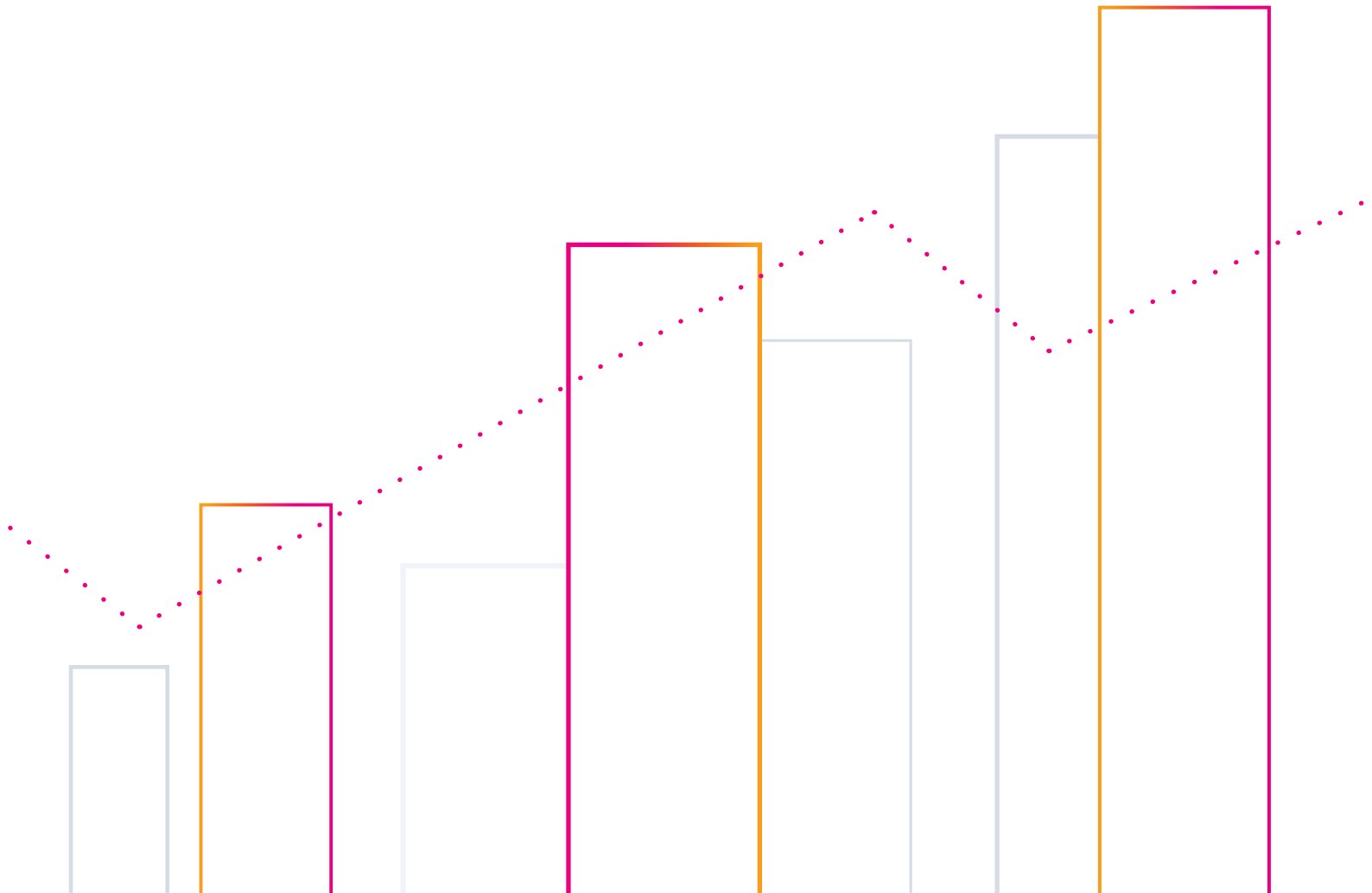
splunk> turn data into doing

ホワイトペーパー

APIの活用

パフォーマンス向上のための API監視ガイド



目次

はじめに 3
APIとは何か 3
APIの活用 4
第1章：SOAP、REST、JSON 5
第2章：APIを監視する理由 5
第3章：APIを監視する方法 7
第4章：SLAの監視 11
第5章：SplunkのAPIチェック 13

はじめに

マイクロサービスの普及、シングルページWebアプリの一般化、そして相変わらずのネイティブモバイルアプリの人気によって、APIは現代のWebを陰で支える存在となっています。

以前から、APIが企業の差別化や他のシステムとの統合に役立つ手段であることは知られていましたが、それが完全に主流となったのはソーシャルメディアが普及してからのことであると言われています。ソーシャルメディアネットワークが発達して多くの一般消費者をユーザーとして獲得していく中、企業はAPIの構築と活用を開始し、ユーザーがソーシャルメディアでのやり取りの際にコンテンツを共有したり埋め込んだりすることができるようになりました。

今では、複数のソーシャルメディアプラットフォームがAPIによって接続され、InstagramからFacebookやTwitterに投稿するといったようなことが行われます。また、API経由でデータセットに接続することによって、自分の現在地にピンを立ててチェックインや配車サービスに利用することも可能です。現在、私たちが使用している多くのWebアプリやモバイルアプリはAPIを利用する前提で構築されています。それらのアプリは、APIが正常に動作し、APIから正確、迅速かつ安全に、信頼できるデータが返されるかどうかに大きく依存しています。

APIは最新のアプリケーションやインフラの推進力であるため、APIのパフォーマンスを監視することは極めて重要です。

このホワイトペーパーでは、APIの基本機能に加え、APIを監視する重要性とAPIパフォーマンスの監視システムを実装する際に導入を検討すべき機能について説明します。

