New Relic.

DevOps実践ガイド

DevOpsチームを成功に導くためのベストプラクティス

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp

目次

概要		03
第1章：	SLOとリリース期間短縮のバランスを取る	04
第2章：	公正かつ効果的なオンコールポリシーの作成	08
第3章：	効率的かつ効果的なインシデント対応	11
第4章：	マイクロサービスの複雑性の克服	14
第5章：	データを活用した開発スピードの向上	17
結論		19

22
DEP

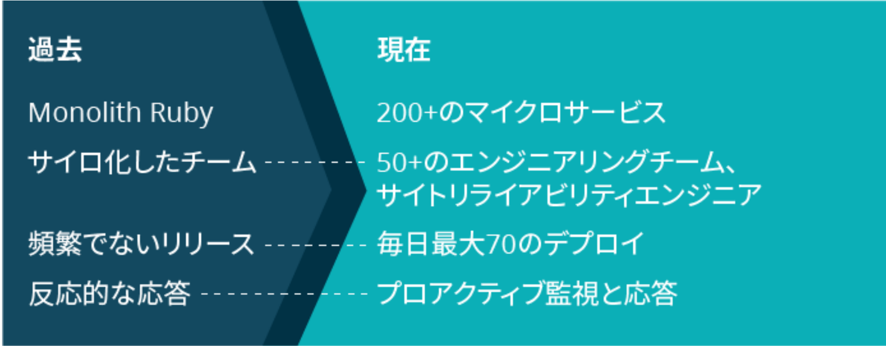
概要

DevOpsを取り入れる企業が増えてきています。新しい開発プロセスの確立、新しいツールの導入、役割を超えて連携するカルチャーの醸成などを通じて、多くの企業がDevOpsの実践を試みていますが、まだまだ効率性、品質などの観点で向上できる余地が残っているように感じます。よりハイパフォーマンスなDevOpsチームになるためには何が足りていないのでしょうか？

多くの場合、それはデータの測定です。測定は、DevOpsエキスパートのJez Humble作の造語CALMSフレームワーク（Culture（文化）、Automation（自動化）、Lean（無駄がない）、Measurement（測定）、Sharing（共有））の5つの柱の1つですが、多くの場合、DevOpsチームは速度と自主性を高めるためになおざりにします。しかしながら、効果的なインシデント・レスポンスから複雑なマイクロサービス間の移動およびその他に至るまで、DevOpsチームがうまく機能するためには正確なデータが不可欠となるため、これは重大な問題を作り出しかねません。

本eBookは、DevOpsの試験導入を通して、全面的にDevOpsを採用することになったチームや組織のために書かれたものです。また、DevOpsに取り組んでいるが完全なデジタルトランスフォーメーションが達成できていないと感じているチームや組織も対象としています。

実世界の経験、特にここNew Relicで私たちが学んだ教訓を共有することで、貴社のDevOps成功の達成を妨げている障壁を打ち破るお手伝いをさせていただければ幸いです。サービス品質目標（SLO）の設定からお客様特有のコミュニケーションおよびマイクロサービス開発アプローチの問題解決に至るまで、これまで以上にスピーディかつ効果的に進めるための実証済みベストプラクティスをまとめました。



New Relic の沿革：DevOpsへの道

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp
New Relic.

第1章

SLOとリリース期間短縮のバランスを取る

SLOとリリース期間短縮のバランスを取る

早い開発サイクルで頻繁にコードのデプロイをしているかもしれませんが、信頼性はどうでしょうか？品質と信頼性は成功をもたらすDevOpsアプローチにとって同等に重要なものです。

そこで SREの登場です。SRE (Site Reliability Engineering = サイト信頼性エンジニアリング)は従来の開発、運用、その他ITグループなど別々に行われる業務の枠を超えて管理する役割を担います。SREは開発(Dev)と運用(Ops)双方と密に連携するコラボレーションに依存するため、DevOps文化と緊密な関係にあります。DevOpsとSREは共通点が多い一方、SREは継続的な改善と測定可能な結果（特にSLO (Service Level Objective)の使用を通して）を得ることに焦点を当てています。

