

Guia de integração

API da Appmax — integração da própria loja

Do primeiro par de credenciais aos webhooks: tudo o que você precisa para colocar a sua loja para transacionar pela API.

Sumário

1. Credenciais pelo painel
2. Ambientes e URLs
3. Autenticação e access token
4. Primeira integração: clientes, pedidos e pagamentos
5. Webhooks do painel
6. Erros, rate limit e boas práticas

A versão mais recente deste guia e de todas as páginas citadas está em docs.appmax.com.br.

1. Credenciais pelo painel

A tela **API** do painel gera as credenciais do merchant (`client_id` e `client_secret`) da sua própria loja, sem passar pelo fluxo de instalação de aplicativo.

Antes de começar

- Acesso ao painel da Appmax com perfil de **produtor**.
- O item **API** visível no menu lateral, logo abaixo de **Aplicativos**. Se não aparecer, fale com o suporte: o acesso é liberado por permissão.

Ambiente	Painel
Sandbox	<code>https://breakingcode.sandboxappmax.com.br</code>
Produção	<code>https://admin.appmax.com.br</code>

Passo a passo

1 Abrir a tela API

No menu lateral do painel, clique em **API**, logo abaixo de **Aplicativos**.

A tela é organizada em abas: **Credenciais**, **Webhooks**, **Documentação** e **Guia para IA**. O **Guia para IA** mostra como usar um assistente de código (Claude, ChatGPT, Cursor, Copilot...) na integração: o contexto para colar na conversa — que já orienta o assistente a usar os [webhooks do painel](#), e não os da Appstore —, o `.env` com o Client ID da loja, prompts prontos e a conexão com o MCP da documentação.

2 Criar ou selecionar a loja

Na aba **Credenciais**, clique em **Gerar credenciais**. O formulário abre numa janela sobre o painel, sem sair da tela. Escolha uma das duas opções:

- **Criar nova loja** — informe o nome e o domínio da loja. O nome também aparece na fatura do cartão de crédito dos seus clientes. Se a sua conta tiver mais de uma empresa, selecione qual delas será a dona da loja.
- **Usar loja existente** — selecione uma das lojas que você já tem na Appmax.

É a única decisão que você precisa tomar nesta tela. A criação da loja, os vínculos e a geração do par de credenciais acontecem em seguida, sem nenhum passo manual.

3

Copiar as credenciais do merchant

A loja passa a aparecer em **Credenciais do merchant**, com um card por loja. Ao lado do título fica o total de lojas ativas; com mais de uma loja, aparece também uma busca pelo nome. Ao lado da lista, o bloco **Como funciona** resume o caminho até a primeira chamada.

O card de cada loja traz as credenciais dela:

- **Client ID** — sempre visível.
- **Client Secret** — exibido **uma única vez**, no momento em que é gerado. Depois disso o campo fica mascarado. Copie e guarde em local seguro.
- **External ID** — identificador desta instalação na loja, usado nas tokenizações via JS.

Perder o secret exige gerar um novo

O `client_secret` não é guardado para consulta posterior nem enviado por e-mail. Se você perder, use **Gerar novo secret** no card da loja: o `client_id` continua o mesmo, mas **um secret novo é emitido e o anterior deixa de valer na hora**. Toda aplicação que usa o secret antigo para de funcionar até você substituí-lo.

4

Cadastrar a URL de webhook

As credenciais permitem **chamar** a API. Para **receber** os eventos da Appmax (pedido aprovado, cliente criado, reembolso e outros), cadastre uma URL de webhook.

Na mesma tela **API**, abra a aba **Webhooks** e preencha:

1. a **loja** que você acabou de configurar;
2. a **URL** do seu endpoint (HTTPS, público);
3. os **eventos** que você quer receber, agrupados em Pedidos, Pagamentos, Clientes e Assinaturas.

O webhook cadastrado aparece na lista logo abaixo, onde você pode **testar** o envio ou **excluir**.

