Mozart EAI

Mozart EAI, é um aplicativo que realiza o papel de *Enterprise Application Integration*. Os principais mecanismos são de *Message Broker* e *Publisher/Subscriber* para entrega de mensagens.

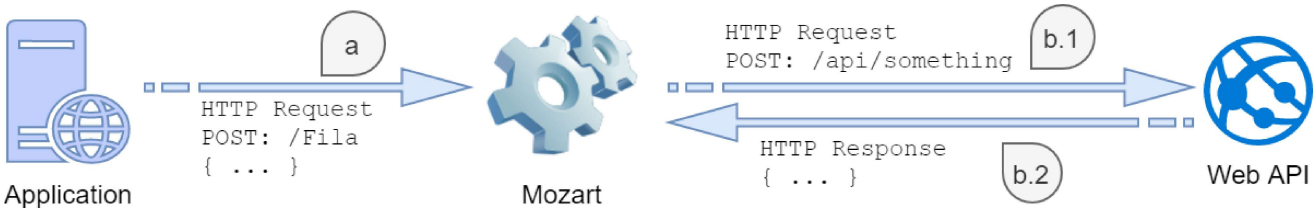
Aplicavel nas plataformas Windows, Linux e MacOS, utilizando bancos de dados MS SQL Server 2008+, Postgre SQL e SQLite (utilizado em cenários de PDV).

Message Broker

A maneira mais simplista do Mozart trabalhar é como *message broker*, onde aplicações enfileiram mensagem **-** diretamente pelo DB ou via protocolo HTTP **-** e, baseado em sua configuração, o Mozart entrega as mensagens para um destino final por protocolo HTTP. A mecanica se assemelha a com a de webhooks:

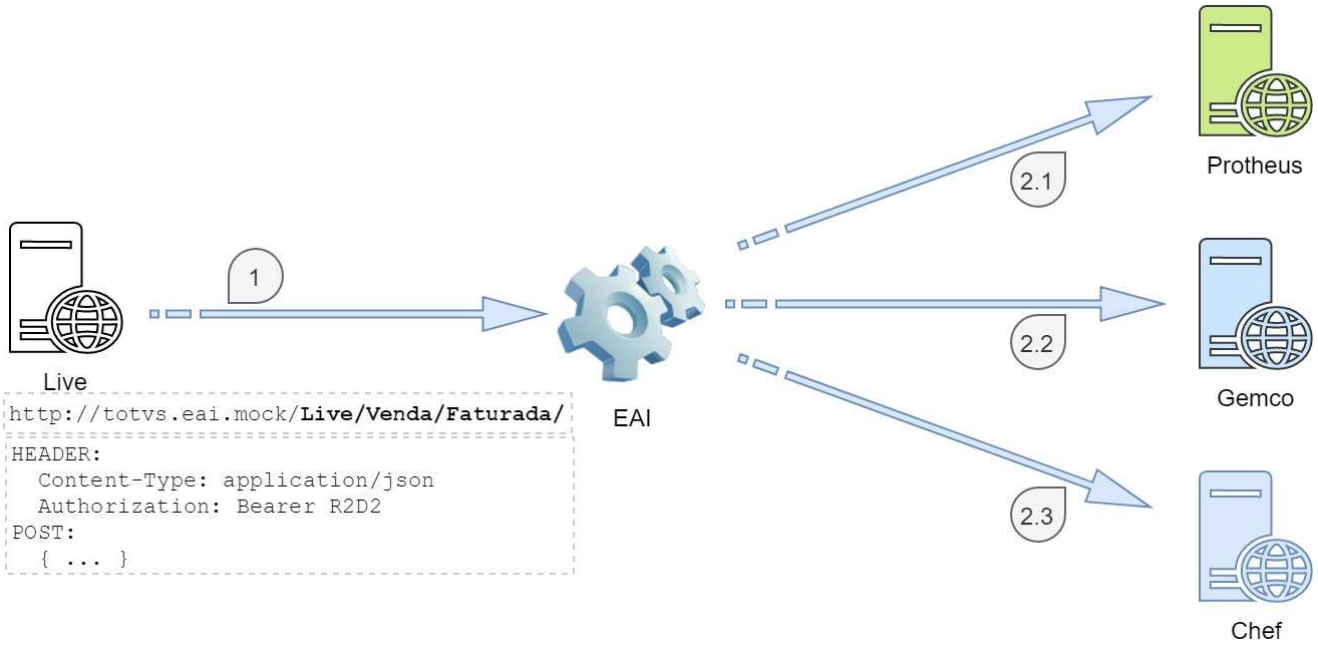
> Users can configure webhooks to cause events on one site to invoke behaviour on another. [...] Since they use HTTP, they can be integrated into web services without adding new infrastructure.

A imagem a seguir ilustra uma aplicação (*a*) enfileirando uma mensagem no Mozart que, por fim, entrega mensagem (*b.1*) para uma Web API configurada, retornando seu devido response (*b.2*).