それでは、重要な定義から始めましょう：

用語	定義	例
サービスレベルインジケータ(SLI)	SLIはパフォーマンスその中核測定です。	「顧客はログインして自身のデータを表示できます…」
サービスレベルオブジェクティブ(SLO)	SLOはシステムパフォーマンスのターゲット値または目標です。SLOは継続的なコミットメントを表します。	「99.9%の時間…」
サービスレベル契約(SLA)	SLAはあなたのSLI/SLOコミットメントが満たされない場合に何が起きるかを定義します。	「…またはサービスの不可用性に起因した損失の返金を要求できます。」

関連するeBookでSREについてもっと知るサイトリライアビリティエンジニアリング：SRE成功のための哲学、習慣、ツール

「基本的には、[[SREは]ソフトウェアエンジニアに運用機能の設計を依頼した場合に発生することです。」

Ben Treynor Sloss、Googleのエンジニアリング担当バイスプレジデント

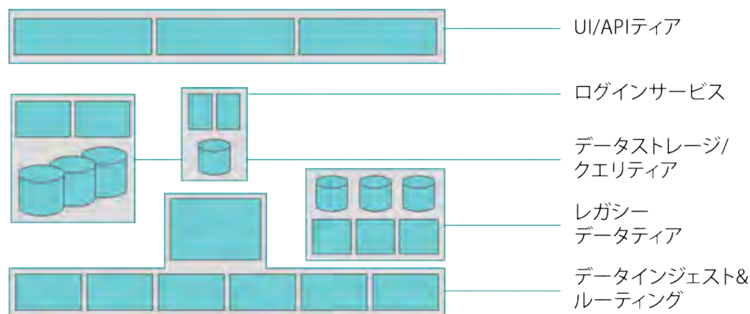
適切なSLIとSLOの設定

SREの業界ベストプラクティスには提供するサービスごとにSLIとSLOの設定が必要となりますが、経験がなければそれらを定義してデプロイするのはかなり困難です。

以下は、New RelicでSLOとSLIを設定するために使用する7つのステップです。

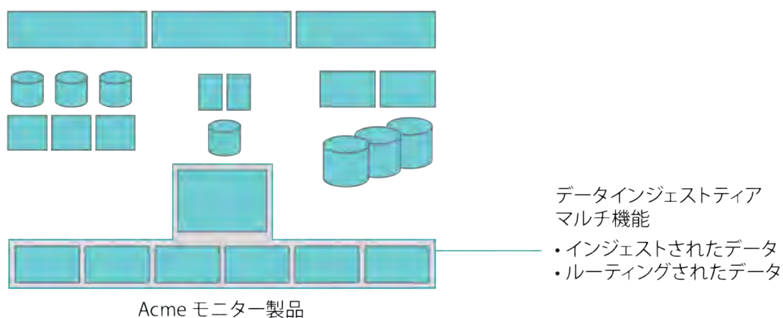
1. **システム境界を識別する：**システム境界は、1つまたは複数のコンポーネントが1つまたは複数の機能を外部のお客様向けに公開されている場所です。内部的にプラットフォームには多くの可動部分（サービスノード、データベース、ロードバランサーなど）がありますが、個々の部分は機能を直接顧客に公開していないため、システムの境界とはみなされません。複数の部分が連携して、全体的な機能を公開します。たとえば、ユーザー資格情報を認証する機能を備えたAPIを公開するログインサービスは、システムとして連携して機能するコンポーネントの論理グループです。SLIを設定する前に、プラットフォームの要素をシステムにグループ化し、それらのシステム境界を定義することから始めます。境界SLIとSLOが最も有益であるため、残りのステップで集中させます。

© 2020 New Relic株式会社
| Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp
New Relic.



プラットフォーム内のシステムと境界

2. **各システムによりさらされる機能を定義する：**次に、プラットフォームのコンポーネントを論理単位（UI/APIティア、ログインサービス、データ・ストレージ/クエリティア、レガシーデータティア、データインジェスト、ルーティングなど）にまとめます。ここNew Relicでは、当社のシステム境界はエンジニアリングチームの境界に合わせます。これらのグループ化を使用して、各システム境界でさらされている機能群を明らかにします。



