# Serviço de Pedidos de Crédito

Esse serviço é uma aplicação web que permite gerenciar e persistir solicitações de crédito dos clientes, também é responsável por armazenar os log/eventos de alterações no pedido. O serviço utiliza o framework Spring Boot para desenvolver uma API REST que expõe os recursos e as operações relacionados aos pedidos de crédito. O serviço também utiliza o banco de dados NoSQL DynamoDB da AWS para armazenar e manipular os dados dos pedidos de crédito modelado sobre os pricípos de uma SingleTableModeling. Além disso, o serviço utiliza o mecanismo de busca distribuído ElasticSearch para indexar e consultar os dados dos pedidos de crédito de forma eficiente e flexível.

## Rotas

GET /pedidos?codigoCliente={}&nomeProduto={}&valorMinimo={}&valorMaximo={}&dataInicio={}&dataFim={}&status={}&sortBy={}&pageSize={}&pageNumber={}

Essa consulta permite consultar os pedidos de crédito de acordo com os critérios especificados nos parâmetros da URL. Os parâmetros são opcionais e podem ser combinados de diferentes formas. Os parâmetros são: - codigoCliente: o código do cliente que fez o pedido de crédito. - nomeProduto: o nome do produto solicitado pelo cliente. - valorMinimo: o valor mínimo do pedido de crédito. - valorMaximo: o valor máximo do pedido de crédito. - dataInicio: a data de início do período de consulta dos pedidos de crédito. - dataFim: a data de fim do período de consulta dos pedidos de crédito. - status: o status do pedido de crédito, que pode ser NOVO, EM\_ANALISE, APROVADO, REPROVADO ou CANCELADO, etc. - sortBy: o campo pelo qual os pedidos de crédito serão ordenados, que pode ser codigoPedidoCredito, codigoCliente, nomeProduto, valorPedido, dataPedido ou statusPedido. - pageSize: o número de pedidos de crédito que serão retornados por página. - pageNumber: o número da página que será retornada.

Retorna uma lista de pedidos de crédito que atendem aos critérios da consulta, no formato JSON, e retorna o código de status 200 (OK) se a consulta for bem-sucedida, ou o código de status 400 (Bad Request) se algum parâmetro for inválido ou faltar.

GET /pedidos/{CodigoPedidoCredito}

Essa Consulta permite consultar um pedido de crédito específico, de acordo com o código do pedido de crédito informado na URL. O parâmetro é obrigatório e deve ser um UUID válido. a consulta retorna o código de status 200 (OK) se a consulta for bem-sucedida, ou o código de status 404 (Not Found) se o pedido de crédito não existir.  
Exemplo de retorno:

{  
 "pedido": {  
 "codigoCliente": "23345677906912",  
 "agencia": <agencia>,  
 "conta": <conta>,  
 "regiao": "01",  
 "dicom": <dicom>,  
 "statusPedido": "NOVO",  
 "segmentoBancario": "K",  
 "codigoCanalSolicitacao": "01",  
 "descricaoCanalSolicitacao": null,  
 "valorPedido": <valorPedido>,  
 "unidadeMonetaria": "BRL",  
 "nomeGrupo": "<Nome Empresa>",  
 "dataPedido": "2020-01-01T11:00:00.00000",  
 "prazo": 360,  
 "unidadePrazo": null,  
 "codigoIdentificacaoOrigem": null,  
 "parecerOrigemPedido": {  
 "parecer": "S",  
 "justificativa": "estou de acordo com aprovação desse credito",  
 "dataParecer": null,  
 "usuarioParecer": "999999999"  
 },  
 "codigoProduto": "1234",  
 "descricaoProduto": "descricao\_produto",  
 "qualifier": "#PEDIDO",  
 "codigoPedidoCredito": "ccc9c52a-34cf-49da-ab31-bacef8f99234"  
 } ,  
 "eventos": [  
 {  
 "codigoEvento": "620097f1-c4c4-4866-b7d0-6a7ecb54c39c",  
 "dataHoraEvento": "2023-11-05T17:01:47.961+00:00",  
 "origemEvento": "Sistema Solicitar Crédito",  
 "dadosEventoAlteracao": null,  
 "detalhes": null,  
 "tipoEvento": "NOVO",  
 "qualifier": "#EVENTO#2023-11-05T15:01:47.954-02:00",  
 "codigoPedidoCredito": "0ac8b6c7-a570-4720-84b5-11cb81b8f46f"  
 }  
 ]  
}