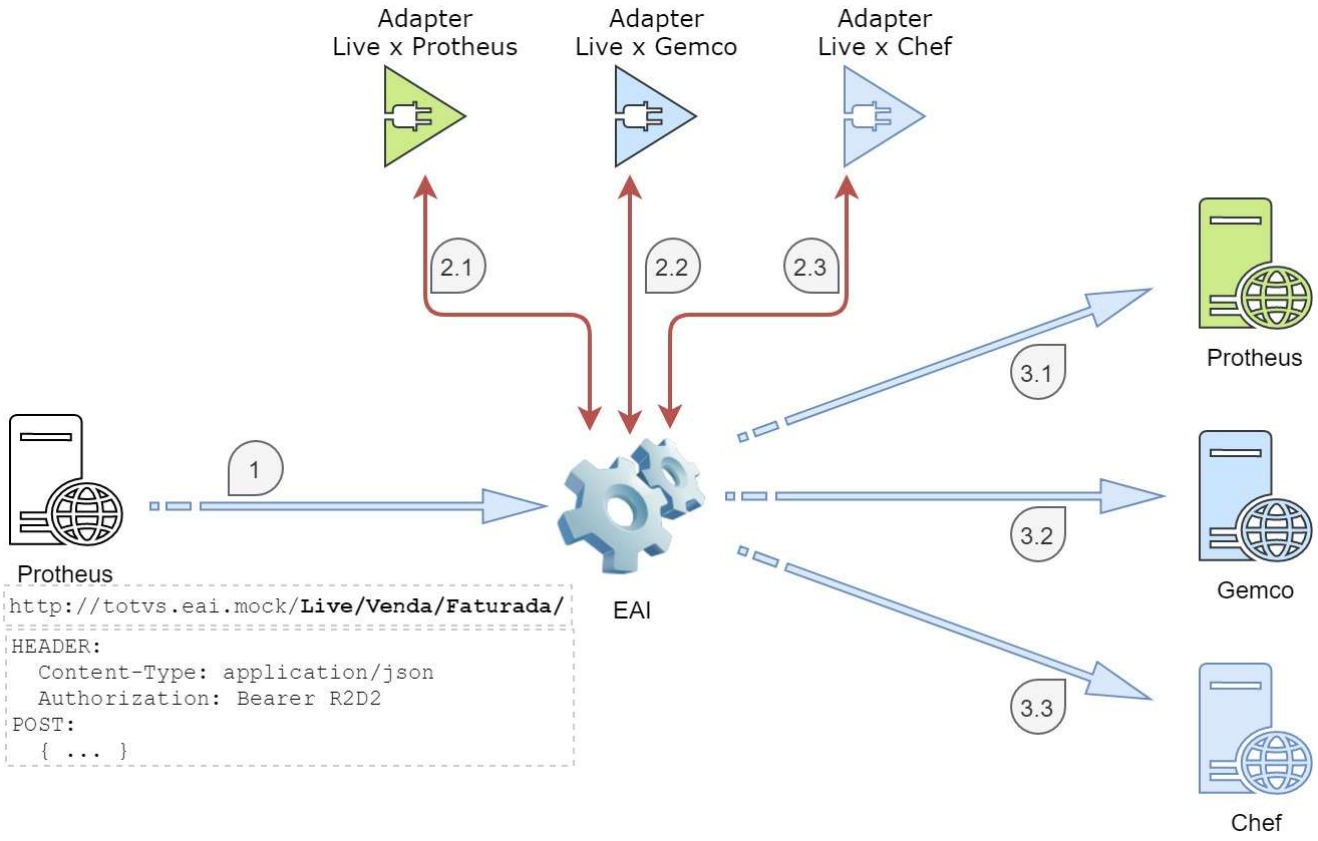


Publisher/Subscriber

Mozart trabalha como *Publisher/Subscriber*, onde um publicador publica/envia um evento de domínio **-** por exemplo, **Venda Faturada -** e assinantes recebem via protocolo HTTP a mensagem original.



Em uma quantidade relevante de cenários, existe a necessidade de - além de receber a mensagem original - converte-la para um formato de aplicação distinto. Para estes cenários aplicamos *endpoints* intermediários denominados *adapters* que trabalham como tradutores da mensagem original para o destino.



Configuração

As configurações do Mozart devem ser realizadas tanto em seu arquivo de configuração quanto no banco de dados, populando suas devidas tabelas.

Objeto de Request

O objeto de *request* é amplamente utilizado pelo Mozart para processamento das filas, contendo informações detalhadas para entrega de dados na *Web API* destino.

O objeto de `Request` contém as seguintes propriedades:

- `Url`: endereço do *endpoint* de destino;
- `Headers`: dicionário de chave/valor a serem enviados como headers;
- `Method`: *HTTP Method* a ser utilizado. Por ex: `POST`, `GET`, `DELETE`, `PATCH`, `PUT`, dentre outros;
- `TimeoutEmSegundos`: tempo para *timeout* a ser considerado para abortar *HTTP Request*;
- `CodigosExcecao`: códigos de exceção são utilizados para especificar que quando um HTTP Code específico for retornado o item da fila será colocado no status configurado; esta propriedade é uma lista de códigos de HTTP contendo seus devido status de destino
  - `Codes`: lista de códigos de HTTP;
  - `IdStatus`: para qual status devem ser direcionados;