APIとは何か

APIとは、Webベースのソフトウェアアプリケーションやサービスにアクセスするためのプログラミング命令と標準規格のセットです。APIでは、サービスとやり取りする方法についてのインストラクションが開発者に提供されており、開発者はAPIを利用して異なるシステム間でデータを結び付けることができます。また、APIでは、そのサービスで利用できる機能、その使用方法とアクセス方法、さらに入出力でどのような形式にアクセスするかについての情報が提供されます。

今日では、ビジネスやアプリケーション全般がオープンAPIを基盤として構築されており、これを利用してシステム間でのデータのやり取りを可能にしています。APIをレストランのウェイターと考えてみてください。レストランでは、料理を注文するためのメニューが用意されています。注文の際には、ディナーをどのように提供してもらいたいかについて具体的な指示を出すでしょう。ウェイターに注文を尋ねられたら、次のように言います。「フィレミニョンをミディアムレアでお願いします。付け合わせはホウレンソウのクリーム煮と大盛りのベイクドポテトで。チャイブは抜いてください」

ウェイターは注文を取り、キッチンに伝え、料理の用意ができたらそれを運んでテーブルに並べます。

このシンプルなシナリオでウェイターが果たした役割はAPIの役割と全く同じです。1つ違うのは、APIが運ぶものはおいしい料理ではなく、データであるということです。ウェイターと同様に、APIはインストラクションのセット(リクエスト)をソース(アプリケーションまたは開発者)から受け取り、そのリクエストをデータベースに伝え、データを取得するか何らかのアクションを行い、ソースにレスポンスを返します。APIはシステムのつながりを維持するためのメッセンジャーなのです。

APIメッセージの送受信では、APIが組織内部で使用されるプライベートなプログラマー APIなのか、パブリックなコンシューマー向けのAPIなのかを区別することが重要になります。プライベートAPIを使用すると、より俊敏で柔軟かつ強力なビジネスが実現します。

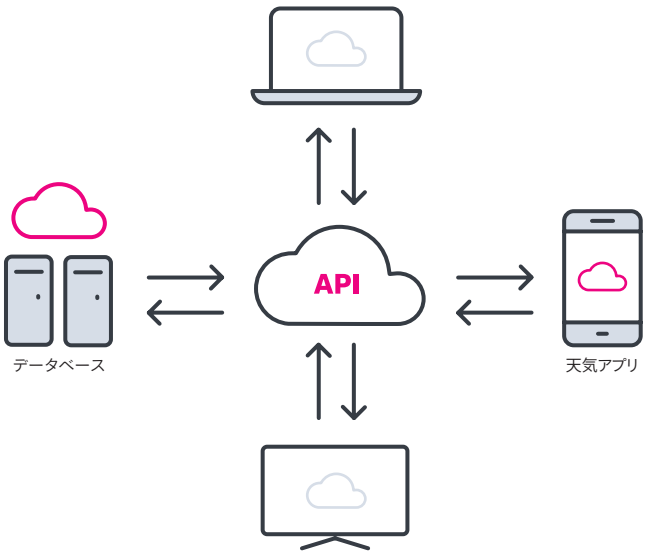
パブリックAPIは、それに接続することでビジネスにおける新たな統合機能を提供したり、新たなパートナーシップを構築したりすることに役立ちます。

APIの活用

パブリックAPIを利用している有名な会社の例を見てみましょう。

The Weather Company社は、世界中の何百万というエンドポイントやソースから気象データを収集し、このデータを蓄積して、コンシューマー向けアプリケーションが何百もの異なるAPIを経由してアクセスできるようにしています。

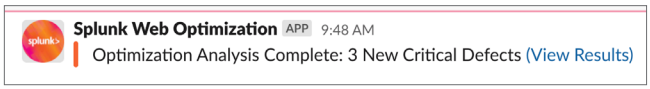
現在地に基づいた現在の気温と気象状況を常時表示するテレビやデスクトップのウィジェットを考えてみてください。デバイスには温度計や気圧計もなければ、気象パターンを検出したり予想したりするようなテクノロジーが内蔵されているわけでもありません。しかし、デバイスはユーザーの現在地の情報をAPIに送ることができ、その情報に基づいて正確な気温や天気予報を表示することができます。たとえThe Weather Company社のWebサイトやネイティブモバイルアプリを使用していなくても、シンプルなリクエストに応答するThe Weather Company社のAPIから送られるデータを利用することができるのです。



では次に、上記のパブリックAPIのシナリオと比較しながら、内部のビジネスプロセスを改善するタスクをトリガーするAPIの仕組みを例に考えてみましょう。

Splunk Web Optimizationはデジタルビジネスに関連する多くのメトリクスを収集して、サイトのパフォーマンス改善に役立つ提案を行います。この最適化ツールのおかげで、Webサイトの変更によってパフォーマンスの低下が発生した、つまりサイトやWebアプリに、ロード時間が想定よりも遅くなるような何らかのコンテンツが加えられた際に、チームがそれに気付くことができます。

最適化ツールのAPIを使用することで、Splunkチームはデプロイプロセスにパフォーマンステストを組み込んで、発生したあらゆるパフォーマンス低下を突き止めることができます。デプロイプロセスにはリクエストが組み込まれ、このリクエストを受けて最適化ツールのAPIがSplunkのステージング環境のパフォーマンススキャンをトリガーし、スキャン結果の詳細をSlackのチャットチャネルに送信します。こうすることで、既存のChatOpsの設定を利用して、ステージング環境でビルドした何かがアプリケーションの速度低下につながっている場合、チームに通知を送ることができるようになります。



このシナリオでは、APIは単にリクエストに基づいてデータを返しているだけではなく、リクエストに基づいてアクションを発生させています。The Weather Company社のAPIもSplunkのAPIも、パブリック、つまりコンシューマー向けにすることができ、それにより、他のシステムがこれらのAPIを利用してデータを返したり、何らかのアクションを発生させたりすることが可能になります。