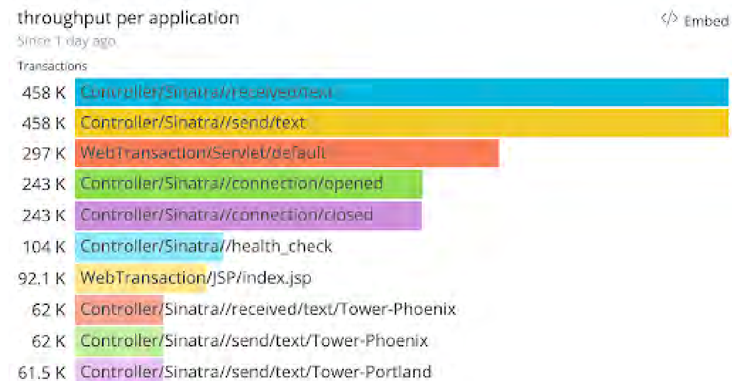
システム境界で定義されている機能群

3. **各機能の「可用性」の明確な定義をします：**たとえば、「正しい宛先へのメッセージの配信」は、データルーティング機能の可用性への期待事項を説明する一つの方法です。平易な英語を使用して可用性になにが期待されるかを説明することで、専門家しか知らない技術用語を使用した場合に生じ得る誤解を避けることができます。

4. **対応する技術SLIを定義します：**次に、各機能の可用性の定義を使用して、各機能ごとに1つまたは複数のSLIを定義します。上記の例を使用すると、データルーティング機能のSLIの1つは「正しい宛先にメッセージを配信する時間」などとして定義できます。

5. **ベースラインを取得する手段：**可用性目標を達成できたかどうかを知るための手段は明らかにモニタリングです。実際にSLOを設定する前に、モニタリングツールを使用して各SLIのベースラインデータを収集します。

6. **SLOターゲットの適用（SLI/機能ごと）：**データを取得したら、SLOを設定する前に、顧客の期待事項を特定し、それを満たすためにSLOをどのように対応させるかを決定する助けとなる質問をします。その後、ベースライン、顧客から聞き出したこと、チームがサポートにコミットできること、および現存の技術に基づいて妥当なことに基づいてSLIターゲットを選択します。上記のデータルーティング用のSLI例を使用する場合、SLOとして「5秒以内に99.5%のメッセークが配信される」などが考えられます。定義するSLOの警告閾値のあるモニタリングアプリケーション内でアラートトリガーを設定することをお忘れなく。



データルーティング機能用SLIの例

7. **反復と微調整：**SLOとSLIは、「設定したことを忘れない」ようにしてください。これらはサービスと顧客のニーズの変化に伴い経時的に進化するであろうことを仮定すべきです。

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp New Relic.

SLOとSLI設定のヒント

- **システムの各論理インスタンスがそれ自身のSLOを持つことを確認してください：**たとえば、ハードシャード（水平スケールに比べて）システムでは、各シャードに対してSLIとSLOを別々に測定します。
- **SLIはアラートとは同じでないことを認識する：**SREプロセスは徹底的なアラートを置き換えるものではありません。
- **適切な場合は合成SLOを使用する：**単一の合成SLOを表現して複数SLI状態を取り込み、顧客が理解しやすくできます。
- **必要に応じて顧客特有のSLOを作成する：**主要顧客が他の顧客に提供されたSLAよりサービス可用性に優れたSLAを受け取ることは珍しくありません。

「運用のエクセレンスを達成するために、私たちはすべてを測定します。これを行うことでのみ、すべてを管理し改善を図ることができます。」

CarRentals.com、DevOpsディレクター、Craig Vandeputte

© 2020 New Relic株式会社
| Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp

第2章

公正かつ効果的な
オンコールポリシーの作成

公正かつ効果的なオンコールポリシーを作成する

デプロイメントを加速しながら信頼性を改善する次のステップは、お客様の組織が昼夜を問わずいつでも発生したソフトウェアの不具合を迅速かつ効果的に処置できることの確認です。これには、オンコールポリシーが必要です。

次の章にはまだ進まないでください。「オンコール」という用語は人々に多くの感情を引き起こし得ます。これは主として多くの組織がオンコールローテーションの概念を誤解しているためです。そのため、顧客にあなたのSLAがないことによるストレスを感じ、非難を受けるだけでなく、消耗して苛立ったエンジニアのチームの非生産的で不愉快な雰囲気の中で働くことになります。