Logs e regras avançadas

A aba cobre o cadastro simples. Histórico de entregas, reenvio de evento e regras condicionais continuam na tela completa de webhooks do painel.

O formato das entregas, a lista de eventos e as regras de retentativa estão em [Webhooks do painel](#).

5

Autenticar e transacionar

Com o par em mãos, troque-o por um `access_token`:

```
curl --location 'https://auth.appmax.com.br/oauth2/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'client_id=SEU_CLIENT_ID_MERCHANT' \
--data-urlencode 'client_secret=SEU_CLIENT_SECRET_MERCHANT'
```

bash

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXLTUwIiwia2lkIiA6...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "customer/create order/create payments/credit-card ..."
}
```

json

Use o `access_token` como `Bearer` nas chamadas transacionais:

```
curl --location 'https://api.appmax.com.br/v1/customers' \
--header 'Authorization: Bearer SEU_ACCESS_TOKEN' \
--header 'Content-Type: application/json' \
--data '{ "firstname": "Maria", "lastname": "Silva", "email": "maria@exemplo.com" }'
```

bash

O token vale 1 hora e não há refresh token: quando expirar, repita a chamada acima. Os escopos do token são os concedidos ao aplicativo — veja [Autenticação e autorização](#).

O mesmo exemplo em outras linguagens

A aba **Documentação** traz este fluxo pronto em cURL, Node.js, PHP e Python, com botão para copiar.

Gerenciar as credenciais de uma loja

No card de cada loja existem duas ações, e a diferença entre elas importa:

	Gerar novo secret	Revogar
<code>client_id</code>	continua o mesmo	deixa de existir
<code>client_secret</code> anterior	deixa de valer na hora	deixa de valer na hora
Integração ativa	volta a funcionar ao trocar o secret	só volta com credenciais novas
Quando usar	perdeu o secret, ou quer trocá-lo por rotina	o secret vazou, ou a integração daquela loja foi desligada

Revogar

Revogar apaga o client no provedor de identidade da Appmax. A partir daí, qualquer chamada em `/oauth2/token` com aquele par responde erro de autenticação.

A revogação não tem volta

Não existe "desrevogar": o client deixa de existir. Para a loja voltar a acessar a API é preciso **gerar credenciais novas**, e nesse caso o `client_id` também é novo — diferente do que acontece em **Gerar novo secret**, que o preserva. Toda aplicação daquela loja precisa ser atualizada com o par novo.

A loja revogada continua na lista, marcada como tal, oferecendo **Gerar novas credenciais**. Ao confirmar, a Appmax emite um par novo — `client_id` e `client_secret` — e o card volta a mostrar as credenciais. Como na geração, o `client_secret` aparece **uma única vez**: copie-o antes de sair da tela e atualize a sua integração com o par novo.

Revogar não desativa a loja

A revogação encosta somente nas credenciais de API. A loja continua ativa na Appmax, vendendo pelo checkout e recebendo pedidos normalmente — o que ela perde é o acesso programático.

2. Ambientes e URLs

A Appmax tem dois ambientes. Comece pelo **Sandbox** e só vá para **Produção** quando a integração estiver validada: as credenciais de um ambiente não funcionam no outro.

Ambiente	URL base da API	URL de autenticação
Sandbox	<code>https://api.sandboxappmax.com.br</code>	<code>https://auth.sandboxappmax.com.br/oauth2/token</code>
Produção	<code>https://api.appmax.com.br</code>	<code>https://auth.appmax.com.br/oauth2/token</code>

Guarde as duas URLs e as credenciais em variáveis de ambiente — por exemplo `APPMAX_API_URL`, `APPMAX_AUTH_URL`, `APPMAX_CLIENT_ID` e `APPMAX_CLIENT_SECRET` — e troque só os valores ao mudar de ambiente.

Para testar pagamentos com cartão no Sandbox, use os [cartões de teste](#).

Instabilidades no sandbox

O ambiente de sandbox pode apresentar lentidão ou indisponibilidade temporária. Se você receber timeout (504) ou erro 503, aguarde alguns minutos e tente novamente. Esses problemas **não afetam** o ambiente de produção. Se persistir por mais de 30 minutos, entre em contato com o suporte.

3. Autenticação e access token

A API usa OAuth 2.0 no fluxo **client credentials**. Troque o `client_id` e o `client_secret` da loja por um `access_token` na **URL de autenticação** — não na URL da API:

```
curl --location "$APPMAX_AUTH_URL" \
  --header 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode 'grant_type=client_credentials' \
  --data-urlencode "client_id=$APPMAX_CLIENT_ID" \
  --data-urlencode "client_secret=$APPMAX_CLIENT_SECRET"
```

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXZWQ0aXBzdUIiwia2lkIiA6...",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Envie o token no header `Authorization: Bearer <access_token>` em todas as chamadas à API. Ele vale **1 hora**: guarde-o e reutilize até expirar, em vez de pedir um novo a cada requisição — a autenticação também conta no limite de requisições.

Erro	Causa provável	Solução
<code>401</code> ao chamar a API	Token expirado (mais de 1 hora)	Gere um novo token com as mesmas credenciais
<code>401</code> ao gerar o token	Credenciais erradas, revogadas ou de outro ambiente	Confira o par na tela API e o ambiente da URL
<code>403</code> em <code>/oauth2/token</code>	Token pedido na URL da API	Use a URL de autenticação do ambiente

Por que não utilizamos refresh tokens?

A API adota um modelo de autenticação sem refresh tokens. Essa decisão baseia-se na arquitetura de comunicação server-to-server.

▼ Motivos detalhados

1. **Natureza server-to-server:** a comunicação ocorre diretamente entre servidores, em ambientes controlados e seguros. Isso reduz a necessidade de mecanismos adicionais para renovação de tokens.
2. **Segurança e simplicidade:** tokens de curta duração (1 hora) limitam a janela de uso. Em ambientes server-to-server, onde credenciais são armazenadas de forma segura, essa abordagem simplifica o gerenciamento.
3. **Redução de complexidade:** elimina o armazenamento seguro de refresh tokens, rotação de tokens e lógica de renovação.
4. **Conformidade com melhores práticas:** em integrações server-to-server, é comum utilizar tokens de acesso curtos com autenticação baseada em chaves.

4. Primeira integração: clientes, pedidos e pagamentos

O fluxo de uma venda pela API tem cinco passos depois da autenticação: coletar o IP do cliente, criar o cliente, criar o pedido, processar o pagamento e confirmá-lo.

2. Coletar o IP do cliente

Antes de criar o cliente, é obrigatório coletar o IP usando o [Appmax JS](#). Inclua o script na sua página de checkout:

```
<!DOCTYPE html>
<html>
<head>
  <title>Checkout</title>
</head>
<body>
  <form id="customer-form" data-appmax-customer>
    <input type="text" name="first_name" placeholder="Nome" required />
    <input type="text" name="last_name" placeholder="Sobrenome" required />
    <input type="email" name="email" placeholder="E-mail" required />
    <input type="text" name="phone" placeholder="Telefone" required />
    <button type="submit">Continuar</button>
  </form>

  <script src="https://scripts.appmax.com.br/appmax.min.js"></script>
  <script>
    let clienteIP = null;

    window.AppmaxScripts.init(
      function onSuccess(data) {
        clienteIP = data.ip;
        console.log('IP coletado:', clienteIP);
      },
      function onError(err) {
        console.error('Erro ao coletar IP:', err);
      }
    );

    document.getElementById('customer-form').addEventListener('submit', function(e)
    {
      e.preventDefault();
      if (!clienteIP) {
        alert('Aguarde a coleta do IP');
        return;
      }
      // Envie os dados do formulário + clienteIP para seu backend
      enviarParaBackend({
        first_name: this.first_name.value,
        last_name: this.last_name.value,
        email: this.email.value,
        phone: this.phone.value,
        ip: clienteIP
      });
    });
  </script>
```

```
</body>
</html>
```

WARNING

No sandbox, se o IP não estiver disponível, use `127.0.0.1` para testes. Em produção, o IP real é obrigatório.

3. Criar o cliente

No seu backend, envie os dados do cliente para a API. A combinação `first_name` + `last_name` + `email` + `phone` + `ip` é a chave única — se já existir, o cliente é atualizado.