Exemplo de configuração

Na configuração a seguir é realizado `POST` na `URL` especificada, com `Headers` `authorization` e `content-type`, obedecendo a regra que caso endpoint retorne `HTTP Code = 409`, item da fila será considerado `status = 3` (sucesso):

```
{
    "Url": "http://localhost/api/something",
    "Method": "POST",
    "Headers": [
        {"content-type": "application/json"},
        {"authorization": "bearer r2d2-t0k3n-c3po"}
    ],
    "TimeoutEmSegundos": 60,
    "CodigosExcecao": [{
        "Codes": [409],
        "IdStatus": 3
    }]
}
```

Message Broker

Quando Mozart é configurado como *message broker* objeto de *Request* é adquirido das tabelas ou configuração, obedecendo de forma prioritária (a primeira configuração que é encontrada é considerada), a seguinte ordem :

- Tabela
  - `MzFila`: campo *Request*;

    - `MzOperacaoCliente`: campo *Request*;
    - `MzOperacao`: campo *Request*;
- Configuração
    - `configuracao.json`: propriedade `Api` da `Operacao`;

Publisher/Subscriber

Quando Mozart é configurado como Publisher/Subscriber o a configuração do *Request* é obtido da tabela `MzFilaPasso`, pois para cada passo há um *Request* distinto.

Cliente

Para o Mozart, cliente é o que define a separação do principal *Tenant*, como por exemplo os clientes Totvs "Boticário", "Loreal", dentre outros. A configuração de cliente deve ser realizada no banco de dados, tabela *MzCliente*.

Operação

Operação é o que define e separa logicamente as filas de processamento que, por consequencia, também separa a visualização das filas nos *dashboards*. Como exemplo temos operações de "integração de venda", "cliente", "produto", dentre outros.

A configuração de Operação deve ser realizada no banco de dados, tabela *MzOperacao*, e também no arquivo de configuração ("*configuracao.json*").

A configuração em arquivo existe para que possam existir aplicações Mozart que facultam por processar apenas operações específicas. Por exemplo, pode existir uma instâcia de Mozart "*MZ1*" que processa operações "produto" e "cliente", e uma segunda instância de Mozart "*MZ2*", que usa mesma base de dados, mas que processa apenas operações de "Venda" e "Fechamento Caixa".

A configuração das operações de processamento no arquivo de configuração ocorre na propriedade "*Operacoes*", contendo as seguintes propriedades:

- `IdOperacao`: Operação a ser processada, de acordo com *Id* da tabela *MzOperacao*;
- `Api`: objeto `Request`; Apenas deve ser preenchido e utilizado caso processamento da fila seja para *Message Broker*;
- `Frequencia`: Frequência - em milissegundos - com que Mozart processará operação;
- `LimiteReprocessamento`: Limite de tentativas de reprocessamento; regra para reprocessamento obedece as definições gerais existentes na propriedade geral `Api.HttpCodesReprocessamento` e regras de exceção existentes nos objetos de `Request`.
- `Coletores`: coletores são responsáveis por trocar o *status* das filas (definido na propriedade `StatusDestino`) baseado em uma condição (definido na propriedade `StatusOrigem`) de acordo com o tempo de ociosidade (propriedade `TempoOciosoEmSegundos`) no status definido. Por ex., Caso um registro da fila esteja a mais de 60 segundos (`TempoOciosoEmSegundos`) no status *Processando* (`StatusOrigem = 2`), atualizar para status *Aguardando Processamento* (`StatusDestino = 1`). Por padrão, na configuração são aplicados coletores que regressam para o status "Aguardando Processamento" (`Status = 1`) todos que estiverem com status `5: "Reprocessar"` e `2: "Processando"`.

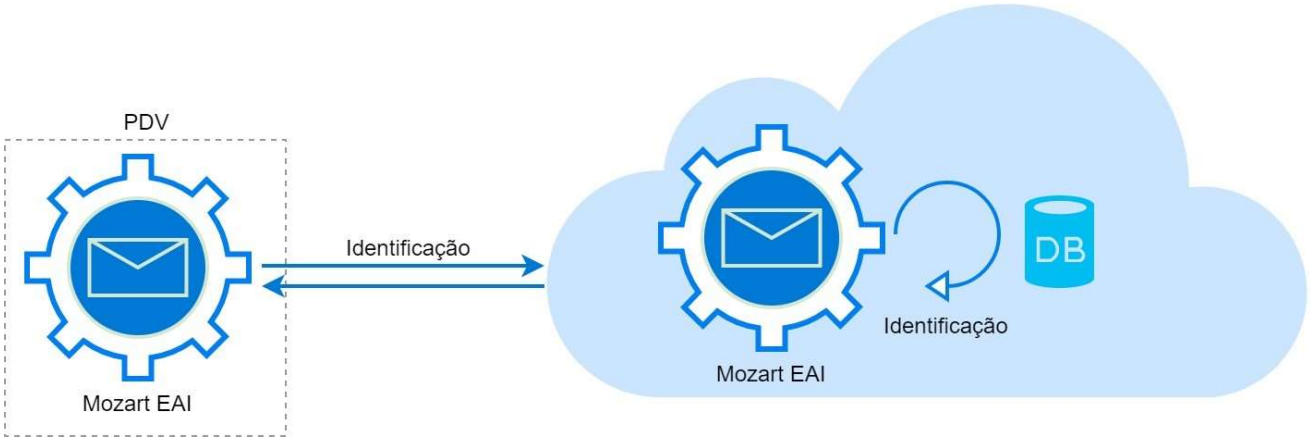
O exemplo a seguir demonstra uma operação devidamente configurada no Mozart:

```
{
  "Id": "Venda",
  "IdOperacao": 1,
  "Frequencia": 10000,
  "QuantidadeRegistrosEnfileirar": 5,
  "LimiteReprocessamento": 3,
  },
  "Coletores": [{
      "Id": "Reciclagem: Processando - Aguard. Proc.",
      "FrequenciaEmMilissegundos": 10000,
      "TempoOciosoEmSegundos": 120,
      "StatusOrigem": 2,
      "StatusDestino": 1
    },{
      "Id": "Reciclagem: Reprocessar - Aguard. Proc.",
      "FrequenciaEmMilissegundos": 10000,
      "TempoOciosoEmSegundos": 60,
      "StatusOrigem": 5,
      "StatusDestino": 1
    }]
}
```

