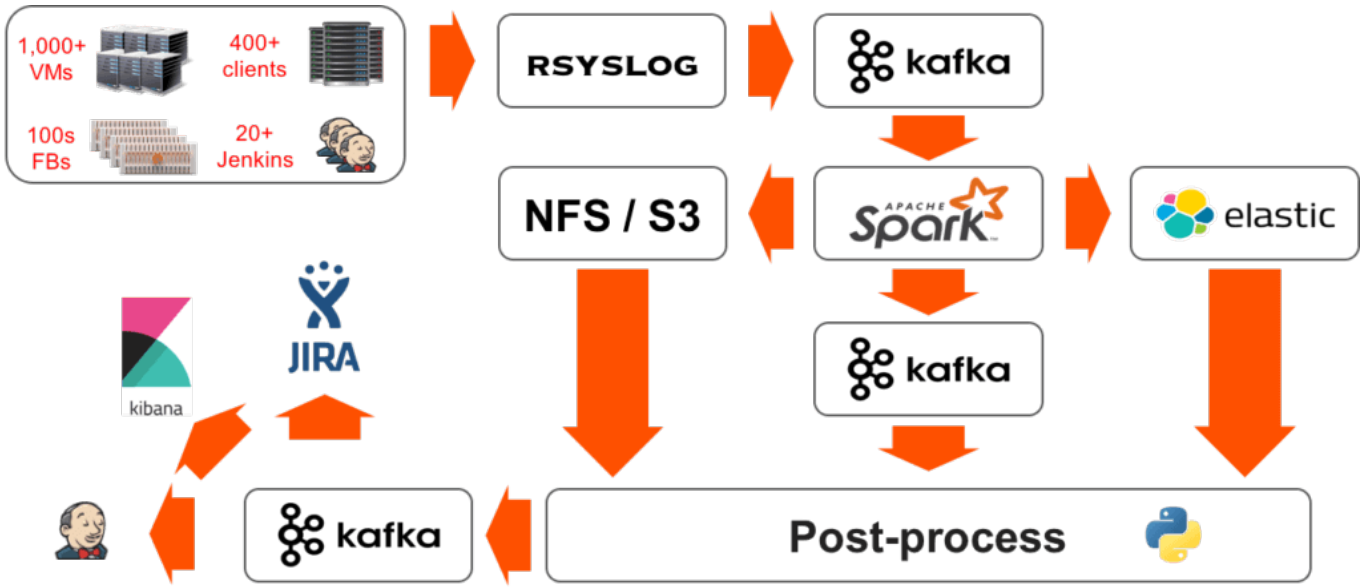
ストリーミングデータ分析でソフトウェア開発を加速する



ピュア・ストレージ・ジャパン、FlashBlade セールスリードの大浦です。

本日ご紹介する記事は、イヴァン・ジバジャ（Ivan Jibaja）による、「ストリーミングデータ分析でソフトウェア開発を加速する」です。イヴァンは現在、ビッグデータ分析の技術リードを努めています。先日までは FlashBlade 開発チームに所属しており、初期の製品開発を支えました。

彼らが限られた開発リソースの中でどのようにしてソフトウェア開発を加速させたのか、その秘密をご紹介します。

ピュア・ストレージでは、ソフトウェア開発を加速し、新機能を迅速に提供できるようにすることは、お客様の満足度向上につながると考えています。

もちろん最優先されるのはソフトウェアの品質です。ピュア・ストレージでは、開発の初期段階から継続的統合モデルを採用し、テストの自動化に十分な投資を行うことで、高品質なソフトウェアの開発を目指しています。テストのエラーからはさまざまな知見を得られますが、障害注入テストやソフトウェアシステムの複雑さが原因でデバッグが困難になることもあります。エンジニアリングチームの規模やインフラストラクチャが拡大するにつれ、実行するテストの数は増大します。デバッグのための手作業でのログ調査は、膨大な作業量を伴うため滞りがちになります。そこで私たちは、Spark や Kafka のような近代的なビッグデータツールを使ってソフトウェア QA インフラストラクチャを構築しました。その結果、1 日に 500 億件を処理し、リアルタイムで、5 秒以内にフィードバックを提供できるようになりました。