基礎から始める

効果的で構成なオンコールポリシーは2つの重要な前提条件から始まります：

1. **構造化システムおよび組織：**不具合に効果的に対応するのはあなたのシステム（サービスまたはアプリケーション）と製品チームが論理ユニットにうまく組織化されており構造化されている必要があります。たとえば、New Relicでは、当社の57のエンジニアチームが200件の個別サービスをサポートしており、各チームは設計からデプロイメントまでの製品ライフサイクル全体を通して自主的に最低3件のサービスを担当します。
2. **責任の文化：**DevOpsを使用して各チームは本番にデプロイするコードの責任を負います。当然のことながら、チームがサービスの責任を負い、オンコールである場合の変更およびデプロイメントに関して異なる意思決定を行い、本番で稼働し始めたらコードのサポートの責任が他者に移る従来環境とは異なります。

これらのベストプラクティスをオンコール実践の改善に適用する

チームと組織を公正に構造化

ここNew Relicでは、製品部門のエンジニアおよびエンジニアリングマネージャは、交替でチームのサービスのオンコール責任をローテーションします。各チームは少なくとも3件のサービスの責任を負いますが、サポートするサービス件数はサービスの複雑さとチームのサイズによって異なります。あなたの組織では、オンコールローテーションアプローチを選択する前にエンジニアリング部門全体のサイズと個別チームのサイズをお調べください。たとえば、チームに6人のエンジニアがいる場合、各エンジニアは6週に一度オンコールの主要人物にできます。

柔軟で創造性に富むローテーションをデザインする必要があります。

各チームに自身のチームのオンコールローテーションポリシーを設計し実装されることを考慮してください。チームに自由と自主性を与え、自身のチームに最適なローテーションを組織化するための型にはまらない思考ができるようにします。New Relicでは、各チームは自主的に自身のオンコールシステムを作成して実装します。たとえば、1つのチームは主要人物以外のオンコールの順番をランダムにローテートするスクリプトを使用しています。

メトリックスを追跡してインシデントをモニターする

公正で効果的なオンコールローテーションを策定するために重要なことは、インシデントメトリックスのモニターです。ここNew Relicでは、呼び出し数、呼び出し時間数、時間外よびだし数を追跡しています。これらのメトリックスをエンジニア、チーム、グループレベルで追跡します。メトリックスの追跡は、対応できない呼び出し負荷に直面しているチームの注意を引く助けとな

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp New Relic.

ります（チームの平均が週あたり時間外呼び出し1件を超える場合、そのチームは高オンコール負荷を持つと見なされます）。これらのメトリックスを追跡することで、チームの技術負債を支払うかサポートを増やしてサービスを改善するかの優先順位をシフトできます。

「[DevOps] の高パフォーマーは収益性、市場シェア、および生産性に対する自身の目標を超える可能性が2倍高くなっています。」[1]

ポリシーを貴社の状況に適合

New Relicのチームに適したオンコールポリシーは貴社では全く維持不可能であるかもしれません。公正かつ効果的なオンコールローテーションを作成するには、以下のような追加入力を考慮してください。

- **成長：**貴社および貴社のエンジニアリンググループの成長速度は？売上高は？
- **地理：**貴社のエンジニアリング部門は中央化されていますか、それとも地理的に分散していますか？フォローザサン（Follow the Sun）ローテーションを展開するリソースはありますか？
- **複雑さ：**アプリケーションの複雑さの程度と構造化方法は？サービス全体を通しての依存関係はどの程度複雑ですか？
- **ツール：**エンジニアに自動的に行動可能な問題通知を提供するインシデント対応ツールはありますか？
- **慣例：**貴社のエンジニアリング部門の慣例でオンコールが職務の重要部分となっていますか？他者を責めるのではなく根本原因を見つけて解決しようとする、非難をしない慣例を実践していますか？

1：出典：“2017 State of DevOps Report,” Puppet and DORA

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp New Relic.