DELETE /pedidos/{CodigoPedidoCredito}

Essa rota permite deletar um pedido de crédito específico, de acordo com o código do pedido de crédito informado na URL. O parâmetro é obrigatório e deve ser um UUID válido. Essa rota não retorna nenhum conteúdo, apenas o código de status 204 (No Content) se o pedido de crédito for deletado com sucesso, ou o código de status 404 (Not Found) se o pedido de crédito não existir.

PUT /pedidos/{CodigoPedidoCredito}

Essa rota permite atualizar um pedido de crédito específico, de acordo com o código do pedido de crédito informado na URL. O parâmetro é obrigatório e deve ser um UUID válido.

POST /pedidos

Essa rota permite criar um novo pedido de crédito, com os dados informados no corpo da requisição. O corpo da requisição deve conter um objeto JSON com os seguintes campos obrigatórios: exemplo de requisição:

{  
 "codigo\_cliente": "99945677906913",  
 "valor\_pedido": 55000000.00,  
 "codigo\_produto":"1234",  
 "segmento":"K",  
 "produto":"1234",  
 "path":"ltc",  
 "nome\_grupo": "Grupo Carrefur",  
 "descricao\_produto":"descricao\_produto",  
 "data\_pedido": "2020-01-01T11:00:00.00000",  
 "prazo": 360,  
 "parecer\_origem\_pedido": {  
 "funcional\_colaborador": "999999999",  
 "texto\_parecer": "estou de acordo com aprovação desse credito",  
 "indicador\_parecer": "S"  
 }  
}

## Arquitetura Hexagonal

O serviço segue a arquitetura hexagonal, que é um padrão arquitetural que visa isolar a lógica de domínio das dependências externas. A lógica de domínio é especificada em um núcleo de negócio, que chamamos de parte interna, com o resto sendo partes externas. O acesso à lógica de domínio a partir do exterior é feito através de portas e adaptadores. Esses componentes são modulares e intercambiáveis, o que aumenta a consistência do processamento e facilita a automação de testes.

O serviço é dividido em três camadas: aplicação (externa), domínio (interna) e infraestrutura (externa):

* Através da camada de aplicação, o usuário ou qualquer outro programa interage com a aplicação. Essa área deve conter coisas como interfaces de usuário, controladores RESTful e bibliotecas de serialização JSON. Ela inclui tudo que expõe a entrada para a nossa aplicação e orquestra a execução da lógica de domínio.
* Na camada de domínio, mantemos o código que toca e implementa a lógica de negócio. Esse é o núcleo da nossa aplicação. Essa camada deve ser isolada tanto da parte de aplicação quanto da parte de infraestrutura. Além disso, ela deve conter interfaces que definem a API para se comunicar com partes externas, como o banco de dados, com as quais o domínio interage.
* Finalmente, a camada de infraestrutura é a parte que contém tudo que a aplicação precisa para funcionar, como configuração do banco de dados ou configuração do Spring. Ela também implementa as interfaces dependentes da infraestrutura da camada de domínio.