第1章

SOAP、REST、JSON

基本的には、APIとはリモートサービスにデータをリクエストし、データを受信する仕組みです。この電子書籍では、リクエストとレスポンスがHTTPを使ってやり取りされる、WebベースのAPIを中心に扱います。リクエストとレスポンスの形式にはいろいろな種類があります。また、APIについて書かれた説明を読むと、さまざまな略語や用語を目にするでしょう。しかし、これらはAPIの大まかな概念や重要性、またそのパフォーマンスをテストしたり、監視したりする方法を知る上ではそれほど重要ではありません。

全体像を理解するための要点を以下に示します。

- APIの形式が異なれば、リクエストボディとレスポンスボディの構築方法も異なります。SOAPは厳密に構造化されており、XMLを使用します。RESTでは任意のものが使用できますが、主にJSONが使用されます。
- APIの形式が異なれば、リクエストを送る際に使用するHTTPメソッドも異なります。SOAPのようなフォーマットでは、ほとんどPOSTのみを使用しますが、RESTではGETやPOST、また頻度は少ないですがPUTやDELETEのようなメソッドも使用することがあります。
- 形式が異なると、資格情報や認証情報の受け渡しの方法も異なります。Cookieが使用されることもありますし、特殊なHTTPヘッダーやクエリーストリングパラメーターも使用されます。

説明がよく分からなかったり、混乱してしまったとしても、これだけは覚えておいてください。APIの役割はリクエストを送ること、そしてデータを受け取って返すことです。この基本となる考え方から、その他のすべてが派生しています。

第2章

APIを監視する理由

APIから返されるデータに依存している、またはサードパーティのAPIを前提にコア機能が構築されているようなシステムについて先ほど触れました。

複数のサービスを接続して、別々に管理されたシステム間でデータの受け渡しができるのがAPIの利点ですが、この接続性と相互依存性は脆弱性にもつながります。そのため、パフォーマンスと可用性については、自分が利用しているAPIと提供しているAPIの両方を監視することが重要になります。

APIパフォーマンスが重要な理由

APIで障害が発生し、Webサイトのパフォーマンスやユーザーエクスペリエンスが損なわれた場合、その障害の影響を受けるのはあなたの会社です。エンドユーザーや顧客は、別の会社に原因があるとは考えないでしょう。また、取引のプロセスにおけるそのAPIの重要度によっては、その障害が収益にすぐさま影響を与える可能性もあります。

たとえば、Webサイトの決済プロセスでの重要なコンポーネントがロケーションベースのサーチ機能であり、それをサードパーティ APIによって提供している場合、そのAPIが正常に動作しなければ、顧客は決済を完了させることができません。

あるいは、ソーシャルメディアプラットフォームから認証を行う必要があるアプリケーションを開発したとします。そのソーシャルメディアプラットフォームがダウンしてしまった場合、ユーザーはあなたのシステムにログインすることができなくなるでしょう。

開発者やサイトオーナーはそれでも、サードパーティのサービスを利用するメリットの方が、このような障害によるリスクに勝ると考えるかもしれません。リスクを正確に評価し、これらのサービスが受ける影響を経時的に可視化する上では、サイトのユーザーフローの中でも、APIに依存している部分を監視することが重要です。

また、パートナーや開発者にオープンなAPIを公開しているのであれば、APIが想定通りに動作し利用できるということを保証する責任があります。

あるいは、注文データをモバイルデバイスから製品倉庫のシステムに渡す新しい内部APIを開発したとします。仮に、2分以内に倉庫にデータが渡ることが重要で、そうしなければ生産スケジュール全体に狂いが生じるとしましょう。APIを開発してステージング環境でテストした際には、常に1分以内にデータが正常に渡されていました。ところが、APIの使用を開始して実際のリクエストを処理し始めると、実際の応答時間が徐々に2分のしきい値に近づいていることに気付きます。本番環境のAPIをプロアクティブに監視していなければ、本番前のテストの結果を根拠に、チームは開発したAPIで十分な速度が出ていると思い込んでしまうかもしれません。

注： 他のユーザーが利用できるようにAPIをホストしている場合は、本番前および本番環境の両方でそのAPIをプロアクティブに監視するようにしてください。

自前のAPIを監視する場合であれ、利用している外部サービスによる影響を確認する場合であれ、APIの可用性、機能、スピード、パフォーマンスを監視することが重要です。過去の動作から信頼性に疑問のあるAPIがあり、そのAPIに対しプロアクティブな監視を行っていないのであれば、すぐにでもチームで検討を開始し、そのAPIの監視を始める計画を立てましょう。

監視対象

ここまでで、APIのパフォーマンスを監視する重要性が理解できたので、以下に示す両方のAPIについて考慮する必要があることも理解できると思います。

- 自社のWebサイトやネイティブアプリが、重要なデータやプロセスのために利用しているAPI
- 顧客、エンドユーザー、または開発者がデータやプロセスのために利用している、自社で管理するAPI

これら両方のタイプのAPIを監視するにあたっては、以下の項目をテストすることが重要です。

- **可用性：**このAPIエンドポイントは稼働しているか。エラーが返されていないか。
- **応答時間：**APIのレスポンスの速度はどれぐらいか。時間の経過に伴って応答時間が遅くなっていないか。本番前環境よりも本番環境の方が応答時間が遅くないか。
- **データの検証：**APIは正確なデータを適切な形式で返しているか。
- **多段階プロセス：**このAPIが返す変数を正常に保存し再利用できているか。認証は想定通りに機能しているか。このAPIが返すデータでトランザクションを完了できているか。

これらは、APIのパフォーマンス監視についてチームで検討すべきごく基本的なコンセプトです。次のセクションでは、APIの監視を実施する上での技術的な側面について、また強力で柔軟なパフォーマンステストを構築するためにどのような機能が重要になるかについて、説明します。