Identificador

Sempre que o Mozart é iniciado este precisa estar devidamente identificado no Mozart centralizador. Portanto, uma rotina de **Identificação** é executada para o que o os casos a seguir sejam atendidos:

- Mozart Secundário: Se registra no Mozart Centralizador. Ex: PDVs e outras instâncias *on-premise*;
- Mozart Centralizador: Se registra localmente. Ex: Mozarts executando na nuvem.



Na configuração dos identificadores - realizada no arquivo *"configuracao.json"* - as seguintes propriedades devem ser preenchidas:

- **IdCliente**: Identifica para qual cliente a instância é atribuída; Deve ser preenchido com *Id* do cliente existente na tabela *MzCliente*; Para as instâncias de Mozart centralizadores é recomendado fixar o *IdCliente* como "Totvs"; Já para instâncias de Mozart Secundários (PDVs), utiliza-se o IdCliente relacionada a loja;
- **Referencia**: Esta propriedade é *schemaless*, permitindo qualquer conteúdo - desde que seja um *JSON* válido;
- **TempoEsperaRetryEmMilisegundos**: Caso o Mozart tente se identificar e falhe - por falta de conectividade *HTTP*, por exemplo - este tempo de espera define quanto esperar para próxima tentativa;
- **Id**: esta propriedade é preenchida **automaticamente** pelo Mozart assim que consegue se identicar;
- **UtilizaApi**: Se estiver com *false*, se identifica localmente (utilizando camadas de serviços locais); caso esteja como *true*, se identifica em uma *Web API* - definida na propriedade *API*; O valor padrão desta propriedade é *false*;
- **API**
  - **Url**: Url da API de Identificador do Mozart Centralizado;
  - **TimeoutEmSegundos**: Tempo de *timeout* para a chamada da *Web API*;

Centralizador

O exemplo a seguir demonstra a configuração no Mozart Centralizador (se identifica localmente):

```
"Identificador": {
  "IdCliente": 1,
  "Referencia": {
    "serial": "7ce9c2ee-3045-40b9-a592-672bff269dee"
  },
  "TempoEsperaRetryEmMilisegundos": 30000
}
```

Secundário

O exemplo a seguir demonstra a configuração do Mozart Secundário (se identifica na Web API):

```
"Identificador": {
  "IdCliente": 1,
  "Referencia": {
    "PDV-Key": "672bff269dee",
    "Auth": "R2D2"
  },
  "TempoEsperaRetryEmMilisegundos": 30000,
  "UtilizaApi": true,
  "Api": {
    "Url": "http://server/BemaHybrid.Mozart.Api/Identificador/",
    "TimeoutEmSegundos": 30
  },
}
```

Aplicação

Aplicação define os consumidores do Mozart, que podem ser Publicadores (*publishers*) ou Assinantes (*subscribers*) de mensagens do modelo *Publisher/Subscriber*.

Aplicações podem ser, por exemplo, "Totvs PDV Móvel", "Totvs Live", "Cia Shop", "Protheus", dentre outros.

Aplicação x Tenant

A associação entre Aplicação e Tenant define quais são os *tenants* de uma determinada aplicação, e a quais clientes estão vinculados. Esta associação não é obrigatória, e apenas é utilizada caso a assinatura não especifique o Cliente, que é obrigatório para inclusão na Fila.

Desta maneira, para a assinatura é analisado qual o Tenant da mensagem e, caso não exista cliente especificado na assinatura, cliente é pesquisado na associação de Aplicação x Tenant.

Por exemplo, temos com a seguinte associação de *Tenants* para a aplicação `PDV` :

- Cliente `Loren` com os *tenants* `loren-loja-c3p0` e `loren-loja-r2d2`;
- Cliente `Ipsum` com os *tenants* `ipsum-loja-kenobi` e `ipsum-loja-windu`.

Transação

As transações definem os eventos de negócio (idealmente usando *event driven design*) que serão publicadas pelas aplicações, como por exemplo, "CustomerVendor", "Cliente", "Produto", "Order", sendo escolhidas a critério de cada aplicação.

Para *event driven design* temos a definição por Martin Fowler:

> This happens when a system sends event messages to notify other systems of a change in its domain.