## Estrutura do Projeto

O projeto é composto por quatro módulos: application, domain, infrastructure e ms-launcher. Cada módulo tem sua própria estrutura de arquivos e classes, conforme descrito abaixo:

### application

Esse módulo contém as classes relacionadas à camada de aplicação, como controladores, serviços e DTOs. A estrutura de arquivos desse módulo é:

application  
├── src  
│ └── main  
│ └── java  
│ └── com  
│ └── example   
│ └── application  
│ ├── controller  
│ │ └── PedidoCreditoController.java  
│ ├── dto  
│ │ ├── PedidoCreditoDTO.java  
│ │ └── PedidoCreditoMapper.java  
│ └── service  
│ └── PedidoCreditoService.java  
├── build.gradle  
└── settings.gradle

As principais classes desse módulo são:

* PedidoCreditoController: define os endpoints da API REST usando as anotações do Spring Boot. Essa classe tem métodos para criar, obter, atualizar e deletar um pedido de crédito no DynamoDB. Essa classe também tem um método para fazer uma consulta avançada no ElasticSearch usando vários critérios de filtragem e ordenação.
* PedidoCreditoDTO: representa um pedido de crédito com os seus atributos em formato DTO (Data Transfer Object). Essa classe é usada para transferir os dados dos pedidos entre as camadas de aplicação e infraestrutura.
* PedidoCreditoMapper: faz o mapeamento entre as classes PedidoCredito e PedidoCreditoDTO, usando a biblioteca MapStruct.
* PedidoCreditoService: implementa a lógica de negócio relacionada aos pedidos de crédito, usando as interfaces da camada de domínio e as classes da camada de infraestrutura.

### domain

Esse módulo contém as classes relacionadas à camada de domínio, como entidades, repositórios e portas. A estrutura de arquivos desse módulo é:

├── src   
│ └── main   
│ └── java   
│ └── com   
│ └── example  
│ └── domain  
│ ├── entity  
│ │ ├── Pedido.java  
│ │ └── PedidoItem.java  
│ ├── port  
│ │ ├── in  
│ │ │ ├── CreatePedidoUseCase.java  
│ │ │ ├── DeletePedidoUseCase.java  
│ │ │ ├── GetPedidoUseCase.java  
│ │ │ ├── ListPedidosUseCase.java  
│ │ │ └── UpdatePedidoUseCase.java  
│ │ └── out  
│ │ ├── LoadPedidoPort.java  
│ │ ├── PersistPedidoPort.java  
│ │ └── SearchPedidosPort.java  
│ └── repository  
│ └── PedidoRepository.java  
├── build.gradle  
└── settings.gradle

As principais classes desse módulo são:

* Pedido: representa um pedido de crédito com os seus atributos e métodos. Essa classe é a raiz do agregado do pedido, que contém uma lista de itens do pedido.
* PedidoItem: representa um item do pedido com os seus atributos e métodos. Essa classe é uma entidade filha do agregado do pedido.
* CreatePedidoUseCase: define a interface para o caso de uso de criar um novo pedido de crédito. Essa interface é implementada pela camada de aplicação e usa a porta PersistPedidoPort para persistir o pedido na camada de infraestrutura.
* DeletePedidoUseCase: define a interface para o caso de uso de deletar um pedido de crédito pelo seu código. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar o pedido da camada de infraestrutura e a porta PersistPedidoPort para deletar o pedido na camada de infraestrutura.
* GetPedidoUseCase: define a interface para o caso de uso de obter um pedido de crédito pelo seu código. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar o pedido da camada de infraestrutura.
* ListPedidosUseCase: define a interface para o caso de uso de listar todos os pedidos de crédito. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar os pedidos da camada de infraestrutura.
* UpdatePedidoUseCase: define a interface para o caso de uso de atualizar um pedido de crédito pelo seu código. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar o pedido da camada de infraestrutura e a porta PersistPedidoPort para persistir o pedido na camada de infraestrutura.
* LoadPedidoPort: define a interface para carregar um ou mais pedidos da fonte de dados. Essa interface é definida pela camada de domínio e implementada pela camada de infraestrutura, usando o repositório do DynamoDB ou o cliente do ElasticSearch.
* PersistPedidoPort: define a interface para persistir ou deletar um pedido na fonte de dados. Essa interface é definida pela camada de domínio e implementada pela camada de infraestrutura, usando o repositório do DynamoDB ou o cliente do ElasticSearch.
* SearchPedidosPort: define a interface para fazer uma consulta avançada nos pedidos usando vários critérios de filtragem e ordenação. Essa interface é definida pela camada de domínio e implementada pela camada de infraestrutura, usando o cliente do ElasticSearch.
* PedidoRepository: define a interface para o repositório do pedido, que estende a interface CrudRepository do Spring Data. Essa interface é usada pela camada de infraestrutura para interagir com o banco de dados DynamoDB.