第3章

APIを監視する方法

外部のプロアクティブな監視システムにアクセスすることができるのであれば、APIのレスポンスによって可用性を監視する作業は比較的シンプルであり、基本的なアップタイムチェックやPingによるチェックで簡単に実行できますプロトコルの定義またはエンドポイントの入力を行い、世界中のさまざまな場所から頻繁にテストを行うように設定し、レスポンスに問題や遅延がある場合にアラートを発生させる、といった流れになります。

しかし、可用性以外のものも監視する必要がある場合はどうすれば良いでしょうか。

APIトランザクションの完全なテストを実行するために、監視ソリューションが備えている必要がある重要な機能が数多くあります。これらは典型的なAPIリクエストの構成要素となるものであり、「どのエンドポイントにアクセスするべきか」という要素に加えて、それらの構成要素もテストの中に含めることが必要となります。

リクエストヘッダー

トランザクションのシミュレーションを効果的に実行するためには、サイトやアプリケーションの仕組みによっては、プロアクティブな監視テストの構成においてリクエストヘッダーが重要な役割を果たすことになります。たとえば、訪問者がCookieを受け取る際のチェックアウトプロセスがどのように動作するかをプロアクティブにテストする必要がある場合、そのトランザクションのテストを構成するには、リクエストヘッダーでCookieを設定する手段が必要になります。

リクエストヘッダーというのは、HTTPリクエストのヘッダー部に渡されるフィールドのことです。リクエストヘッダーには、HTTPトランザクションがどのように動作するかを定義するためのルールや設定を含めることができます。

サポートされるリクエストヘッダーの種類には標準的なセットがあり、名前と目的が決まっています。一般的なリクエストヘッダーには、たとえば以下のようなものがあります。

- **Authorization：**HTTPベーシック認証でのアクセス許可のために認証情報を送信します。
- **Cache-Control：**どのぐらいの時間リソースがキャッシュされて再利用できるかをブラウザーに伝えます。
- **Content-Type：**サーバー側でデータの解析方法が分かるように、リクエストボディのMIMEタイプをサーバーに伝えます。
- **Cookie：**ブラウザーに保存するCookieを設定して状態やセッションを追跡できるようにします。

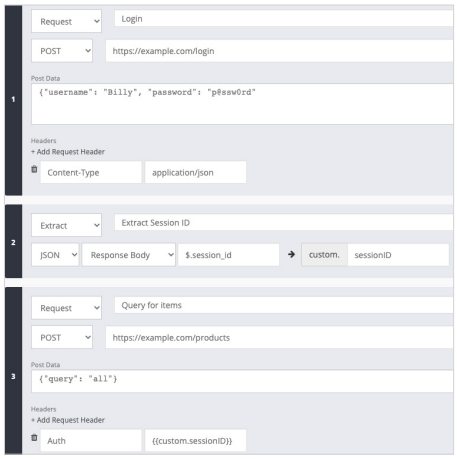
開発者によっては、カスタム名を持つカスタムリクエストを用意する場合もあります。一般的には、カスタムヘッダーの接頭辞としては、「X」がよく使用されます。たとえば、X-Http-Method-Overrideであれば、リクエストメソッドをPOSTなどからPUTやDELETEのような他のメソッドにオーバーライドするといった使い方です。

APIチェックでのリクエストヘッダー

SplunkのAPIチェックを使用すると、APIとのトランザクションでの可用性、応答時間、およびデータ品質を監視することができます。APIチェックでは、トランザクションの各リクエストパートでリクエストヘッダーを設定することができます。

資格情報としてユーザー名とパスワードをPOSTして何らかの情報にアクセスするというシナリオを考えてみます。そのエンドポイントにログインしたら、そのセッションでの他のコンポーネントに自動的に入力できるように、セッションIDを保存および設定する必要があります。

APIチェックでステップを構築する際には、[+ Add a Request Header (リクエストヘッダーの追加)]をクリックすると、各リクエストステップに1つ以上のヘッダーを追加することができます。



上の例では、SplunkのAPIチェックで以下のことを行っています。

1. ユーザー名とパスワードをPOSTでエンドポイントに送信してログインするリクエストを作成します。
2. JSONパスを使用してレスポンスボディからセッションIDを抽出し、この後のステップで再利用できるようにそのIDを変数として保存します。
3. リクエストヘッダーにこのセッションIDをセットし、別のエンドポイントにPOSTを実行するリクエストを作成します。

このトランザクションに、異なる機能を持つステップをさらに追加したり、Assertステップを追加してセッションIDが正しくセットされたか確認したりといったことも可能です。

認証処理

上の例では、リクエストヘッダーを使用して認証用のユーザー名とパスワードを送信しました。今度はAPIのセキュリティ面に注目し、パフォーマンステストを構成する際にAPIのセキュリティをどのように考慮すべきかを見てみましょう。

APIの認証では、保護されたデータやエンドポイントにアクセスする権限が誰にあるかということが規定されます。このことは、機密情報を共有するAPI、エンドユーザーに変更を許可するAPI、API経由でのデータアクセスによって請求金額を変更している企業などでは特に重要です。また、人間のエンドユーザーからAPIを保護する必要がある一方で、ますます増えている人間以外のエンティティでのシステム認証では、さらなる考慮事項があります。

APIのセキュリティが向上するにしたがい、プロアクティブな監視システムもそれに合わせて、セキュアなシステムに対して外部からのアクセスを可能とするように進化しています。

直接認証

HTTPベーシック認証は直接認証の良い例です。HTTPベーシック認証はHTTPの標準の一部であり、APIエンドポイントをはじめ、あらゆるHTTP URLで使用できます。ユーザー名とパスワードをともにBase64でエンコードし、リクエストに含めてAPIに送信するだけです。HTTPベーシック認証の利点は、実装が簡単で通常は標準的なフレームワークに含められている点です。一方欠点は、HTTPベーシック認証では高度なオプションが提供されず、簡単に復号されてしまう可能性がある点です。