第3章

効率的かつ効果的なインシデント対応

効率的かつ効果的なインシデント対応

オンコールローテーションに付随するインシデント管理概念。インシデントとは？システムが顧客（またはパートナーあるいは従業員）に悪影響を及ぼす可能性があるような予期されない振る舞いをすること。

「自分で構築して自分で所有する」というDevOpsアプローチ内のコアコンピテンシーの一つであるインシデント管理は軽視されることが多く、問題が解決されるとチームは興味を失います。効果的なインシデント管理のない組織は多くの場合、アドホック組織、方法、コミュニケーションを用いて「火消し」アプローチを取ります。何かが爆発したら、誰もが一丸となって必死にその問題を解決しようとします。

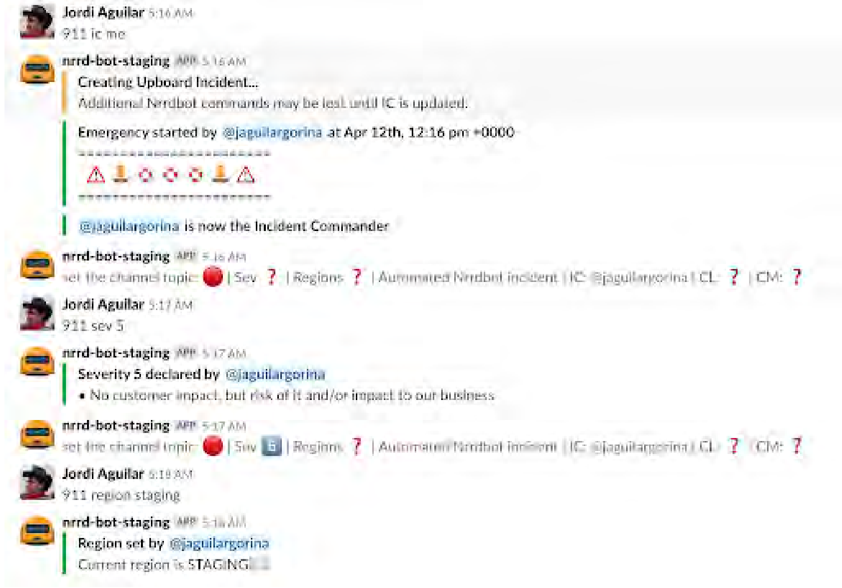
これより良いインシデント対処方法として、サービス停止時間と頻度を最小化できるだけでなく、担当エンジニアが効率的かつ効果的に対応するために必要なサポートを提供するものがあります。