Assinatura

Nas assinaturas são definidas as aplicações (assinantes) que assinam os eventos de outras aplicações. Por exemplo:

- Aplicação "**Totvs Live**" **assina** publicação de mensagens de "**Cupom Fiscal Emitido**" provenientes da aplicação "**Totvs PDV Móvel**";
- Aplicação "**Protheus**" **assina** publicação de mensagens de "**Cupom Fiscal Emitido**" provenientes da aplicação "**Totvs PDV Móvel**";

Passos da Assinatura (pipeline)

Para cada assinatura criada podemos ter um ou mais passos que serão seguidos para entregar a mensagem com sucesso ao seu destino.

Os passos de uma assinatura se assemelham com *pipelines*, onde o *output* de cada passo é o *input* para o seguinte, conforme definição:

> consists of a chain of processing elements (processes, threads, coroutines, functions, etc.), arranged so that the output of each element is the input of the next; the name is by analogy to a physical pipeline.

Os passos da assinatura podem ser definidos como, por exemplo:

1. Entregar mensagem para "*Web API 1*";
2. Em seguida, com o retorno da "*Web API 1*", entregar mensagem para "*Web API 2*";

Setup

Uma base de dados deve ser criada (execute o arquivo de criação das estruturas contido no arquivo `"Scripts/[DB]/Create-Tables-FULL.sql"`).

A conexão com banco de dados deve ser configurada no arquivo `"appsettings.json"` . As configurações do aplicativo deve ser realizada no arquivo `"configuracao.json"` contendo dados de Identificador e Operações - conforme explicado nos devidos tópicos.

Message Broker

O uso do Mozart como *message broker* ocorre de maneira mais frequente nos casos de PDV (Mozart secundários, que são cenários que publicam mensagens diretamente para o Mozart Central (Cloud).

Para este cenários, apenas precisam ser configurados **Cliente**, **Operação** e a relação entre **Cliente x Operação**, conforme a seguir.

Criação de Cliente

Este passo de setup envolve a criação de Clientes denominados *Totvs*, *Lorem* e *Ipsum*.

```
SET IDENTITY_INSERT MzCliente ON;
INSERT INTO mzCliente (Id, Descricao) values
(1, 'Totvs'), (2, 'Lorem'), (3, 'Ipsum')
SET IDENTITY_INSERT MzCliente OFF;
```

Criação de Operação

Este passo de setup envolve a criação de Operações denominadas *Venda* e *Cliente*;

```
SET IDENTITY_INSERT MzOperacao ON;
INSERT INTO MzOperacao (Id, Descricao) values
(1, 'Venda'), (2, 'Cliente')
SET IDENTITY_INSERT MzOperacao OFF;
```

Associação de Cliente com Operação

Para que o *dashboard* e as operações internas do Mozart associem quais clientes utilizam determinadas operações é criado um vinculo entre Cliente e Operação.

```
INSERT INTO MzOperacaoCliente (IdOperacao, IdCliente)
SELECT MzOperacao.Id, MzCliente.Id FROM MzOperacao, MzCliente;
```

Enfileiramento diretamente no Database

Os enfileiramentos que ocorrem direto no DB geralmente ocorrem pela necessidade atômica do enfileiramento. Por exemplo, é criada uma transação distribuída que garante que no momento que a venda é criada/salva, na mesma transação será salva a fila para o Mozart.

Os campos que são obrigatórios para a fila são:

- `IdCliente`: Id indicando para qual Cliente fila existe;
- `DataInicio` e `DataFim`: fixados na data (com *timezone*) atual
- `IdOperacao`: Indica qual operação realizará o processamento da fila;
- `IdStatus`: fixado em `1` - "Aguardando Processamento";
- `Conteudo`: Conteúdo a ser enviado no body do *HTTP Request* que será realizado para entrega da mensagem. O conteúdo é *schemeless*, e pode ser um XML ou JSON.

O exemplo a seguir insere um registro na fila para `IdCliente = 1`, `IdOperacao = 1` com o conteúdo `{"id":1, "descricao": "lorem ipsum"}`:

```
INSERT INTO MzFila (IdCliente, DataInicio, DataFim, IdOperacao, IdStatus, Conteudo)
VALUES (1, SYSDATETIMEOFFSET(), SYSDATETIMEOFFSET(), 1, 1, '{"id":1, "descricao": "lorem
ipsum"}')
```

Publisher/Subscriber

O uso do Mozart como *publisher/subscriber* ocorre de maneira mais frequente nos casos de Mozart em Cloud (centralizador), e atua recebendo mensagens e redirecionando para assinantes.

Para este cenários, apenas precisam ser configurados - além de **Cliente**, **Operação** e a relação entre **Cliente x Operação - Aplicação**, **Transacao**, **Assinatura** e, por fim, **Assinatura Passo**.

