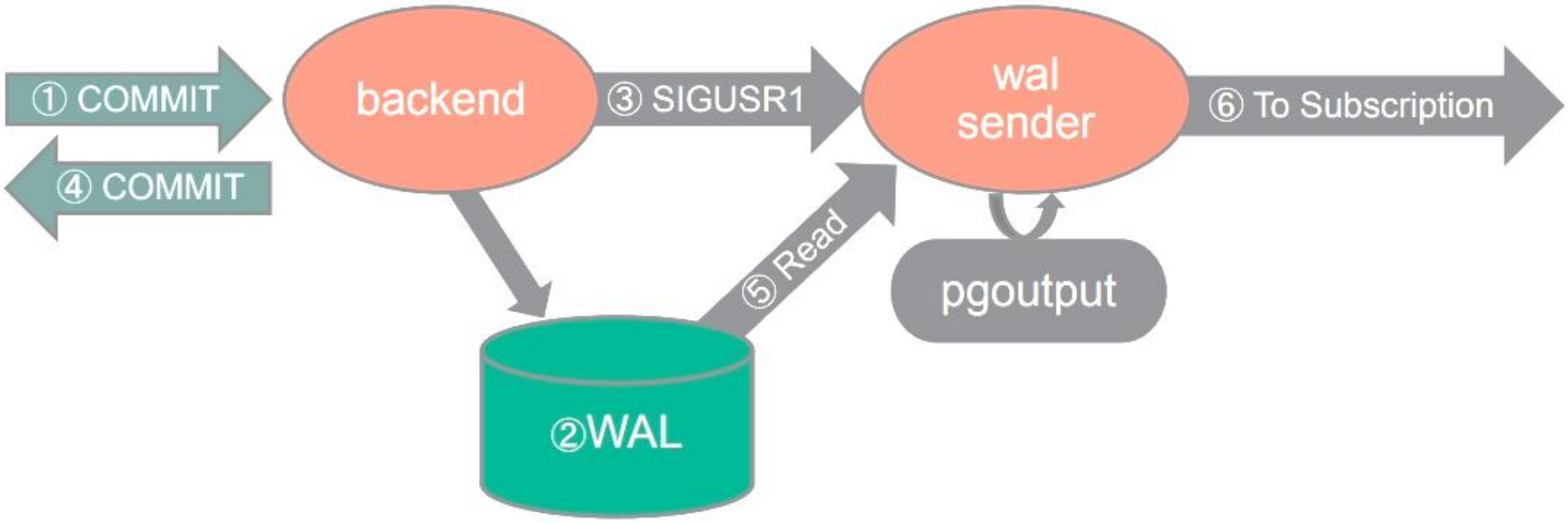
Protocolo de Streaming de Replicação
- Transferência contínua

bgworker no subscriber aplica as alterações

Sincronização inicial (snapshot) semelhante a um COPY

REPLICAÇÃO LÓGICA

REPLICAÇÃO ASSÍNCRONA



REPLICAÇÃO LÓGICA
CONFIGURAÇÃO

Publisher Cluster

wal_level = logical

max_replication_slots = pelo menos o número de assinantes esperados mais uma reserva para sincronização

max_wal_senders = pelo menos, max_replication_slots mais o número de réplica que se conectam ao mesmo tempo no cluster

Subscriber Cluster

max_logical_replication_workers = pelo menos o número de assinaturas que serão criadas mais uma reserva para sincronização

max_worker_processes = pelo menos max_logical_replication_workers + 1

REPLICAÇÃO LÓGICA

PUBLICATIONS E SUBSCRIPTIONS

```
CREATE PUBLICATION pub_cidade FOR TABLE cidade;
```

```
CREATE SUBSCRIPTION sub_cidade CONNECTION
        'host=IP_OU_HOSTNAME_DO_PUBLICADOR port=5432
        dbname=teste user=repuser' PUBLICATION pub_cidade;
```

REPLICAÇÃO LÓGICA
MONITORAMENTO

Visões

pg_replication_slots
: slots de replicação, plugin, tipo, temporário, estado, xmin

pg_stat_replication
: wal sender, informações do cliente, estado, último lsn enviado/flushed/escrito/aplicado

pg_stat_subscription
: nome da sub, pid do worker, último lsn recebido, último lsn reportado

pg_publication/pg_publication_rel
: nome da pub, FOR ALL TABLES/table OID, publication OID

pg_subscription
: informações de conexão, nome do slot de replicação, nome da pub

CAPTURECHANGEPOSTGRESQL

ASF JIRA NIFI-4239

Desenvolvido em Java

Logical Replication API

Copy API (sincronização inicial)

Eventos são retornados como flowfiles em formato JSON

cdc.sequence.id, cdc.event.type e mime.type

Em fase de ajustes, melhorias e testes

LOGICAL REPLICATION API

CRIANDO UM CONEXÃO DE REPLICAÇÃO

```
String url = "jdbc:postgresql://localhost:5432/test";

Properties props = new Properties();
PGProperty.USER.set(props, "postgres");
PGProperty.PASSWORD.set(props, "postgres");
PGProperty.ASSUME_MIN_SERVER_VERSION.set(props, "10");
PGProperty.REPLICATION.set(props, "database");
PGProperty.PREFER_QUERY_MODE.set(props, "simple");

Class.forName("org.postgresql.Driver");
Connection con = DriverManager.getConnection(url, props);

PGConnection replConnection = con.unwrap(PGConnection.class);
```

LOGICAL REPLICATION API

CRIANDO UM SLOT DE REPLICAÇÃO

```
public void createReplicationSlot() throws SQLException {
     PGConnection pgcon = this.connection.unwrap(PGConnection.class);

     pgcon.getReplicationAPI()
          .createReplicationSlot()
          .logical()
          .withSlotName(this.slotName)
          .withOutputPlugin("pgoutput")
          .make();
}
```