```
curl --request POST \
  --url https://api.sandboxappmax.com.br/v1/customers \
  --header 'Authorization: Bearer SEU_TOKEN' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{
    "first_name": "Junior",
    "last_name": "Almeida",
    "email": "junior.almeida@email.com",
    "phone": "51983655100",
    "ip": "127.0.0.1",
    "document_number": "25226493029",
    "address": {
      "postcode": "91520270",
      "street": "Rua Francisco Carneiro da Rocha",
      "number": "582",
      "complement": "Casa",
      "district": "Moinhos de Ventos",
      "city": "Porto Alegre",
      "state": "RS"
    }
  }'
```

bash

json

```
{
  "data": {
    "customer": {
      "id": 2023
    }
  }
}
```

Salve `data.customer.id` — este é o `customer_id` que será usado no pedido.

4. Criar o pedido

Vincule o pedido ao cliente. Os valores são em **centavos** (R\$ 123,00 = `12300`).

bash

```
curl --request POST \
  --url https://api.sandboxappmax.com.br/v1/orders \
  --header 'Authorization: Bearer SEU_TOKEN' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{
"customer_id": 2023,
"products": [
  {
    "sku": "LIVRO-001",
    "name": "Livro de receitas",
    "quantity": 1,
    "unit_value": 12300,
    "type": "digital"
  }
],
"shipping_value": 0,
"discount_value": 0
}'
```

json

```
{
  "data": {
    "order": {
      "id": 3531,
      "status": "pendente"
    }
  }
}
```

Salve `data.order.id` — este é o **order_id** que será usado no pagamento. O status começa como `pendente`.

5. Processar o pagamento

Escolha um dos métodos abaixo. No sandbox, use o [cartão de teste](#) `4000000000000010` para simular sucesso.

Cartão de crédito

Pix

Boleto

Existem duas formas: via **token** (recomendado) ou via **Appmax JS**. Este exemplo usa tokenização:

Passo 1 — Tokenizar o cartão:

bash

```
curl --request POST \
  --url https://api.sandboxappmax.com.br/v1/payments/tokenize \
  --header 'Authorization: Bearer SEU_TOKEN' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{
  "payment_data": {
    "credit_card": {
      "number": "4000000000000010",
      "cvv": "123",
      "expiration_month": "12",
      "expiration_year": "28",
      "holder_name": "Junior Almeida"
    }
  }
}'
```

json

```
{
  "data": {
    "token": "422146c7523a46119d6073ea56193913"
  }
}
```

Passo 2 — Pagar com o token:

bash

```
curl --request POST \
  --url https://api.sandboxappmax.com.br/v1/payments/credit-card \
  --header 'Authorization: Bearer SEU_TOKEN' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{
"order_id": 3531,
"customer_id": 2023,
"payment_data": {
  "credit_card": {
    "token": "422146c7523a46119d6073ea56193913",
    "holder_document_number": "25226493029",
    "holder_name": "Junior Almeida",
    "installments": 1,
    "soft_descriptor": "MINHALOJA"
  }
}
}'
```

json

```
{
  "data": {
    "order": {
      "id": 3531,
      "status": "autorizado"
    },
    "payment": {
      "method": "creditcard",
      "installments": 1,
      "paid_at": "2025-03-15 14:30:00"
    },
    "upsell_hash": "4000114202503117156088040208561001715608804"
  }
}
```

O status `autorizado` indica que o pagamento foi aceito e está em análise antifraude. Aguarde o webhook `order_approved` para confirmação.

6. Confirmar o pagamento

A confirmação é assíncrona. O caminho recomendado é receber o **webhook do painel** do evento de pagamento (veja o capítulo 5). Também é possível consultar o pedido:

Via consulta (polling)

Se precisar verificar o status manualmente:

```
curl --request GET \  
  --url https://api.sandboxappmax.com.br/v1/orders/3531 \  
  --header 'Authorization: Bearer SEU_TOKEN' \  
  --header 'Accept: application/json'
```

bash

```
{  
  "data": {  
    "order": {  
      "id": 3531,  
      "status": "aprovado",  
      "total_paid": 12300,  
      "amounts": {  
        "sub_total": 12300,  
        "shipping_value": 0,  
        "discount": 0,  
        "installment_fee": 0  
      }  
    },  
    "customer": {  
      "id": 2023,  
      "name": "Junior Almeida",  
      "email": "junior.almeida@email.com"  
    },  
    "payment": {  
      "method": "creditcard",  
      "installments": 1,  
      "paid_at": "2025-03-15 14:30:00"  
    }  
  }  
}
```

json

Veja todos os [status possíveis](#).

7. Após o pagamento

Produtos físicos: cadastrar código de rastreio

Para liberar os saques do merchant, atualize o pedido com o código de rastreio:

```
curl --request POST \
  --url https://api.sandboxappmax.com.br/v1/orders/shipping-tracking-code \
  --header 'Authorization: Bearer SEU_TOKEN' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{
    "order_id": 3531,
    "shipping_tracking_code": "BR123456789XX"
  }'
```

bash

Estorno (se necessário)

Para solicitar um reembolso total:

```
curl --request POST \
  --url https://api.sandboxappmax.com.br/v1/orders/refund-request \
  --header 'Authorization: Bearer SEU_TOKEN' \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{
    "order_id": 3531,
    "type": "total",
    "value": 12300
  }'
```

bash

Testando cenários de erro

Use o cartão `4000000000000028` para simular falha no pagamento. A API retornará:

json

```
{
  "error": {
    "message": "Payment not authorized"
  }
}
```

Veja mais [cartões de teste](#) na documentação de cartão de crédito.

5. Webhooks do painel

Cadastre a URL do seu endpoint na tela **API**, aba **Webhooks**: escolha a loja, a URL (HTTPS) e os eventos que quer receber.

Como a entrega chega

Cada evento gera um **POST** com corpo JSON e três campos de primeiro nível:

```
{
  "environment": "production",
  "event": "OrderPaid",
  "data": { "id": 99042, "status": "aprovado", "...": "..." }
}
```

json

Campo	Descrição
<code>environment</code>	Ambiente que gerou o evento. Sandbox e Produção enviam <code>production</code> ; não use este campo para separar os ambientes — use URLs diferentes em cada um.
<code>event</code>	Nome do evento. É o identificador da tabela abaixo com a primeira letra maiúscula (<code>orderPaid</code> → <code>OrderPaid</code>).
<code>data</code>	Dados do recurso: pedido, cliente ou assinatura, conforme o evento.

A requisição chega com o header `User-Agent: apphook` .

Eventos

São os mesmos 28 eventos da aba **Webhooks**, agrupados como lá.

Pedidos

Evento (<code>event</code>)	Quando dispara	<code>data</code>
<code>OrderApproved</code>	Pedido aprovado	pedido
<code>OrderAuthorized</code>	Pedido autorizado	pedido
<code>OrderAuthorizedWithDelay</code>	Pedido autorizado, enviado 60 minutos depois	pedido
<code>OrderIntegrated</code>	Pedido integrado	pedido
<code>OrderPendingIntegration</code>	Pedido pendente de integração	pedido
<code>OrderPaid</code>	Pedido pago	pedido
<code>OrderRefund</code>	Pedido estornado	pedido
<code>OrderPartialRefund</code>	Pedido estornado parcialmente	pedido
<code>OrderUpSold</code>	Upsell pago	pedido
<code>OrderRefusedByRisk</code>	Pedido recusado pelo antifraude	pedido

Pagamentos

Evento (<code>event</code>)	Quando dispara	<code>data</code>
<code>OrderPixCreated</code>	Pix gerado	pedido
<code>OrderPaidByPix</code>	Pix pago	pedido
<code>OrderPixExpired</code>	Pix expirado	pedido
<code>OrderBilletCreated</code>	Boleto gerado	pedido
<code>OrderBilletOverdue</code>	Boleto vencido	pedido
<code>PaymentNotAuthorized</code>	Pagamento não autorizado	pedido, com <code>issuer_message</code>
<code>PaymentNotAuthorizedWithDelay</code>	Pagamento não autorizado, enviado 60 minutos depois	pedido, com <code>issuer_message</code>
<code>OrderChargeBackInTreatment</code>	Chargeback em tratamento	pedido
<code>OrderChargeBackGain</code>	Chargeback ganho	pedido

Clientes

Evento (<code>event</code>)	Quando dispara	<code>data</code>
<code>CustomerCreated</code>	Cliente criado	cliente
<code>CustomerContacted</code>	Cliente contatado	cliente
<code>CustomerInterested</code>	Cliente interessado, enviado 60 minutos depois	cliente

Assinaturas

Evento (<code>event</code>)	Quando dispara	<code>data</code>
<code>SubscriptionDelayedEvent</code>	Assinatura atrasada	cliente, com <code>subscription</code>
<code>SubscriptionCancellationEvent</code>	Assinatura cancelada	cliente, com <code>subscription</code>
<code>CreatedSubscription</code>	Assinatura criada	ciclo de vida da assinatura
<code>CanceledSubscription</code>	Assinatura cancelada	ciclo de vida da assinatura
<code>ChargeSuccess</code>	Cobrança da assinatura aprovada	ciclo de vida da assinatura
<code>ChargeFailed</code>	Falha na cobrança da assinatura	ciclo de vida da assinatura

O que vem em **data**

Pedido

Grupo	Campos
Identificação	<code>id</code> , <code>customer_id</code> , <code>status</code> , <code>origin</code> , <code>created_at</code> , <code>paid_at</code> , <code>refunded_at</code> , <code>integrated_at</code> , <code>upsell_order_id</code> , <code>invoice_id</code> , <code>payment_link_id</code>
Valores	<code>total_products</code> , <code>freight_value</code> , <code>freight_type</code> , <code>discount</code> , <code>interest</code> , <code>total</code> , <code>full_payment_amount</code> , <code>total_refunded</code> , <code>refund_amount</code> , <code>cashback</code> , <code>partner_total</code> , <code>partner_affiliate_total</code>
Pagamento	<code>payment_type</code> , <code>installments</code> , <code>card_brand</code> , <code>card_details</code> , <code>issuer_message</code>
Pix	<code>pix_creation_date</code> , <code>pix_expiration_date</code> , <code>pix_emv</code> , <code>pix_ref</code> , <code>pix_qrcode</code> , <code>pix_end_to_end_id</code> , <code>pix_payment_link</code>
Boleto	<code>billet_url</code> , <code>billet_digitable_line</code> , <code>billet_date_overdue</code> , <code>order_billet_payment_code</code>
Relacionados	<code>customer</code> , <code>bundles</code> , <code>products</code> , <code>visit</code> , <code>traffic_description</code>
Empresa e comissões	<code>company_name</code> , <code>company_cnpj</code> , <code>company_email</code> , <code>co_production_commission</code> , <code>affiliate_comission</code> , <code>seller_name</code>

Campos que não se aplicam ao pedido chegam como `null` — os de Pix num pedido de cartão, por exemplo. `issuer_message` só é preenchido nos eventos de pagamento não autorizado.

▼ Exemplo de **data** de pedido (resumido, dados fictícios)

json