### infrastructure

Esse módulo contém as classes relacionadas à camada de infraestrutura, como configurações, implementações das portas e adaptadores. A estrutura de arquivos desse módulo é:

infrastructure  
├── src  
 └── main  
 ├── java  
 │ └── com  
 │ └── example  
 │ └── mslibrary  
 │ └── infrastructure  
 │ ├── adapter  
 │ │ ├── DynamoDBPedidoAdapter.java  
 │ │ └── ElasticSearchPedidoAdapter.java  
 │ ├── config  
 │ │ ├── DynamoDBConfig.java  
 │ │ └── ElasticSearchConfig.java  
 │ └── repository  
 │ └── DynamoDBPedidoRepository.java  
 └── resources  
 └── application.properties

As principais classes desse módulo são:

DynamoDBPedidoAdapter: implementa as interfaces LoadPedidoPort e PersistPedidoPort usando o repositório do DynamoDB. Essa classe faz o mapeamento entre as entidades Pedido e PedidoItem e as tabelas do DynamoDB.  
ElasticSearchPedidoAdapter: implementa as interfaces LoadPedidoPort, PersistPedidoPort e SearchPedidosPort usando o cliente do ElasticSearch. Essa classe faz o mapeamento entre as entidades Pedido e PedidoItem e os documentos do ElasticSearch.  
DynamoDBConfig: define a configuração do DynamoDB, como o endpoint, a região, as credenciais e as tabelas. Essa classe usa as anotações do Spring Boot para criar os beans necessários para o DynamoDB.  
ElasticSearchConfig: define a configuração do ElasticSearch, como o endpoint, a porta, o esquema e o índice. Essa classe usa as anotações do Spring Boot para criar os beans necessários para o ElasticSearch.  
DynamoDBPedidoRepository: implementa a interface PedidoRepository usando o repositório do Spring Data DynamoDB. Essa classe usa as anotações do Spring Boot para definir os métodos CRUD para o pedido.  
application.properties: contém as propriedades da aplicação, como o nome, a versão, a porta e os perfis.

projeto  
├─ README.md  
├─ pom.xml  
└── src  
 └── main  
 ├── java  
 │ └── com  
 │ └── example  
 │ └── mslibrary  
 │ │── application  
 │ │ ├── controller  
 │ │ │ └── PedidoCreditoController.java  
 │ │ ├── dto  
 │ │ │ ├── PedidoCreditoDTO.java  
 │ │ │ └── PedidoCreditoMapper.java  
 │ │ └── service  
 │ │ └── PedidoCreditoService.java  
 │ │── domain  
 │ │ ├── entity  
 │ │ │ ├── Pedido.java  
 │ │ │ └── PedidoItem.java  
 │ │ ├── port  
 │ │ │ ├── in  
 │ │ │ │ ├── CreatePedidoUseCase.java  
 │ │ │ │ ├── DeletePedidoUseCase.java  
 │ │ │ │ ├── GetPedidoUseCase.java  
 │ │ │ │ ├── ListPedidosUseCase.java  
 │ │ │ │ └── UpdatePedidoUseCase.java  
 │ │ │ └── out  
 │ │ │ ├── LoadPedidoPort.java  
 │ │ │ ├── PersistPedidoPort.java  
 │ │ │ └── SearchPedidosPort.java  
 │ │ └── repository  
 │ │ └── PedidoRepository.java  
 │ └──── infrastructure   
 │ ├── adapter   
 │ | ├── DynamoDBPedidoAdapter.java   
 │ | └── ElasticSearchPedidoAdapter.java   
 │ ├── config   
 │ | ├── DynamoDBConfig.java   
 │ | └── ElasticSearchConfig.java   
 │ └── repository   
 │ └── DynamoDBPedidoRepository.java   
 └── resources   
 └── application.properties