Aplicação

A seguir temos criação das aplicações `PDV`, `ERP` e `ECommerce`.

```
-- Aplicação
SET IDENTITY_INSERT MzAplicacao ON;
insert into MzAplicacao (Id, Nome) values
(1, 'PDV'),
(2, 'ERP'),
(3, 'ECommerce');
SET IDENTITY_INSERT MzAplicacao OFF;
```

Aplicação x Tenant

A seguir temos a configuração da aplicação `PDV` com a seguinte associação:

- Cliente `Loren` com os *tenants* `loren-loja-c3p0` e `loren-loja-r2d2`;
- Cliente `Ipsum` com os *tenants* `ipsum-loja-kenobi` e `ipsum-loja-windu`.

```
SET IDENTITY_INSERT MzAplicacaoTenant ON;
insert into MzAplicacaoTenant (Id, IdAplicacao, Tenant, IdCliente) values
(1, 1, 'loren-loja-c3p0', 2),
(2, 1, 'loren-loja-r2d2', 2),
(3, 2, 'ipsum-loja-kenobi', 3),
(4, 2, 'ipsum-loja-windu', 3)
SET IDENTITY_INSERT MzAplicacaoTenant OFF;
```

Transação

A seguir temos a criação das Transações *Venda* e *CustomerVendor*.

```
insert into MzTransacao (Id, Nome) values
(1, 'Venda'), (2, 'CustomerVendor')
```

Aplicação x Transação x Evento

A seguir temos a criação das associção das Transações `Venda` e `CustomerVendor` com as Aplicações, onde:

- `PDV` lança eventos de domínio de `Venda Criada` e `Venda Cancelada`;
- `ECommerce` lança eventos de domínio de `Venda Criada` e `Venda Cancelada`;
- `ERP` lança eventos de domínio de `CustomerVendor Upsert`.

```
SET IDENTITY_INSERT MzAplicacaoEvento ON;
insert into MzAplicacaoEvento (Id, IdAplicacao, IdTransacao, Evento, IdOperacao) values
(1, 1, 1, 'Criada', 1),
(2, 1, 1, 'Cancelada', 1),
(3, 2, 2, 'Upsert', 2),
(4, 3, 1, 'Criada', 1),
(5, 3, 1, 'Cancelada', 1);
SET IDENTITY_INSERT MzAplicacaoEvento OFF;
```

Assinatura

A seguir temos a criação das Assinaturas, onde:

- `ERP` assina eventos de domínio de `Venda Criada` e `Venda Cancelada` lançados pelo `ECommerce`;
  - Apenas para *tenant* `loren-ecom-r2d2`;
  - Associado ao cliente `Lorem`;
  - Respectivamente, assinaturas `id = 1` e `id = 2`;
- `ERP` assina eventos de domínio de `Venda Criada` e `Venda Cancelada` lançados pelo `PDV`
  - Respectivamente, assinaturas `id = 3` e `id = 4`;
  - Associada qualquer *tenants* (campo *tenant* `null`);
  - Associação do cliente com o *tenant* será pela `MzAplicacaoTenant`;
- `ECommerce` assina eventos de domínio de `CustomerVendor Upsert` lançados pelo `ERP`
  - Assinatura `id = 5`;
  - Associada qualquer *tenants* (campo *tenant* `null`);
  - Associado ao `Cliente Totvs`;

```
SET IDENTITY_INSERT MzAssinatura ON;
insert into MzAssinatura (Id, IdAplicacaoEvento, IdAplicacaoAssinante, IdCliente, tenant) values
-- ECOM -> ERP: venda criada; tenant: loren-ecom-r2d2; cliente: 2 (loren)
(1, 4, 2, 2, 'loren-ecom-r2d2'),
-- ECOM -> ERP: venda criada; tenant: loren-ecom-r2d2; cliente: 2 (loren)
(2, 5, 2, 2, 'loren-ecom-r2d2'),
-- PDV -> ERP: venda criada; tenant: todos;
(3, 1, 2, null, null),
-- PDV -> ERP: venda cancelada; tenant: todos;
(4, 2, 2, null, null),
-- ERP -> ECOM: customervendor upsert; tenant: todos; cliente: 1 (totvs)
(5, 3, 3, 1, null);
SET IDENTITY_INSERT MzAssinatura OFF;
```

Assinatura Passo

A seguir temos a criação dos Passos das Assinaturas, onde:

- **Assinatura 1**: Assinante `ERP` recebe `Venda Criada` lançados pelo `ECommerce` para *tenant* `loren-ecom-r2d2`:
  - **Passo 1**: enviar mensagem para *Url* configurada;
- **Assinatura 2**: Assinante `ERP` recebe `Venda Cancelada` lançados pelo `ECommerce` para *tenant* `loren-ecom-r2d2`:
  - **Passo 1**: enviar mensagem para *Url* configurada;
- **Assinatura 3**: Assinante `ERP` recebe `Venda Criada` lançados pelo `PDV` para todos os *tenants*:
  - **Passo 1**: enviar mensagem para *adapter* (objeto de `Request` possui propriedade `Adapter = true`) na *Url* configurada; Utiliza propriedade `content` retornada pelo *Adapter* como *body* do passo seguinte.
  - **Passo 2**: enviar mensagem na *Url* configurada. Como *Url* é dinâmica - possui variáveis -, utiliza objeto `DynamicRequest` retornado pelo *Adapter* no **Passo 1**;
- **Assinatura 4**: Assinante `ERP` recebe `Venda Cancelada` lançados pelo `PDV` para todos os *tenants*:
  - **Passo 1**: enviar mensagem para *adapter* (objeto de `Request` possui propriedade `Adapter = true`) na *Url* configurada; Utiliza propriedade `content` retornada pelo *Adapter* como *body* do passo seguinte.
  - **Passo 2**: enviar mensagem na *Url* configurada. Como *Url* é dinâmica - possui variáveis -, utiliza objeto `DynamicRequest` retornado pelo *Adapter* no **Passo 1**;
- **Assinatura 5**: Assinante `ECommerce` recebe `CustomerVendor Upsert` (**Mensagem Padronizada**) lançado pelo `ERP` para todos os *tenants*:
  - **Passo 1**: enviar mensagem para *Url* configurada; Como o passo está configurado como `AwaitCallback = true`, não segue para **Passo 2** até que *callback* seja recebido;
  - **Passo 2**: enviar mensagem na *Url* configurada. Como *Body* da mensagem é utilizado conteúdo recebido no *callback*;

```
SET IDENTITY_INSERT MzAssinaturaPasso ON;
insert into MzAssinaturapasso (Id, IdAssinatura, Ordem, Request) values
-- assinatura 1
(1, 1, 1, '{ "Url": "http://www.mocky.io/v2/5a4fcbc12f00005617790cb0", "Method":
"POST","Headers": { "content-type": "application/json", "authorization": "bearer r2d2kenobic3po"
} }'),
-- assinatura 2
(2, 2, 1, '{ "Url": "http://www.mocky.io/v2/5a5375c630000008251ebf69", "Method": "DELETE",
"Headers": { "content-type": "application/json", "authorization": "bearer r2d2kenobic3po" } }'),

-- assinatura 3
(3, 3, 1, '{ "Adapter": true, "Url": "http://www.mocky.io/v2/5a53767330000070261ebf6a",
"Method": "POST", "Headers": { "Content-Type": "application/json" } }'),
(4, 3, 2, '{ "Url": "http://www.mocky.io/v2/{mockio-id}", "Method": "POST", "Headers": {
"Content-Type": "application/json" } }'),

-- assinatura 4
(5, 4, 1, '{ "Adapter": true, "Url": "http://www.mocky.io/v2/5a5376d130000076261ebf6d",
"Method": "POST", "Headers": { "Content-Type": "application/json" } }'),
(6, 4, 2, '{ "Url": "http://www.mocky.io/v2/{mockio-id}", "Method": "DELETE", "Headers": {
"Content-Type": "application/json" } }'),
```

```
-- assinatura 5: msg padr.
(7, 5, 1, '{ "AwaitCallback": true, "Url": "http://demo8591049.mockable.io/mock-protheus-
receipt", "Method": "POST", "Headers": { "Content-Type": "text/xml", "SoapAction":
"http://www.totvs.com/RECEIVEMESSAGE" } }'),
(8, 5, 2, '{ "Url": "http://demo8591049.mockable.io/mock-practicolive-receipt", "Method":
"POST","Headers": { "Content-Type": "text/xml", "SoapAction":
"http://www.totvs.com/RECEIVEMESSAGE" } }');
SET IDENTITY_INSERT MzAssinaturaPasso OFF;
```

Enfileiramento por HTTP

Versões

- .Net 4.0
  - S.O.: Windows;
  - Executado como *Windows Service* ou *Stand-alone Console*;
  - Para arquiteturas legadas que são limitadas a *.Net Framework 4.0* - casos de *Windows XP* e *Windows Server 2003*, por exemplo;
- .Net Core
  - S.O.: MacOS, Windows, Linux
  - Executado como *Stand-alone Console* ou *Web App* (IIS/Kestrel);

Bancos de Dados

- MS SQL Server 2008+;
- Postgre SQL;
- SQLite.

Message Queue

- A ser implementado:
  - MQTT

Releases

Versão 4.2.0.0

Branch: $/Novos Projetos/Produtos/Mozart/Releases/4.2.0.0

Notas

Carga de dados

Quando configurado (na tabela *MzCarga*), Mozart realizará leitura da tabela MzCarga e invocará *endpoints* configurados. Com o retorno dos *endpoints* enfileirará em suas filas locais para invocações de *endpoints* de carga;

Gravação de Históricos

Quando configurado (no arquivo *configuracao.json*) históricos apenas serão gravados nas condições aplicadas: Propriedades:

- Historico
  - CondicaoGravarHistoricos
    - **WhenStatus**: *array* que contém IdStatus desejados para salvar histórico;
    - **Request**: quando *true* grava dados do *request* enviados;
    - **Response**: quando *true* grava dados do *response* recebidos;
    - **WhenCallback**: quando *true* grava dados em caso de **callback** (aplicável quando recebido retorno de um passo configurado como *AwaitCallback* = *true*;

Exemplo

No exemplo a seguir:

- *CondicaoGravarHistoricos[0]*: configurado para gravar histórico quando status = 4 (erro) com o *request* e *response*;
- *CondicaoGravarHistoricos[1]*: configurado para gravar histórico quando status = 3 (sucesso); não grava request e response;
- *CondicaoGravarHistoricos[2]*: configurado para gravar histórico quando callback com o *request* e *response*;

```
"Historico": {
    "CondicaoGravarHistoricos": [
      {
        "WhenStatus": [
          4
        ],
        "Request": true,
        "Response": true
      },
      {
        "WhenStatus": [
          3
```

```
        ]
      },
      {
        "WhenCallback": true,
        "Request": true,
        "Response": true
      }
    ]
  }
```

DB

Nova tabela:

- *MzCarga*: contém configuração para carga de dados;

Script

Script de migração *"4.2.0.0.sql"* criado nas respectivas pastas dos DBs suportados.