```
{
  "id": 99042,
  "customer_id": 17,
  "status": "aprovado",
  "payment_type": "CreditCard",
  "card_brand": "Visa",
  "installments": 12,
  "total_products": 242.03,
  "freight_value": 39.99,
  "freight_type": "PAC",
  "discount": 0,
  "interest": 57.97,
  "total": 339.99,
  "paid_at": "2026-10-01 15:02:52",
  "created_at": "2026-10-01 15:02:36",
  "origin": "API",
  "customer": {
    "id": 17,
    "firstname": "Maria",
    "lastname": "Silva",
    "email": "maria@exemplo.com",
    "document_number": "000000000000"
  },
  "bundles": [
    {
      "id": 23,
      "name": "Livro de receitas",
      "products": [{ "sku": "9000010", "name": "Livro de receitas", "price": "121.02",
"quantity": 2 }]
    }
  ],
  "company_name": "Minha Loja",
  "issuer_message": null
}
```

Cliente

id, **firstname**, **lastname**, **fullname**, **email**, **telephone**, **document_number**, **hash**, endereço (**postcode**, **address_street**, **address_street_number**, **address_street_complement**, **address_street_district**, **address_city**, **address_state**, **uf**), **site_id**, **created_at**, **interested_bundle** (o produto de interesse, com link para o checkout) e **visited_url**.

Assinatura

- **SubscriptionDelayedEvent** e **SubscriptionCancellationEvent**: os campos de **cliente** mais um objeto **subscription** com **id**, **installments**, **total**, **created_at**, **cancelled_at** e o **bundle** assinado (com os produtos).
 - **CreatedSubscription**, **CanceledSubscription**, **ChargeSuccess** e **ChargeFailed**: o ciclo de vida da assinatura, com **order_id** e **subscription_id**. Conforme o evento, vêm também **interval** e **interval_count** (na criação), **uuid** (nas cobranças) e **status**.
-

Atrasos, novas tentativas e tempo de resposta

- **Envio imediato**, exceto nos três eventos marcados com 60 minutos na tabela, que só saem uma hora depois do fato.
 - **Responda com HTTP 2xx**. Qualquer outro status, ou falha de conexão, conta como erro.
 - **Até 4 tentativas no total**. Depois de um erro, a Appmax espera 10 segundos, depois 100 e depois 1.000 (cerca de 17 minutos) antes de cada nova tentativa. Esgotadas as tentativas, a entrega é descartada.
 - **Responda rápido**. Coloque o evento numa fila e devolva **200** antes de processar — uma resposta lenta pode estourar o tempo limite e virar uma nova tentativa.
-

Segurança

As entregas **não são assinadas**: não há header de assinatura para validar. Para não aceitar chamadas forjadas:

- Use uma URL **HTTPS** com um caminho difícil de adivinhar (por exemplo, com um token aleatório) e não a publique.
 - Antes de agir sobre um evento sensível — liberar um pedido pago, por exemplo —, **confirme pela API**: consulte o pedido com **GET /v1/orders/{order_id}** usando o **data.id** recebido.
 - Ignore eventos repetidos: guarde o **event** com o **data.id** já processados.
-

Testar o endpoint

Na lista de webhooks da aba **Webhooks**, o botão **Testar** envia para a URL o primeiro evento que o webhook assina, com **dados fictícios**. Use para validar que o endpoint responde **2xx** — não para conferir valores de um pedido real.

Histórico de entregas, reenvio e regras condicionais ficam na tela completa de webhooks do painel.

6. Erros, rate limit e boas práticas

Códigos de status HTTP

Código	Descrição
200	Requisição bem-sucedida
201	Recurso criado com sucesso
400	Erro na requisição (ex.: pedido já pago)
401	Token inválido ou expirado
404	Recurso não encontrado
422	Erro de validação dos dados
500	Erro interno do servidor

Tratamento de erros

Respostas de erro seguem o formato envelope com o campo `error` ou `errors`:

```
{
  "error": {
    "message": "Order not found"
  }
}
```

json

Em erros de validação (`422`), os detalhes de cada campo são retornados:

```
{
  "message": "The given data failed to pass validation.",
  "errors": {
    "message": {
      "campo": ["Mensagem de validação"]
    }
  }
}
```

json

TIP

Sempre verifique o código HTTP da resposta antes de processar o corpo. Para erros `401`, obtenha um novo token e repita a requisição.

Visão geral

A API da Appmax aplica limites de requisições em dois níveis para garantir estabilidade e disponibilidade para todos os integradores. Sua requisição precisa passar por ambos para ser processada.

Níveis de controle

Nível 1 — Limite por credencial

Aplicado por `client_id` do merchant. Controla a taxa e o volume total de requisições.

Métrica	Limite	Descrição
Burst	50 requisições	Máximo de requisições simultâneas (pico instantâneo)
Rate	5 requisições/segundo	Taxa sustentada de requisições
Quota mensal	100.000 requisições/mês	Total de requisições por mês (reseta no dia 1)

Nível 2 — Limite por rota

Aplicado por **email + IP de origem**. Controla o uso de rotas individuais.

Tipo de rota	Limite	Janela
Rotas transacionais (padrão)	60 requisições	1 minuto
Operações sensíveis (login, credenciais)	5 requisições	1 minuto

TIP

Na prática, você pode fazer um burst de até 50 requisições instantâneas, e depois manter uma taxa sustentada de 5 requisições por segundo sem ser limitado. Se o limite for excedido, a requisição é rejeitada com `429`.

Quota mensal

Além do rate limit por segundo, há um limite mensal de **100.000 requisições** por `client_id`:

- O contador é incrementado a cada requisição
- Reseta automaticamente no primeiro dia de cada mês

WARNING

Quando a quota mensal é excedida, todas as requisições são bloqueadas com **429** até o próximo mês.

INFO

Os limites existem para garantir a segurança e estabilidade da plataforma, mas são totalmente flexíveis. Se a sua integração precisa de limites maiores, entre em contato com o nosso time — ajustamos conforme a sua necessidade.

Resposta de rate limit

Quando o limite é excedido, a API retorna status **HTTP 429** com headers informativos:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 0
Retry-After: 45
```

http

```
{
  "message": "Too many requests",
  "retryAfter": "45(s)"
}
```

json

Header	Descrição
<code>X-RateLimit-Limit</code>	Número máximo de requisições permitidas na janela
<code>X-RateLimit-Remaining</code>	Requisições restantes na janela atual
<code>Retry-After</code>	Segundos até poder tentar novamente

Boas práticas

1. **Implemente retry com backoff exponencial.** Ao receber `429`, aguarde o tempo indicado no header `Retry-After`. Se não houver header, use backoff exponencial (1s, 2s, 4s, 8s...).
2. **Use filas para operações em lote.** Se precisa criar muitos pedidos ou clientes, enfileire as requisições e processe respeitando o rate de 5 req/s.
3. **Cache o token Bearer.** O token dura 1 hora. Reutilize-o em vez de gerar um novo a cada requisição — a autenticação também consome quota.
4. **Monitore os headers de rate limit.** Use `X-RateLimit-Remaining` para ajustar a velocidade antes de atingir o limite.
5. **Agrupe operações quando possível.** Prefira criar cliente + pedido em sequência rápida em vez de múltiplas chamadas distribuídas.

Exemplo de retry com backoff

Go

Node.js

Python

PHP

```
func requestWithRetry(client *http.Client, req *http.Request) (*http.Response, error) {
    maxRetries := 3

    for attempt := 0; attempt <= maxRetries; attempt++ {
        resp, err := client.Do(req)
        if err != nil {
            return nil, err
        }

        if resp.StatusCode != http.StatusTooManyRequests {
            return resp, nil
        }
        resp.Body.Close()

        retryAfter := resp.Header.Get("Retry-After")
        wait, _ := strconv.Atoi(retryAfter)
        if wait == 0 {
            wait = 1 << attempt // backoff: 1s, 2s, 4s
        }

        log.Printf("Rate limited, retrying in %ds (attempt %d/%d)", wait, attempt+1,
maxRetries)
        time.Sleep(time.Duration(wait) * time.Second)
    }

    return nil, fmt.Errorf("rate limit exceeded after %d retries", maxRetries)
}
```