直接認証のもう1つの例は、APIキーまたはトークンを使用する方法です。APIキーとは、たとえば「34d83d84f28d146aeae0e32f7803c88d」のような16進数による長い文字列で、APIエンドポイントへのアクセス認証のためにユーザー名とパスワードの代わりに送信することができます。本質的にはAPIキーも、ユーザー名とパスワードの組み合わせと変わりはありませんが、APIキーには便利な抽象化レイヤーがあります。たとえば、複数のエンドユーザーが単一のAPIキーを共有するといったことも可能です。

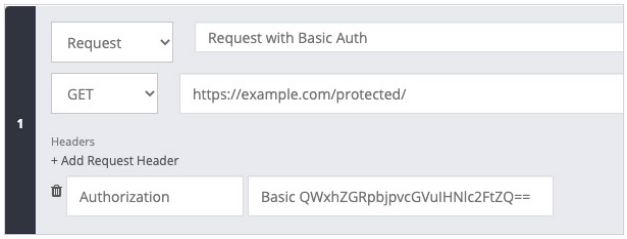
どのタイプの直接認証を使用する場合でも重要なのは、SSL/TLSを使用するか、APIエンドポイントのURLの先頭に「https://」を付けることです。SSL/TLSを使用すると、HTTPベーシック認証の資格情報やAPIキーがURL内に表示されないようにすることができます。

SSL/TLSの詳細とパフォーマンスの最適化方法について興味がある場合は、**Zoompfのブログにすぐれた記事が数多くあります**のでそちらをご覧ください。

HTTPベーシック認証

APIの保護にベーシック認証を使用している場合は、APIのパフォーマンスをチェックする外部監視を構成する際にその認証情報を含めるだけであり、極めてシンプルです。

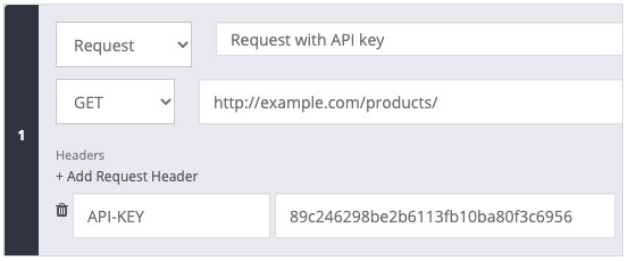
HTTPベーシック認証で監視リクエストを設定する最も一般的かつ信頼の置ける方法は、そのユーザー名とパスワードの値をBase64でエンコードし、その値をAuthorizationヘッダーに含めて送信することです。



ユーザー名とパスワードをBase64でエンコードする方法は手軽ではありますが、システムがリクエストを認証できるように復号することも非常に簡単であるという点に注意する必要があります。**オンラインのBase64エンコーダー /デコーダー**を使って自分で試してみてください。Base64は復号が簡単であるため、このタイプの直接認証はSSL/TLSを使用して保護することが重要です。

APIキー

監視の観点から言えば、URLやリクエストヘッダーにAPIキーを含めてエンドポイントにアクセスするプロセスを再現することは非常に簡単です。キーを入力し、キーに変更があった場合は監視テストの設定も忘れずに更新するようにします。



注：システムが異なると、APIキーを受け付ける方法も異なる場合があります(リクエストヘッダーではなく、POSTのデータの一部として受け付けるなど)。したがって、監視対象のAPIを確認し、APIキーを適切に送る方法を把握しておいてください。

チケットベース認証

直接認証の実装には確かに便利な点もありますが、APIにさらにセキュリティレイヤーを追加する必要がある場合があります。チケットベース認証では中央認証サーバーを使用します。中央認証サーバーは仲介役として機能し、資格情報をユーザーから受け取って、チケット、トークン、またはキーを送り返し、これによって特定のエンドユーザーが保護された特定のデータのみにアクセスできるようにしています。チケットベース認証が適しているのは、機密情報を保護している、APIでオブジェクトの生成や何らかの変更が可能である、またはAPIの利用によって料金が発生する場合があるといったシナリオです。

チケットベース認証の仕組み

チケットベース認証は、自動車のテスト走行のためのキーを受け取るような場合と似ていると考えられるかもしれません。

たとえば、私が自分の乗り潰した車をCraigslistで売りに出したとします。あなたは車のテスト走行をするために地元のダイナーで私と顔を合わせます。あなたが運転免許証を見せると、私が言います。「オンラインで知り合ったばかりだけど、あなたは見たところとても良い人そうだ。信用しますよ。キーをどうぞ。ちょっと走らせたら、数分後には戻ってきてくださいね」。これは、名前を確認しただけで車のキーをすぐに渡してしまうという点で、直接認証と似ていると言えるでしょう。

今度は、私が販売店であなたに高級車を売るとしましょう。あなたは販売店で私と対面し、運転免許証を私に手渡します。私は敷地内のすべての車を動かせる単独のキーを持ってはいません。私はあなたの運転免許証を受け取って、コンピューターシステムに情報を登録することで、あなたが信用に足る人物かどうかを確認します。これによってキーの箱が解錠され、テスト走行のために登録した自動車のみで使用できるキーをそこから取り出すことができます。これは、中央のシステムに依存する形でキーの提供を行い、それがチケットによってあなたに結び付けられるという点で、チケットベース認証システムに似ていると言えます。

OAuth、Kerberos、シングルサインオン、Webフォームはすべてチケットベース認証システムの例です。自分でカスタムの認証システムを開発することもできます。このタイプのプロトコルを実装するにはさまざまな方法がありますが、ほとんどのチケットベース認証システムは似た構造になっています。それは、まずチケットかトークンをリクエストし、保護されたデータやエンドポイントにそのチケットかトークンを使用してアクセスするというものです。