このブログでは、私たちがソフトウェアの自動テストインフラストラクチャをどのように活用したかを説明します。また、規模を拡張する過程で発生した問題や、それらの問題を解決するために、ビッグデータのリアルタイム分析を使って自動テストインフラストラクチャを拡大した方法についても説明します。

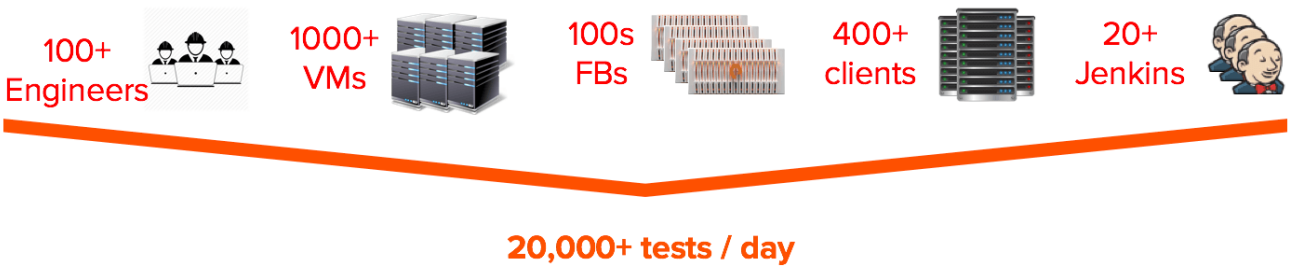
初めは 1 日数百件のテストから

スケールアウトファイルおよびオブジェクトストレージプラットフォームである FlashBlade の開発を始めたばかりの頃、私たちはまだ小規模なエンジニアチームでした。数台の製品プロトタイプと数台のサーバーを使ってお客様の負荷のシミュレーションを行っていましたが、この段階で既に、開発中のテストはできるだけ自動化しようと決めていました。当初の目標は、ソフトウェア開発のための継続的統合モデルを含む、高速反復サイクルを確立することでした。

この頃、私たちは多数のテストを抱え、1 日のテストランは数百回にのぼりました。このモデルは非常に有効で、プロトタイプの段階から FlashBlade をリリースするまでの間、アジャイル開発サイクルを可能にしてくれたので、迅速にバグを特定し、修正することができました。これが、製品の品質向上につながりました。

1 日 2 万件のテストへと増加

その後、FlashBlade に対するお客様の需要が急増したため、それに合わせて、エンジニアリングチームの規模を拡大し始めました。エンジニアの人数が増えるに従って、並行して開発する機能の数も増えていきました。機能が増えれば増えるほど、テストの件数も増え、テストのサポートに必要なインフラストラクチャ（FlashBlade サーバ、Jenkins など）も増えていきました。さらに、テストの実施規模を拡大するために FlashBlade の機能シミュレーションシステムへも投資しなければなりませんでした。個々の要素は急速に大きくなり、1 日あたりのテスト件数は、あっという間に 2 万件を超えました。



必然的に QA エンジニアが 20 人必要に

どのようなテストにも、複数の障害点があります（FlashBlade、ネットワーク、クライアント、コード行、テストケース、などです。これら多くの要素が、テストで発生したエラーの原因特定（トリアージ）を複雑な作業にしています。例えば、テストの根本的な原因を解明する場合、エラー 1 件につきエンジニア 1 人で 10 分かかるとしましょう。ここで、エラー率を 5% とすると、1 日あたり 2 万件のテストに対してエラー数は 1,000 件となります。つまり、これらのエラーの原因を特定するには、（1 日あたり 1,000 件のエラー）×（10分/エラー）÷（60分）÷（8 時間/日）となり、約 20 人のエンジニアチームにフルタイムで働いてもらわなければなりません。

革新的な新機能やソリューションの市場投入の機会が増すごとに、エンジニアリングチームが増大することになり、この問題はさらに大きく、より複雑になります。

実際には QA エンジニア 3 人で

現在、私たちのトリアージチームには数人のメンバーしかいません。トリアージを自動化していなければ、テストエラーの多くはトリアージされないまま残されているでしょう。ソフトウェアの品質を確保するには、自動化されたテストフレームワークが不可欠でした。私たちが自動化ソリューションの構築に着手したのは