効果的なインシデント管理プロセスを作成する

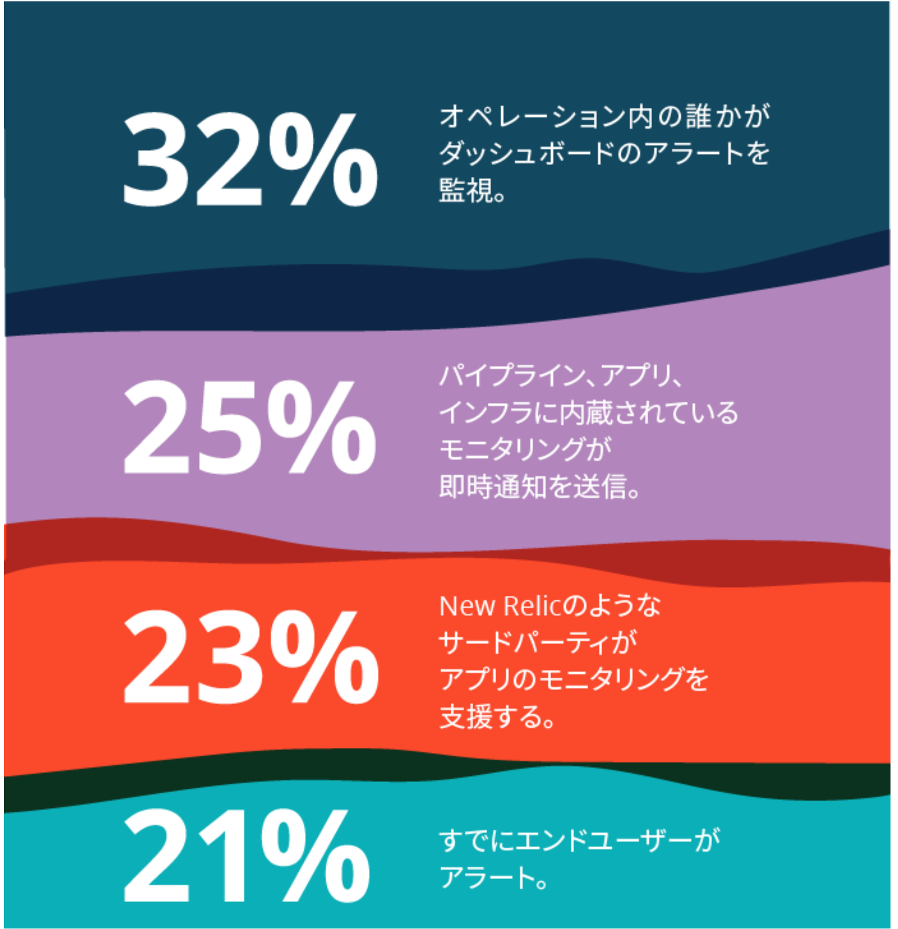
1. **重大度を定義する：**重大度は、必要なサポート、顧客への潜在的な影響の程度を定義します。たとえば、New Relicでは、重大度を1～5の5レベルで示します。
    - レベル5は顧客への影響はなく、問題についての意識を高めるために使用されます。
    - レベル4は顧客に影響はあるが、支障はない軽度のバグや軽度のデータ遅延を示します。
    - レベル3は重度のデータ遅延または機能が使用できない状態を示します。
    - レベル2と1はサービス停止を伴う重大インシデントです。
2. **サービスをインストゥルメントする：**どのサービスも、プロアクティブなインシデントレポートのためのモニターおよびアラートシステムを持たせるべきです。目標は、顧客より前にインシデントを発見することで、苛立った顧客がサポートに連絡してくるか、ソーシャルメディアにコメントを投稿するという最悪シナリオを避けることです。プロアクティブなインシデントレポート作成により、早急に対応してインシデントを解決できます。
3. **レスポンダーロールを定義する：**New Relicでは、インシデント時にはエンジニアリングとサポートのチームメンバーが以下のロールを行います：インシデントコマンダー（解決を進める）、テックリード（診断と修正）、コミュニケーションリード（全員に情報通知）、コミュニケーションマネージャ（緊急通信計画の手配）、インシデントリエゾン（重大度1でサポートおよびビジネスと連絡）緊急コマンダー（重大度1ではオプション）、エンジニアリングマネージャ（インシデントオのプロセスを管理）。
4. **作戦を立てる：**これはインシデントのライフサイクル全体をとおして発生したすべてをカバーする役割ごとの一連のタスクで、インシデントの宣言、重大度の設定、連絡すべき適切なテックリードの決定、問題のデバッグと修正、コミュニケーションフローの管理、責任の分配、インシデントの終了、回顧録を含みます。
5. **適切なツールとオートメーションを実装してプロセス全体をサポートする：**適切なチームメンバーへの情報提供、タスクへの従事、および作戦の効率的な実践を維持するためには、モニターとアラートからダッシュボードとインシデント追跡まで、プロセスの自動化が不可欠です。

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp
New Relic.

6. **回顧：**インシデント後1～2日以内に、チームにインシデントを振り返って回顧録するように求めます。回顧には非難を含めず、問題の真の根本原因を追求することに的を絞ります。
7. **インシデントインシデントを繰返さない(DRI)ポリシーを実践する：**サービス上の問題が顧客に影響を及ぼす場合、技術的な負債を特定して返済する時期です。DRIポリシーには、問題の根本原因が修正されるか緩和されるまでそのサービス関連の新しい業務を停止することが書かれています。



Slack（スラック）でのインシデント宣言の例



2：出典：“DevOps Survey Results,” 2nd Watch, 2018.

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp New Relic.

第4章

マイクロサービスの複雑性の克服

マイクロサービスの複雑性の克服

マイクロサービスとDevOpsは、ピーナツバターとチョコレートのように相性が良い。現在、会社はモノリシックアプリケーションを分離されたサービスに変換することにより生産性、スピード、俊敏性、スケーラビリティ、信頼性が劇的に向上することを理解しています。

しかし、チームはマイクロサービスの開発、テスト、デプロイに必要な変更は認識しているものの、多くの場合コラボレーションやコミュニケーションに必要な大幅な変更を見過ごしています。New Relicのエンジニアは、マイクロサービスの世界につきもののコラボレーション環境の複雑さとコミュニケーション課題の簡素化を育てるための以下のベストプラクティスを開発しました。