チケットベース認証システムにおけるチケットは、前の方で説明したAPIキーとよく似ているように見えます。1つ大きく違うのは、チケットは一時的なものであるという点です。有効なのはわずかな期間のみで、すぐ無効になるため、これが追加のセキュリティレイヤーとして機能しています。

チケットベース認証での監視

チケットベース認証を利用しているAPIを効果的に監視するためには、複数のステップを完了させ、チケットまたはトークンをその後のステップで再利用できるように変数として保存することが必要になります。

そのシンプルな例は、ユーザー名とパスワード、その他何種類かの項目をヘッダーに指定し、システムからトークンを取得してそれを変数として保存し、次にそのトークンをヘッダーとして使用してエンドポイントに別のリクエストを送信するといったものです。

APIに何らかの認証システムをまだ実装していないのであれば、すぐにでも実装を開始してデータとシステムを保護することが重要です。そして、セキュリティを向上させると同時に、外部監視システムにもシステムのパフォーマンスと信頼性を監視するための権限と能力を持たせる必要があります。アプリケーション側のAPIパフォーマンスばかりを監視していると、エンドユーザーがAPI経由でデータにアクセスしたり変更を行ったりする際に支障が出るような、接続に関する多くの問題を見逃してしまう可能性があります。

データの監視と検証

ブラウザーでWebサイトの監視を行う際には、レスポンスコードを確認するだけではなく、コンテンツや画像がページにロードされていることも確認する必要があります。ページから「200 OK」というコードが返されたがページが空白であるといった場合は、直ちに調査する必要があるはずです。この考え方はAPIエンドポイントを監視する場合にも当てはまります。APIエンドポイントを監視する際には、レスポンスコードが想定通りのものであることを確認するだけではなく、正しいデータが正しい形式で返されているかどうかも確認する必要があります。基本的なExtractおよびAssertオプションを使用して、APIがデータを正しい形式で返していることを検証する方法ついて、シンプルなユースケースを例に確認してみましょう。

データ検証の例

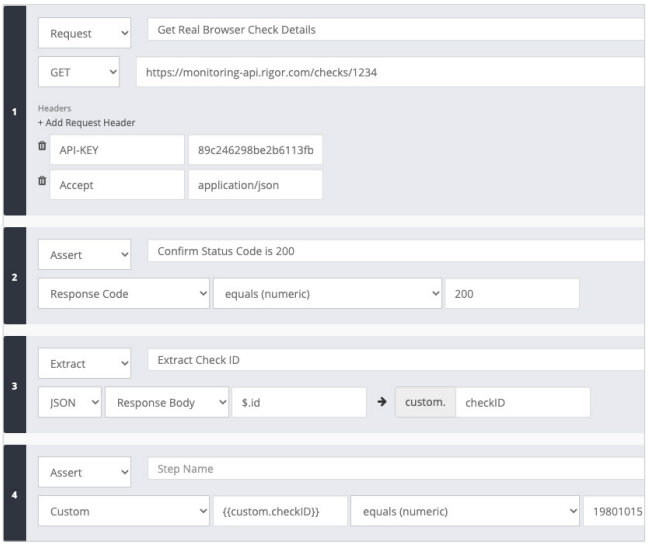
Splunk Optimizationは**オープンAPI**を備えており、Splunkユーザーはこれによって定期的にデータを取得してレポートを作成できます。SplunkユーザーがAPIキーを使ってチェックのためにエンドポイントにアクセスしたときに、「200 OK」のレスポンスコードと、エンドポイントに一致した正しいIDに対するデータセットが返されることが重要です。

チェック対象のエンドポイントに、設定した頻度で複数の場所からアクセスする外部/合成テストを作成して、以下を確認することができます。

1. レスポンスコードが「200」であるかどうか。
2. JSON出力に含まれるチェックIDが、アクセスしているURLエンドポイントに一致するかどうか。

以下の例では、Splunk APIチェックで次のことを行っています。

1. Splunk APIのエンドポイントに対し、APIキーを使用してリアルブラウザーチェックのデータをリクエストする。
2. レスポンスコードに値「200」が含まれているかどうか検証する。
3. JSONパスを使用してJSONデータからチェックIDを抽出する。



4. JSONパスによって抽出されたチェックIDが想定した値であるかどうか検証する。この非常にシンプルなユーザーフローによって以下のことをテストできます。

- **可用性：**APIが「200 OK」ではないレスポンスコードを返した場合、チェックは失敗します。
- **データ形式：**APIから返されたデータがJSON以外の形式であれば、JSONパスを使用して値を抽出するステップでチェックは失敗します。
- **データ品質：**IDの値の抽出に成功しても、それが想定した値に一致しない場合、Assertステップでチェックは失敗します。

たとえば、アラートを受信し、そのJSONデータから外部監視がチェックIDを抽出できなかった場合、受け取ったアラートをチェックして、APIエンドポイントからの出力やレスポンスボディを調査する必要があります。レスポンスボディを調べることで、形式が正しくないのか、またはIDの値が出力に含まれていないのかがすぐに分かり、この情報をもとに、トラブルシューティングを直ちに開始することができます。

これは、強力なAPI監視を実装する方法のほんの一例です。現在行っているAPIテストが、レスポンスコードと応答時間のみを監視するものであるならば、データの形式と品質をチェックする基準も追加することを検討すべき時機が来ているかもしれません。

失敗を想定したパフォーマンステスト

これまでのところ、リクエストヘッダー、認証、およびデータ検証を念頭に置いた監視の方法を見てきました。パフォーマンステストを書く際には、システムがAPIを呼び出した際にデータが返されないようなケースも許容できるテストを書くことが重要です。