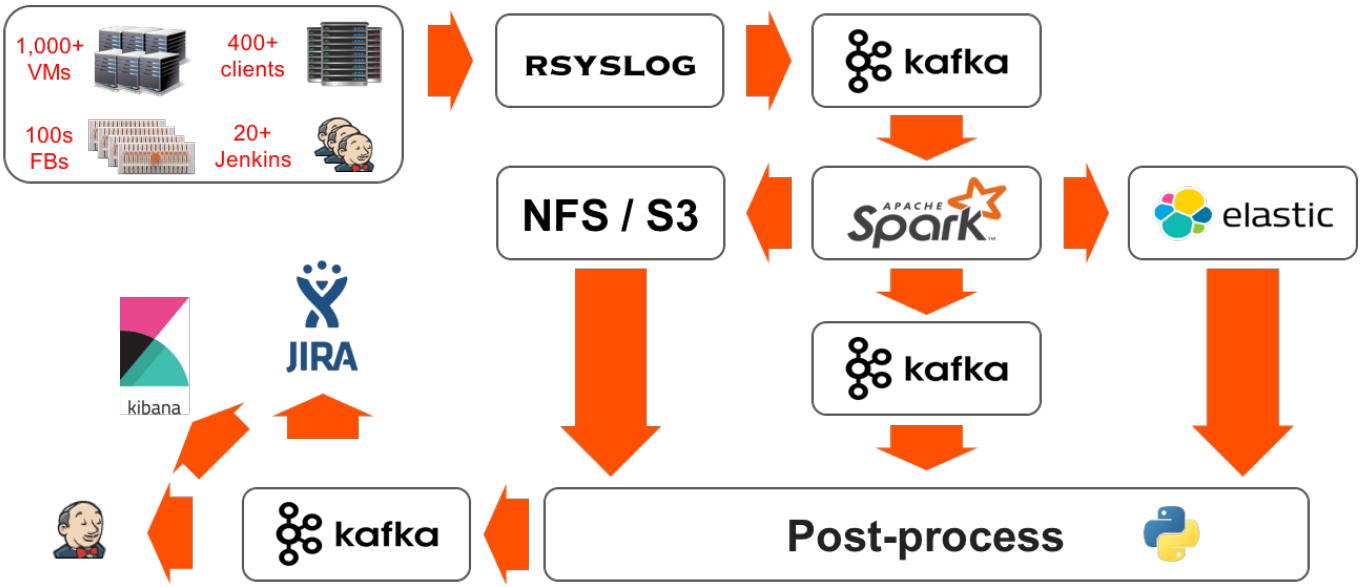
このためです。

私たちの目標とするシステムは次のようなものでした。

- **トリアージ作業の重複を最小限に抑える：**ある 1 つの不具合が、複数のテストスイート、Jenkins インスタンス、クライアント、VM などでエラーを起こすことがあります。異なるシナリオにある同一の不具合に対して 2 人のエンジニアがそれぞれ手作業でトリアージを行なっていたら、それは作業の重複になります。そこで、このような重複を最小限に抑えるシステムを構築することにしました。システムは、ある不具合を特定するパターン（シグネチャ）を使用して、その不具合を原因とする過去および未来のテストエラーをすべてトリアージします。
- **将来の使用に備えてログを保存する：**開発サイクルにおける非常に早い段階（数年前）に入り込んだ不具合が、システムの新しい部分と想定外の相互作用を引き起こしたケースがありました。万が一、あるシグネチャの最初の出現記録を時をさかのぼって見つけなければならなくなった場合に備えて、システムはインフラストラクチャ関連のログをすべて、長期間にわたって保存しておく必要があります。
- **システムの規模を簡単に拡張できる：**1年後、エンジニアリングチームの人数は 2 倍になりました。必要なインフラストラクチャの規模も 2 倍になりました。システムを簡単に拡張できるようにしておき、チームの成長に従って、さらに多くのパターンを使い、より多くのログを分析できるようにする必要があります。
- **結果をリアルタイムで提供する：**インフラストラクチャの問題の中には、より迅速な是正措置をとれるよう、不具合の発生から警告・特定までの時間を最小限に抑えたいものがあります。つまり、新たな不具合を数秒以内に特定できるシステムが必要です。
- **ログからパフォーマンスの指標を取り出す：**他の製品開発と同様に、私たちは定期的にパフォーマンステストを実行し、新バージョンの FlashBlade ソフトウェアがパフォーマンスのリグレッションを引き起こさないことを確認します。また、例えば、「ブレードがダウン」、「特定のネットワークリンクがダウン」といった、さまざまなコーナーケースを想定したテストを行っています。このシステムは、すべてのテストケースやシナリオ全体からパフォーマンス指標を収集できるものでなければなりません。