マイクロサービス環境における良好なコミュニケーションの実践

- **上下の依存関係に主な変更を知らせる：**マイクロサービスに主な変化をデプロイする前に、それに依存する上下のチームに通知して、問題が発生した場合にその根本原因を追跡しようとして時間を浪費しないで済むようにします。
- **早めに頻繁に連絡する：**これはバージョニング、廃止、一時的に下位互換性を提供する必要がある状況では特に重要です。
- **内部APIを外部APIと同様に取り扱う：**文書化、情報エラーメッセージ、テストデータ送信プロセスを使用して、APIが開発者にとって使いやすいものにします。
- **下流のチームを顧客と同様に取り扱う：**アーキテクチャダイアグラム、説明、ローカル実行のための説明、貢献の仕方に関する情報を含めたREADMEを作成します。
- **アナウンス専用チャネルを作成する：**重要なアナウンスに対しては、チームが複数のボード、ディスカッション、メールから重要情報を抽出するのではなく、真実を伝えるシングルソースをもたせることが重要です。
- **サービスの「近隣」を重視する：**上流、下流、インフラ、セキュリティチームはすべてサービスの「隣人」です。良い隣人として、彼らのデモやスタンドアップに参加し、あなたの隣人があなたのサービスへのロードマップにアクセスできるようにし、連絡リストを維持すべきです。

マイクロサービスの機能の仕組みをより良く理解するためのデータの使用

モノリシックアプリケーションをマイクロサービスに分離するのは容易な作業ではありません。システムをサービス境界に区分する前に、まずシステムの深い理解が必要です。そうした後でも、完全なマイクロサービス（単一機能、細粒度、かつアプリケーションや他のサービスとの結合がゆるいもの）を作成することは容易ではありません。

以下に、他のサービスやアプリケーションとの相互依存関係がまだ多すぎる環境で、マイクロサービスを検出するために使用できるいくつかのメトリックとモニタリングのヒントを示します。

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp New Relic.

1. **デプロイメント：**あなたのチームはデプロイメントを上流および下流のチーム全体を通して同期していますか？複数のサービス間で同期されているモニターソリューションにデプロイメントマーカーが表示されている場合、サービスを完全に分離したとは言えません。
2. **コミュニケーション：**マイクロサービスがその機能を実行するために必要な他のサービスとのコミュニケーションは最小限のはずです。サービスに同一の下流サービスとの間で多数の要求のやり取りがある場合、それは分離されていないことを明らかに示しています。スループットスループットもチェックすべきマーカーです。あるマイクロサービスへの毎分のコール数がアプリケーション全体のスループットに比べて著しく多い場合、これはサービスが分離されていないことを明確に示しています。
3. データストア：各マイクロサービスには、デプロイメント上の問題、データベース競合上の問題、データストアを共有する他のサービスで問題を引き起こすようなスキーマの変更を防止するためにそれ自身のデータストアをもたせるべきです。適切なモニターにより各マイクロサービスがそれ自身のデータストアをしようしているかを見ることができます。
4. スケーラビリティ：完全なマイクロサービス環境では、ホスト上にあるサービスのスパイクは各サービス上のスループットのスパイクに対応します。これは、マイクロサービスの最大メリットの一つであるデアルダイナミックスケーリングを示します。一方、対応するスパイクが全サービスおよびホストにわたって見られる場合、それはサービスが分離されていないことを示しています。
5. **アプリケーションごとの開発者：**マイクロサービスチーム間で効果的なコミュニケーションがある場合、各マイクロサービスに触れて良好な動作を確認する「アーキテクチャのグル」は必要ありません。100人のエンジニアと10のサービスがり、そのうち2人のエンジニアが10のすべてのサービスを開発している場合は、おそらくそれらのサービスが実際には分離されていなことを示しています。すべてのサービスの開発がその2人の開発者のトライバルナレッジとコミュニケーションスキルに依存していることを示していると言えるでしょう。

ほとんどはマイクロサービスとコンテナが不可欠だと考える[3]

80%

マイクロサービスとコンテナベースのイネーブルメント機能が不可欠、非常に重要または重要と考えているが、自社がそれらの機能を迅速に提供できると考えているのは4人に1人にすぎない。

3：出典：“Enterprise Priorities for Hybrid Cloud Management,” Ponemon Institute, June 2018.