## Observability

##

Icon-Architecture/64/Arch\_Amazon-CloudWatch\_64

<desc>Created with Sketch.</desc> <defs> <linearGradient x1="0%" y1="100%" x2="100%" y2="0%" id="linearGradient-1"> <stop stop-color="#B0084D" offset="0%"></stop> <stop stop-color="#FF4F8B" offset="100%"></stop> </linearGradient> </defs> <g id="Icon-Architecture/64/Arch\_Amazon-CloudWatch\_64" stroke="none" stroke-width="1" fill="none" fill-rule="evenodd"> <g id="Icon-Architecture-BG/64/Management-Governance" fill="url(#linearGradient-1)"> <rect id="Rectangle" x="0" y="0" width="80" height="80"></rect> </g> <path d="M55.0592315,46.7773707 C55.0592315,42.8680281 51.8575588,39.6876305 47.9220646,39.6876305 C43.9865705,39.6876305 40.785903,42.8680281 40.785903,46.7773707 C40.785903,50.6867133 43.9865705,53.8671108 47.9220646,53.8671108 C51.8575588,53.8671108 55.0592315,50.6867133 55.0592315,46.7773707 M57.0697011,46.7773707 C57.0697011,51.7881194 52.9663327,55.8642207 47.9220646,55.8642207 C42.8788018,55.8642207 38.7754334,51.7881194 38.7754334,46.7773707 C38.7754334,41.7666219 42.8788018,37.6905206 47.9220646,37.6905206 C52.9663327,37.6905206 57.0697011,41.7666219 57.0697011,46.7773707 M65.5096522,60.4735504 L58.5011554,54.2026253 C57.9352082,54.9944794 57.2808004,55.7174332 56.5540156,56.3634982 L63.5524601,62.6334248 C64.1495696,63.1686502 65.0784065,63.1187225 65.6182176,62.5255808 C66.155013,61.9324392 66.1067617,61.010773 65.5096522,60.4735504 M47.9220646,57.6616197 C53.9645309,57.6616197 58.8801289,52.7786859 58.8801289,46.7773707 C58.8801289,40.7750569 53.9645309,35.8931217 47.9220646,35.8931217 C41.8806036,35.8931217 36.9650056,40.7750569 36.9650056,46.7773707 C36.9650056,52.7786859 41.8806036,57.6616197 47.9220646,57.6616197 M67.1119965,63.8626459 C66.4264264,64.6165549 65.47849,65 64.5285431,65 C63.7002296,65 62.8699057,64.708422 62.207456,64.1172774 L54.9305615,57.5987107 C52.9070239,58.8968321 50.505518,59.6587296 47.9220646,59.6587296 C40.7718297,59.6587296 34.9545361,53.8800921 34.9545361,46.7773707 C34.9545361,39.6746493 40.7718297,33.8960118 47.9220646,33.8960118 C55.0733048,33.8960118 60.8905985,39.6746493 60.8905985,46.7773707 C60.8905985,48.8154213 60.3990387,50.7366411 59.5465996,52.4511599 L66.8556616,58.9906963 C68.2750531,60.265851 68.3896499,62.4496907 67.1119965,63.8626459 M21.2803274,29.392529 C21.2803274,29.9117776 21.3124949,30.429029 21.3738143,30.9293051 C21.4089975,31.2138932 21.3205368,31.4984814 21.1295422,31.7131707 C20.9777518,31.8839236 20.7736891,31.9967603 20.550527,32.0347054 C18.0786547,32.6687878 14.0104695,34.5880104 14.0104695,40.3456782 C14.0104695,44.6933865 16.4240382,47.0929141 18.4495863,48.3411077 C19.1411878,48.7744806 19.9594489,49.0051468 20.8229456,49.0141338 L32.9450717,49.0251179 L32.9430613,51.0222278 L20.811888,51.0112437 C19.5664021,50.9982625 18.384246,50.6607509 17.3840374,50.0346569 C15.3765836,48.7974474 12,45.8896553 12,40.3456782 C12,33.66235 16.5999543,31.191925 19.3000149,30.319188 C19.2799102,30.0116331 19.2698579,29.702081 19.2698579,29.392529 C19.2698579,23.9324305 22.9982737,18.2696254 27.9420183,16.2215892 C33.7241287,13.8150717 39.8500294,15.0083449 44.3263399,19.4109737 C45.7135638,20.7749998 46.8545053,22.4316024 47.7300648,24.3478294 C48.9061895,23.3802296 50.355738,22.8460027 51.8836949,22.8460027 C54.8863312,22.8460027 58.2659305,25.1097268 58.8680661,30.0605622 C61.6797078,30.7046302 67.6206453,32.9553731 67.6206453,40.422567 C67.6206453,43.4042521 66.6797455,45.8666886 64.8230769,47.7419748 L63.3896121,46.3410022 C64.8632863,44.8531553 65.6101757,42.8620367 65.6101757,40.422567 C65.6101757,33.891019 60.1055101,32.2663701 57.737177,31.8719409 C57.4677741,31.827006 57.2295334,31.6752256 57.0757325,31.4515493 C56.9259525,31.2358614 56.8686541,30.9712444 56.9138897,30.7146157 C56.5851779,26.6604826 54.1605516,24.8431126 51.8836949,24.8431126 C50.4472144,24.8431126 49.1001998,25.5381069 48.1874466,26.7503526 C47.9652897,27.0439277 47.6044105,27.193711 47.2344841,27.139789 C46.8695838,27.085867 46.5629872,26.8362283 46.4373329,26.4917268 C45.6140456,24.2260057 44.4278686,22.3207628 42.9119745,20.8309188 C39.0327735,17.0154404 33.7281496,15.9809374 28.7170543,18.0649216 C24.5463352,19.7924217 21.2803274,24.7672224 21.2803274,29.392529" id="Amazon-CloudWatch\_Icon\_64\_Squid" fill="#FFFFFF"></path> </g></svg> Logs