画期的な QA 自動化のために近代的なビッグデータツールを展開

上記の要件を考慮し、私たちは、インフラストラクチャからすべてのログをストリームして処理するビッグデータ分析パイプラインを選択しました。パイプラインからの手順は次のようになります。



ロギング

私たちは全インフラストラクチャでロガーを使用しています。Jenkins には、集中型 rsyslog サーバークラスタにログを送信する rsyslog 構成のコンポーネントを作成するプレタスクがあります。このプレタスクは、サービスをポーリングして、最も使用率の低い rsyslog サーバーを見つけ、負荷を分散します。

rsyslog

これはパイプラインの最初と最後の手順で、ここでは、作成された順にログが処理されます。そのため、処理の後で一意に特定できる十分なメタデータで、各行をタグ付けする必要があります。rsyslog はログの各行を json に変換し、それがどこから来たものかを表すフィールド（ホスト、コンポーネント、時刻など）を抽出し、さらに、そのときに実行していたテストに関するフィールド（Jenkins ジョブ、Jenkins ビルド ID、VM、ブランチなど）を追加します。

これらの rsyslog サーバーには、将来のために、ログをすべて、json 形式で保存する役目もあります。このようにメタデータを追加し続けているため、rsyslog から送出されるデータの量は 3 倍になります。将来のために、ファイルシステムでは、<FlashBlade>/<年>/<月>/<日>/<ホスト>/<ログ> というディレクトリツリー構造を用い、あらかじめ分類した形式でログを保存しています。この形式のおかげで、ログファイルを細かく分類することができ、再処理がしやすくなっています。いっぽう、私たちは 1 日に 50 万個のファイルを作成し、FlashBlade に保存しています。ログの再処理に必要な、これだけの量のメタデータをすべて処理するには、FlashBlade の高 IOPs が大いに活躍します。

Kafka

データ分析パイプラインで有益だとわかったのは、データのコンシューマとプロデューサを分離する特性でした。パイプラインにある任意の 2 つのコンポーネント（例えば rsyslog と Spark）の間にブローカーとして Kafka を配置すれば、コンシューマを開始/停止したり、以前停止した地点からデータの取り込みを続行したりできるようになります。また、保存に関するポリシー（3 日間）に基づいて、ほかのコンシューマ（テストパイプライン、新しいシグネチャのルックバックなど）を添付し、ある期間を反復、再生することもできます。Kafka は、NFS 経由で FlashBlade を補助ストレージとして使用するように構成されています。これは、Kafka クラスタをサポートするインフラストラクチャを気にせずに、保存に関するポリシーを動的に変えられることを意味します。