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp

第5章

データを活用した開発スピードの向上

データを活用した開発スピードの向上

DevOpsの基礎的な特徴はスピードつまり高速なソフトウェアの配信、高速な問題解決、高速なイノベーションです。それでは、いかにして貴社は市場機会を獲得し、競争で他社をしのぎ、顧客の満足を維持するのに必要なスピードを達成できるでしょうか？

New Relicは、以下の理由でデータがDevOpsの成功の鍵であることを知っています。

- DevOpsのパフォーマンスを測定して追跡する誰もが正しいことに集中できるように即時のフィードバックを提供する
- ソフトウェア配信、パフォーマンス、ビジネスの結果を最適化する

DevOpsに関して顧客から問い合わせのある最も一般的な質問の一つは「どのように始めるべきですか？」です。DevOpsへのジャーニーをガイドするための中核となる原則は次のとおりです：

データサイロを排除する

測定はDevOpsの中核ですが、多くのチームはデータサイロがパフォーマンスに関して各人が見ていることが異なることを意味していることを認識していません。パフォーマンスデータのサイロがある場合、アプリケーション、インフラ、およびユーザーエクスペリエンスを通しての一貫した共通言語がなく、1つのプラットフォームですべてを表示するために、すべてのシステム全体でインストゥルメンテーションをデプロイすることで、共通データとメトリックスを使用して異なる機能を1つのチームにもたらし、全員が同じページに表示されます。ここNew Relicでは、エンジニアリングチームはツールの急増と競合する優先事項に苦慮していました。どのSLOが最も重要であるかの合意を得て、それらのSLOを共有ダッシュボードに配置することで、これらのチームが同一方向に進み始めるようになりました。

複雑さを簡素化する

最新のアプリケーションアーキテクチャは多くの方法で開発を簡素化し加速化する助けになりますが、今日のモジュラーアーキテクチャのダイナミックな品質により、一貫した全体を形成している多くの個別コンポーネント間での新しいタイプの複雑さが生じています。アーキテクチャの全体像を認識することは、それがいかに一過性のものであってもこの複雑さに対応する鍵となります。正しいデータを持つことで、何がうまく機能しており、チームがどこに焦点を合わせるべきかがわかります。

変更の影響を理解する

DevOpsでは、コードのデプロイと変更はより頻繁で、あなたのチームはそれらをうまく把握して問題の発生を回避する必要があります。最近のデプロイメントとデプロイ前後のアプリケーションパフォーマンスおよび顧客体験への影響（発生したエラーを含む）を示すレポート機能があることを確認してください。これにより、変更を潜在的な影響に素早く相関させ、発生したインシデントにチームが素早く対応し、リリースをロールバックするか、即時の解決を提供できるようになります。

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp New Relic.

結論

DevOpsの人気は主流の割合に達し、多くの異なる業界が、DevOpsの原則の採用によりスピード、生産性、品質の改善、およびイノベーションの向上を追求しています。最適なデジタル速度に達することで、高パフォーマンスのDevOps組織は低パフォーマンスの組織に比べてダウンタイムからの回復時間が劇的に短縮され、より頻繁にデプロイすることが示されています。

高パフォーマンスDevOpsエンジンの燃料はデータです。このeブックにあるようなベストプラクティスはすべて、そのデータをいかに効果的に使用して成功に導くかです。New Relicは、DevOpsの効果をモニターし、各ステージの成果を継続的に改善するために必要なエンド・ツー・エンドの可視性を提供します。

DevOpsの成功はここからnewrelic.co.jp/devopsから始めてください。

優れたモニタリングはパフォーマンスが高いチームの要です。過去数年間、アプリケーションとインフラストラクチャを積極的にモニタリングし、この情報を使用してビジネス上の意思決定を行うことがソフトウェア配信のパフォーマンスに緊密に関連していることがわかりました。[4]

4：出典：“Accelerate: State of DevOps 2018: Strategies for a New Economy,” DORA

© 2020 New Relic株式会社 | Email japan_marketing@newrelic.com | Phone 03-4577-9065 | www.newrelic.co.jp | www.twitter.com/newrelic | blog.newrelic.co.jp New Relic.

©2008-20 New Relic, Inc. All rights reserved. 04.2020