Para atender ao pilar de logs será utilizado o CloudWatch, para monitorar e observar o seu serviço de pedidos de crédito. O CloudWatch é um serviço da AWS que permite coletar visualizar logs, métricas e dados de eventos em tempo real em painéis automatizados para otimizar sua infraestrutura e manutenção de aplicações Você pode usar o CloudWatch para:

Monitorar a performance da aplicação, criando alarmes e ações automatizadas baseados em limites pré-definidos ou modelos de machine learning. Executar a análise de causa-raiz, analisando métricas, logs, análises de logs e solicitações de usuários para acelerar a depuração e reduzir o tempo médio de resolução. Otimizar os recursos de forma proativa, automatizando o planejamento de recursos e reduzindo os custos. Testar os impactos do site, visualizando capturas de tela, logs e solicitações da Web a qualquer momento.

## X-Ray

AWS X-Ray é um serviço de rastreamento distribuído que permite visualizar e depurar as solicitações que atravessam o seu sistema. Com o X-Ray, você pode coletar dados sobre as latências, os erros, as falhas e os gargalos das suas chamadas de API, das suas consultas ao DynamoDB e do seu envio de dados ao ElasticSearch. Você pode usar o X-Ray para criar um mapa de serviço que mostra a arquitetura do seu sistema e o fluxo das solicitações. Você também pode usar o X-Ray para analisar as solicitações individuais e ver os detalhes de cada segmento e subsegmento do seu rastreamento. Para usar o X-Ray, você precisa instrumentar o seu código com o SDK do X-Ray, que está disponível para várias linguagens, incluindo Java, que é o que você usa no seu serviço. Você também precisa habilitar o X-Ray no seu ambiente de execução, que pode ser o AWS Lambda, o Elastic Beanstalk, o ECS, o EKS ou o EC2. Você pode consultar os dados do X-Ray na console da AWS, na API do X-Ray ou no AWS CLI.