Spark

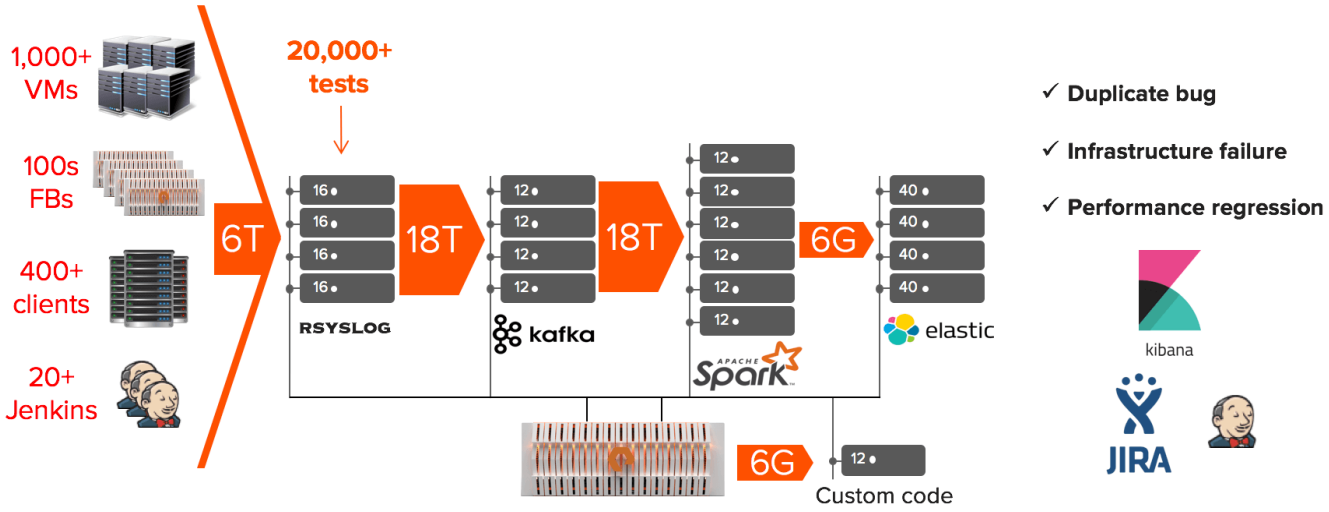
Spark は、ファイルシステム（NFS/S3）へストリームされたログと出力を分類、フィルタするパイプラインの計算コアです。フィルタ後の行のメタデータの保存には、<フィルタされたログ>/<FlashBlade>/<年>/<月>/<日>/<時>/<ホスト>/<シグネチャID>/<テキスト>という形式のファイルシステムディレクトリ構造を使います。この方法では、個々のシグネチャ（およびそれに関連するパターンすべて）を 1 時間単位で分類できるため、フィルタされたログ行の後処理が加速されます。私たちのユースケースの大部分で、一意の行の処理が行われると分かったので、そのリアルタイムの部分を最適化しました。つまり、Spark はフィルタと分類だけで、アグリゲーションを一切行わないのです。

後処理

Kafka キューの Python コンシューマを使用して、複数行とスクリプト化されたシグネチャに対し、一意のパターンの後処理とアグリゲーションを行います。後処理レイヤーは、特定の Kafka トピックから同時にプルした複数のコンシューマを持つことができるので、拡張が可能です。

レポート

私たちは、内部的に JIRA を使用して、エンジニアリング作業項目やバグ、テストエラーの追跡を行っています。したがって、JIRA は、ユーザー（ソフトウェアエンジニア）にとってはシグネチャをエンコードする、また、分析パイプラインにとっては特定のシグネチャの新たな複製を報告する理想的なツールです。ハードウェアのリソース管理とテストのスケジューリングには Jenkins を使用します。また、分析パイプラインは、テストエラーに特定の Jenkins ビルドの説明を追加して、それが「既知のエラー」であるという印をつけます。



私たちが構築した分析パイプラインは、さまざまな領域で、エンジニアリング組織に大きな影響をもたらしています。

- トリアージにかかるエンジニアの手間を省くために、重複をすべて取り除きます。
- 継続的統合ができるように、既存の不具合と新しい不具合を区別します。
- パフォーマンステストだけでなく、すべてのシナリオで、パフォーマンスの退行を追跡します。
- テストとは直接関係のないパターンや問題を受動的に収集できるようにします。
- エンジニアが、機能のパターンを長期にわたって追跡するのを支援します。

QA パイプラインの構築には近代的な分析ツールが重要ですが、パイプラインの実験と成長のスピードや俊敏性は、コンピューティングレイヤーとストレージレイヤーの分割無しには実現できなかったでしょう。この

アーキテクチャでは、私たちは、オンデマンドで大量のコンピューティングノードをスピンアップし、これらを FlashBlade に向けることにより、既存のジョブを邪魔せずに大きな分析ジョブをバッチ処理することができます。パイプラインは、FlashBlade というインフラストラクチャの上に構築されています。FlashBlade は、すべてのデータサイロを 1 つの共有システムに統合し、あらゆるデータ、アクセスパターン、アプリケーションに対するリアルタイムパフォーマンスを可能にします。

大浦からのコメント

今回の記事は弊社開発チームによるQAパイプラインについてのご紹介でした。現在、エンタープライズのハードウェア製品、自動車や家電など多くの製品が、ソフトウェアによって支えられています。

FlashBladeを用いたQAパイプラインの構築は、これらの製品の開発スキームを劇的に改善する可能性につながります。

既存のストレージシステムやビッグデータプラットフォームでは想像もつかなかったアプローチを、皆様のビジネスの加速にご活用ください。

*英語版*URL:*https://blog.purestorage.com/accelerating-software-development-streaming-data-analytics/*