ローカルの呼び出しを多用するコードを書いていると、外部APIへの呼び出しを行うラッパーが、アプリケーションのコンテキストではAPIから気付かれないということがよく発生します。データがなかった場合にアラートを出すようにテストを設計することで、重大なエラーを見逃さないようにすることができます。エラーメッセージや壊れたデータを受け取ったり、全く応答がなかった場合でも動作を続行できるように、コードには弾力性を持たせるよう心がけましょう。

第4章

SLAの監視

これより前の箇所で、Web経由でデータをやり取りする必要のある相互に依存したシステムで発生する脆弱性の問題に触れました。サードパーティのAPIを前提に構築されるアプリケーションの数が増えるにしたがい、企業にとってはパートナーのAPIが高速、安全、セキュアで、信頼性の高いものであることが極めて重要になります。

ビジネスの世界ではこのリスクを管理する方法が整備されてきており、その取り決めのことを通常は「SLA」と呼んでいます。

SLAの概要

SLA(サービスレベルアグリーメント)とは、ある当事者から別の当事者に対してどのようなサービスが提供されるかについての二者間の合意です。広義では、この合意にはカスタマーサポートの回答時間から製品提供にいたるまでのあらゆるサービスが含まれます。

2つのテクノロジープロバイダーまたはソフトウェアプロバイダー間でSLAが締結された場合、合意には以下の2項目に関する概要を定めるのが一般的です。

- **可用性：**パートナーによって保証されるアップタイムの割合はどれぐらいか。計画されたダウンタイムやメンテナンスについて、どの程度前もってパートナーに通知する必要があるか。
- **応答性：**パートナーのシステムからの応答速度はどの程度見込めるか。

そして、これらのSLAが満たされ履行されているかどうかによって、一方の当事者は、支払いの控除や返金を受ける、あるいは自由に契約を解消することができます。

たとえば、ライドシェア用のWebアプリがあり、そのアプリは地図サービスに特化したサードパーティのデータに大きく依存しているとしましょう。このライドシェアWebアプリが、あるすぐれた地図サービスプロバイダー 1社のみを使用するという合意を交わすとします。この地図サービスプロバイダーはたとえば次のように請け合います。「このライドシェアアプリからの当社の地図データへのアクセスを99.99%の可用性で保証します。また、メンテナンスが予定されている場合は3週間前にお知らせします」

ライドシェアWebアプリを管理しているチームは、次のように答えます。「満足のいく内容だと思います。ユーザーにとっても、それぐらいの障害頻度であれば許容範囲でしょうし、Twitterに不満を投稿されることもないでしょう。99.99%のアップタイムであれば十分です。3週間という期間も、ユーザーにダウンタイムを周知するのには十分な時間といえます。契約に合意させていただきます。ただし、御社のAPIの可用性が99.99%を下回った場合には、全額返金とさせてもらいます」

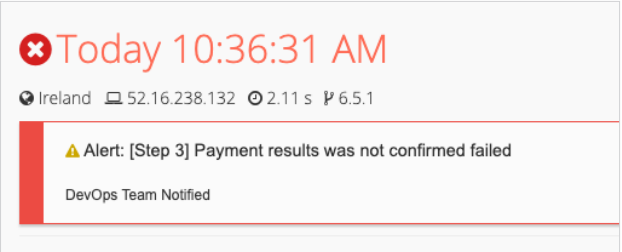
双方が契約書にサインをして握手を交わし、これでライドシェアWebアプリは地図サービスプロバイダーのAPIに接続してデータを引き出す機能を備えることになります。

SLAの遵守を確認する方法

地図サービスプロバイダー側のビジネスオーナーであれば、次のように考えるでしょう。「この契約の遵守を徹底するためには、問題の発生に先回りしてデータを取得する必要がある。それに、パートナーであるライドシェアWebアプリの会社にそのデータを公開できれば、うちのサービスが信頼できるということを知ってもらえるのでさらにいいのだが」

アプリケーションの内部監視は有効ではありますが、それだけでは全体像を把握したことにはなりません。自社のシステムの外側にいるエンドユーザーがAPI経由で地図データを利用できているかどうかを知るにはどうしたら良いでしょうか?データが利用可能なだけではなく、適切なフォーマットであることを確かめるには?

合成/外部監視を構成して、APIからデータを引き出してテストし、適宜アラートを発生させるようにすれば、契約違反につながりかねない問題が発生した際に、直ちにエンジニアがそれを知ることができます。



上の例は、Splunk Synthetic MonitoringのAPIチェックによるアラートです。

エンドユーザーと地図システムとの実際の通信をシミュレートしたプロアクティブなデータによって、以下のことが可能になります。

1. 実際にユーザーに影響がおよぶ前に、パフォーマンスの問題に先回りして対処する。
2. パートナーにレポートを共有し、契約通りにアップタイムが確保されていることを示す。

SLA Report
Jan 18 2021, 10:46AM – Feb 17 2021, 10:46AM

SLA Report: New Panel 1
Last 30 Days (Jan 18 2021, 10:46AM to Feb 17 2021, 10:46AM)
Checks: About, Checkout and 6 more checks | Metrics: Uptime

Name	Uptime (Mean)
About	99.907%
Checkout	97.847%
Homepage (Desktop)	99.988%
Homepage (mobile)	100%
Search Flow (mobile)	99.969%
Search Flow	100%
Product page (mobile)	100%
Product page	100%