## image/svg+xml Prometheus

Prometheus é um sistema de monitoramento e alerta de código aberto que coleta e armazena métricas de séries temporais de vários componentes do seu sistema. Com o Prometheus, você pode definir regras para gerar alertas quando as métricas ultrapassarem certos limites ou apresentarem anomalias. Você também pode usar o Prometheus para consultar e visualizar as métricas usando uma linguagem de consulta específica e uma interface web. Para usar o Prometheus, você precisa instalar e executar um servidor do Prometheus, que pode ser hospedado na AWS ou em outro lugar. Você também precisa configurar o Prometheus para descobrir e raspar os alvos que expõem as métricas, que podem ser os seus serviços, os seus bancos de dados, os seus contêineres, os seus nós ou outros recursos da AWS. Você pode usar o cliente do Prometheus, que está disponível para várias linguagens, incluindo Java, para expor as métricas do seu serviço. Você também pode usar o AWS Distro for Prometheus, que é um serviço gerenciado da AWS que oferece uma distribuição compatível e segura do Prometheus. Você pode integrar o Prometheus com outras ferramentas de observability, como o Grafana, que é um painel de visualização de dados, ou o AWS X-Ray, que é um serviço de rastreamento distribuído.

Algumas métricas que você pode considerar para o seu serviço são:

* Métricas de negócio: são métricas que refletem o valor que o seu serviço entrega para os seus clientes e para a sua organização. Por exemplo, você pode medir o número de pedidos de crédito criados, aprovados, reprovados ou cancelados por período, por cliente, por produto ou por canal de solicitação. Essas métricas podem ajudá-lo a entender a demanda, a satisfação, a rentabilidade e a efetividade do seu serviço.
* Métricas de utilização: são métricas que indicam o quanto o seu serviço está sendo usado e o quanto de capacidade ele ainda tem disponível. Por exemplo, você pode medir a taxa de ocupação da sua fila de pedidos, o número de solicitações de API por segundo, o tamanho médio dos arquivos de entrada e saída, ou o tempo médio de processamento dos pedidos. Essas métricas podem ajudá-lo a dimensionar, otimizar e balancear o seu serviço.
* Métricas de erro: são métricas que mostram o quanto o seu serviço está falhando ou apresentando problemas. Por exemplo, você pode medir a taxa de erros da sua API, o número de pedidos inválidos, o número de tentativas de reprocessamento, ou o número de falhas na conexão com o DynamoDB ou o ElasticSearch. Essas métricas podem ajudá-lo a identificar, diagnosticar e corrigir as causas dos erros.
* Métricas de saturação: são métricas que revelam o quanto o seu serviço está próximo do seu limite de capacidade ou de desempenho. Por exemplo, você pode medir a utilização da CPU, da memória, do disco ou da rede dos seus nós, contêineres ou serviços, ou o tempo de resposta do DynamoDB ou do ElasticSearch. Essas métricas podem ajudá-lo a prever, prevenir e mitigar situações de sobrecarga ou degradação do seu serviço.
* Métricas de latência: são métricas que medem o tempo que o seu serviço leva para responder ou completar uma operação. Por exemplo, você pode medir a latência da sua API, o tempo de extração, de processamento ou de armazenamento dos pedidos, ou o tempo de consulta ou de indexação dos dados. Essas métricas podem ajudá-lo a avaliar, melhorar e garantir a qualidade e a eficiência do seu serviço.