Versão 4.1.0.0

Branch: $/Novos Projetos/Produtos/Mozart/Releases/4.1.0.0

Notas

FIFO por entidade

Suporte FIFO na fila por entidade na operação foi criado. Com a opção FIFO ativada, o processamento das filas passa a ser de forma sequencial, de acordo com as mensagens enfileiradas, sendo processada a fila seguinte apenas quando a fila atual estiver no status "Processado". Essa opção é ativada na operação, dentro do arquivo configuracao.json. O enfileiramento das mensagens e a ordem processamento podem ser feitos das seguintes maneiras:

- Controle de FIFO por transação (*Default*)
    - As mensagens serão organizadas e processadas por ordem de chegada de acordo com a transação (Pedido, Produto);
    - Parâmetro TipoFifo = 2 e PK não informada;
- Controle de FIFO por identidade (*Identity*)
    - As mensagens serão organizadas e processadas por ordem de chegada de acordo com a PK da entidade enfileirada, por exemplo, um código identificador de um produto, e a transação (Pedido, Produto);
    - Parâmetro TipoFifo = 1 e PK preenchida;

Rota para enfileiramento por **Identity**:

```
http://localhost/publicacao/{idAplicacao}/{transacao}/{evento}/{idUnico}/{tenant}?{Fifo.Tipo}&
{Fifo.Pk}
```

Rota para enfileiramento por **Transação**:

```
http://localhost/publicacao/{idAplicacao}/{transacao}/{evento}/{idUnico}/{tenant}?{Fifo.Tipo}
```

Exemplo

Propriedade "Fifo" igual a "true" no objeto Operacoes dentro do arquivo *configuração.json*:

```
"Operacoes": [{
    // ...
    "IdOperacao": 1,
    "Fifo": true,
    // ...
}]
```

Enfileiramento por Identity:

```
http://localhost/publicacao/1/CUSTOMERVENDOR/UPSERT/43867ad5-6f2c-4339-815f-
8b7f53d78708/tenantX?Fifo.Tipo=1&Fifo.Pk=123
```

Enfileiramento por transação:

```
http://localhost/publicacao/1/CUSTOMERVENDOR/UPSERT/43867ad5-6f2c-4339-815f-
8b7f53d78708/tenantX?Fifo.Tipo=2
```

DB

Novas tabelas:

```
- *MzFilaFifo* (tabela de controle de fila que utiliza fifo);
- *MzTipoFifo*;
- *MzTransacao* (nova tabela que normaliza tabela *MzAplicacaoEvento*).
```

Script

Script de migração *"4.1.0.0.sql"* criado nas respectivas pastas dos DBs suportados. **Importante** ressaltar que script de migração normaliza tabela *MzAplicacaoEvento* migrando *Transacao* para tabela *MzTransacao* e cria relacionamento.

4.0.1.3

Notas

- Correção de **dashboard** para mostrar todos os clientes;

4.0.1.2

Notas

- Nova funcionalidade que força fim dos passos (não segue para passos seguintes) definida nos objetos de **Request**:
  - Nova propriedade no objeto **Request.CodigosExcecao** denominada **ImporFimDosPassos**;
- Ajuste no tratamento de *Http Headers* para os objetos **Request** e **RequestDinamico**.
- No *Request Dinâmico* há nova propriedade para forçar método Http:
  - Nova propriedade no objeto **RequestDinamico** denominada **Method**;
- **Application Insights** da plataforma Azure ativados;
- Ajuste de visualização da versão no *dashboard*.

Versão 4.0.0.0

Notas

- Nova tabela: *MzUsuario* (script de criação da tabela incluído no pacote);
- Descoberta
  - Armazena histórico da última execução;
  - Detalhes da última execução é mostrado no *Dashboard*.

3.1.0.1

Notas

- Ajuste de coletores que nos ambientes *Azure* não eram executados corretamente;

3.1.0.0

Notas

- Nova tabela: *MzAplicacaoTenant* (script de criação da tabela incluído no pacote);
- Assinante que não especificar *Tenant* recebe publicação de todos;
- Caso *MzAssinatura.IdCliente* = *NULL*, adquire da MzAplicacaoTenant baseado no *Tenant* e *IdAplicacao* recebidos na publicação;

3.0.0.2

Notas

Parametrização dos atributos *SourceApplication* e *ProductName* da Mensagem Padronizada. A parametrização ser realizada no arquivo *appsettings.json*, conforme exemplo abaixo:

```
"MensagemPadronizada": {
    "Retorno": {
      "SourceApplication": "TOTVSEAI",
      "ProductName": "TOTVSEAI"
    }
  }
```

3.0.0.1

Notas

- Nova coluna *IdUnicoOriginal* na tabela *MzFila* (script de migração incluído no pacote);
    - *IdUnicoOriginal* sempre é preenchido de acordo com *IdUnico* informado na publicação de uma transação/evento;

3.0.0.0

Notas

- Atomicidade aplicada na geração das filas para assinantes;
- Melhora de geração de histórico na *MzFilaHistorico*;