パートナーにSLAの履行を徹底させる方法

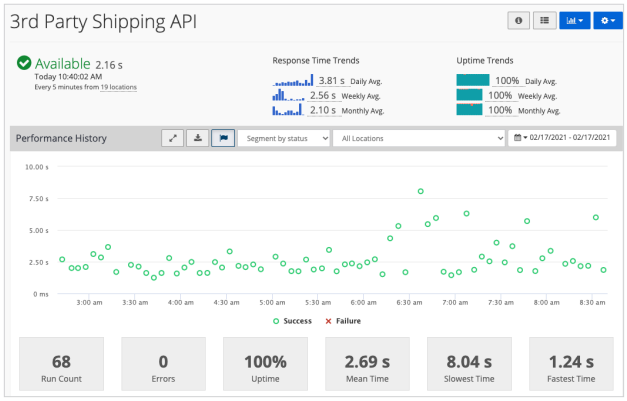
ライドシェアWebアプリ側のビジネスオーナーであれば、次のように考えるでしょう。「地図サービスプロバイダーがこのようなレポートを共有してくれるのは大変結構だが、自分たちの側でも独自の調査を行って、外部データとレポートを突き合わせてみる必要がある」。外部APIチェックを行えば、地図サービスのAPIのパフォーマンスを監視し、APIが本当に99.99%の可用性を提供しているか確認することができます。

そして、可用性がSLAを下回っていたり、3週間前に通達されていない長時間のダウンタイムが発生したりした場合には、このレポートに基づいて、契約違反の是正についての話し合いをパートナーと開始することができます。さらに、APIチェックを使用することで、パートナーのAPIでの問題が実際のユーザーにどのような影響を与えているかについて、より詳細に把握することができます。

サービスレベル契約を交わした両パートナーがAPIチェックを利用すれば、契約通りに合意内容が履行されていることを相互に確認できるようになります。APIのオーナー側は、プロアクティブな監視を行うことで、問題の影響がパートナーにおよぶ前にそれを捕捉することができ、レポートを活用してパートナーと簡単に共有できます。一方、APIのエンドユーザー側は、プロアクティブな監視を行うことで、自社のユーザーに影響するサードパーティの問題についてアラートを受け取ることができ、パートナーの契約遵守に関して一層の安心感を得ることができます。

以下は、地図サービスプロバイダーのAPIに対する多段階のAPIチェックの例です。データが期待値の範囲に収まっていることが確認でき、リクエストに対する可用性の追跡データも表示されています。

注：APIチェックは、可用性と応答性の両方の監視に使用することができます。また、APIチェック用に構成するユースケースは、いずれもSLAの要件を満たす必要があります。契約が可用性のみに基づいたものである場合は、単にエンドポイントにアクセスして、アップタイムのパーセント値を参照するようにチェックプロセスを構成すれば良いでしょう。契約がAPIからのデータ応答速度に基づいたものである場合は、APIからデータを取得して、平均応答時間と合意した基準値とを比較する多段階のチェックを構成することができます。



第5章

SplunkのAPIチェック

Splunkでは、APIが想定通りに動作しているかどうかを把握するのに役立つパフォーマンステストのフレームワークを開発しました。

APIチェックには以下の機能があります。

- 重要なAPIの可用性を追跡。
- APIの応答時間を取得して傾向を把握。
- APIが返す値と構造を検証することでAPIの機能を確認。
- APIの障害やパフォーマンス低下を示すあらゆる兆候について警告。これによって、チームはユーザーへの影響やSLA違反が発生する前に問題に対処することができます。

SplunkのAPIチェックは以下の4つのシンプルなコンセプトで構築されています。

1. HTTPリクエストを送信
2. レスポンスからデータを抽出
3. 追加リクエストや分析で使用するためにデータを保存
4. データとそのフォーマットについてアサーションを作成

このコンポーネントはシンプルですが強力です。たとえばフラクタルのように美しく複雑なものは、時として極めて基本的な構成要素によって組み立てられているものです。SplunkのAPIチェックを構成している要素はシンプルなものですが、これらを組み合わせることで、極めて複雑なAPIフローを監視、検証するようなテストケースにも対応できます。

APIチェックの開発とベータテストの期間中、Splunkのエンジニアはお客様と緊密に連携しながら作業を進め、このシンプルなアプローチでチェック機能を分かりやすいものに維持しながら、確実にお客様のニーズに応えるものとするように努めました。モバイルセキュリティソリューションの有力ベンダーであるLookout社は、このベータテストに参加してくださった会社の1つです。Lookout社のクラウドオペレーションエンジニアであるDean Ross-Smith氏は、次のように述べています。

「当社は、この柔軟なSplunkの新しいAPIチェックを使ってインフラの重要箇所を対象としたチェックプロセスを構成し、それらの監視を行えるようになりました。これは、これまでのどのソリューションでも実現できなかったことです」

ビジネスのニーズに応える設計

APIを活用しているのがWebプロパティか、モバイルアプリか、あるいは企業のコアインフラであるかを問わず、SplunkのAPIチェックでは、どのような技術レベルのユーザーであっても理解できるような方法で、APIの可用性、パフォーマンス、機能を確認することができます。

APIチェックによって解決できるいくつかのビジネスニーズを以下に紹介します。

- 多段階の複雑なAPIフローのテスト
- 世界中の複数地域における可用性と応答時間の監視
- サードパーティ APIのパフォーマンスSLAの追跡と徹底
- APIレスポンスの正確性の検証
- API経由のデータオブジェクトにおけるCRUDライフサイクル全体のテスト
- トークンベースの複雑なAPI認証システムの処理
- アプリケーションステータスページの監視

結論

自社のAPIやサードパーティのAPIに依存するアプリケーションを使っている、API経由で顧客にデータ提供している、あるいはAPIの可用性、機能、パフォーマンスの可視化を徹底する必要があるといった場合、Splunk Synthetic Monitoringの新しいAPIチェックは、その課題に対する答えとなります。SplunkのWebパフォーマンス製品の詳細やデモをご覧になりたいお客様は今すぐお問い合わせください。

より詳しい内容と技術解説については、**Splunkbase**でAPI Checkの詳細をご確認ください。

営業へのお問い合わせはこちら：https://www.splunk.com/ja_jp/talk-to-sales.html
〒100-0004 千代田区大手町1-1-1 大手町パークビルディング 8階
www.splunk.com/ja_jp
splunkjp@splunk.com

21-16715-SPLK-APIs-In-Action-TWP-JA-202107