LOGICAL REPLICATION API

CRIANDO UM STREAM DE REPLICAÇÃO

```
public PGReplicationStream getReplicationStream() throws SQLException {
    PGConnection pgcon = this.start.connection.unwrap(PGConnection.class);

    return pgcon.getReplicationAPI()
        .replicationStream()
        .logical()
        .withSlotName(this.start.slotName)
        .withSlotOption("proto_version", "1")
        .withSlotOption("publication_names", this.start.publicationName)
        .withStatusInterval(20, TimeUnit.SECONDS)
        .start();
}
```

LOGICAL REPLICATION API

LENDO UM STREAM DE REPLICAÇÃO EM TEMPO REAL

```
public void readPendingStream() throws SQLException, InterruptedException, ParseException, UnsupportedEncodingException {

	Decode decode = new Decode();
	if(Decode.dataTypes.isEmpty())
		decode.loadDataTypes(this.start.connection);

	PGReplicationStream stream = getReplicationStream();

	while (true) {
		ByteBuffer buffer = stream.readPending();

		if (buffer == null) {
		  TimeUnit.MILLISECONDS.sleep(10L);
		  continue;
		}

		JSONObject json = new JSONObject();

		if(this.start.messageQueuePretty) {
			this.start.messageQueue.addLast(decode.decodeLogicalReplicationMessagePretty(buffer, json).toJSONString());
		} else {
			this.start.messageQueue.addLast(decode.decodeLogicalReplicationMessage(buffer, json).toJSONString().replace("\\\"", "\""));
		}

		/* Feedback */
		stream.setAppliedLSN(stream.getLastReceiveLSN());
		stream.setFlushedLSN(stream.getLastReceiveLSN());
	}
}
```

LOGICAL REPLICATION API

CARREGANDO TIPOS DE DADOS

```
public void loadDataTypes(Connection connection) throws SQLException {

	connection.setAutoCommit(true);
	PreparedStatement stmt = connection.prepareStatement("SELECT oid, typname FROM pg_catalog.pg_type");

	ResultSet rs = stmt.executeQuery();

	while(rs.next()) {
		Decode.dataTypes.put(rs.getInt(1), rs.getString(2));
	}

	rs.close();
	stmt.close();
}
```

LOGICAL REPLICATION API

DECODIFICANDO UMA MENSAGEM

```
public class Decode {

      protected static HashMap<Integer, String> dataTypes = new HashMap<>();
      protected HashMap<Integer, Relation> relations = new HashMap<>();

      public JSONObject decodeLogicalReplicationMessage(ByteBuffer buffer, JSONObject json) throws ParseException, SQLException,
            UnsupportedEncodingException {

            char msgType = (char) buffer.get(0);     /* (Byte1) Identifies the message as a begin message. */
            int position = 1;

            switch (msgType) {
            case 'B':                                /* Identifies the message as a begin message.*/

                        JSONObject jsonMessage_B = new JSONObject();

                        jsonMessage_B.put("xLSNFinal", buffer.getLong(1));              /* (Int64) The final LSN of the transaction. */
                        jsonMessage_B.put("xCommitTime", getFormattedPostgreSQLEpochDate(buffer.getLong(9)));
                                                                                        /* (Int64) Commit timestamp of the transaction.
                                                                                              The value is in microseconds since
                                                                                              PostgreSQL epoch (2000-01-01). */
                        jsonMessage_B.put("xid", buffer.getInt(17));                    /* (Int32) Xid of the transaction. */

                        json.put("begin", jsonMessage_B);
                        return json;

                        ...
```

COPY API

REALIZANDO A SINCRONIZAÇÃO INICIAL

```
public void getInitialSnapshot() throws SQLException, IOException {

     ArrayList<String> pubTables = this.getPublicationTables();
     JSONObject jsonSnapshot = new JSONObject();

     for (String table : pubTables) {
          ArrayList<String> lines = this.getInitialSnapshotTable(table);
          JSONArray tableLines = new JSONArray();

          for (String line : lines) {
               tableLines.add(line);
          }

          jsonSnapshot.put(table, tableLines);
     }

     this.start.messageQueue.addFirst("{\"snaphost\":" + jsonSnapshot.toJSONString().replace("\\\"", "\"") + "}");
}
```

COPY API

REALIZANDO A SINCRONIZAÇÃO INICIAL

```
public ArrayList<String> getInitialSnapshotTable(String tableName) throws SQLException, IOException {

    PGConnection pgcon = this.start.connection.unwrap(PGConnection.class);
    ByteArrayOutputStream out = new ByteArrayOutputStream();

    CopyManager manager = pgcon.getCopyAPI();
    manager.copyOut("COPY (SELECT REGEXP_REPLACE(ROW_TO_JSON(t)::TEXT, '\\\\', '\\', 'g')
        FROM (SELECT * FROM " + tableName + ") t) TO STDOUT", out);

    return new ArrayList<String>(Arrays.asList(
            out.toString("UTF-8").split("\n")));
}
```

REPLICAÇÃO LÓGICA
LIMITAÇÕES

Esquemas e operações DDL não são replicadas

Sequências não são replicadas

TRUNCATE não é replicado

Large Objects (LO) não são replicados

Apenas tabelas regulares são replicadas (pg_class.relkind = ‘r’)

- Visões, Visões Materializadas, Tabelas Estrangeiras, Tabelas Pai em uma relação de Herança, entre outras não são replicadas

Statement triggers são disparadas apenas na sincronização inicial

Replicação bidirecional não é suportada

DÚVIDAS?

OBRIGADO!

gerdan@gmail.com
machado.davy@gmail.com

PGConf.Brasil 2018