# 利用者数800万人突破の「家族アルバム みてね」に学ぶクラウドセキュリティの勘所

ITmedia Security Week 2021春

Isao Shimizu
mixi, Inc.

# 自己紹介

- 清水 勲（しみず いさお） @isaoshimizu
- 株式会社ミクシィ みてね事業部 開発グループ SREチームリーダー
- 経歴
  - SIerで受託開発、プロダクト開発を経験後、2011年に株式会社ミクシィへ入社
  - SNS mixi インフラ運用、モンスターストライクのSREを経て、現在「家族アルバム みてね」のSRE

  - 2020年1月 **SRE NEXT 2020 IN TOKYO**
    **SREがセキュアなWebシステムを構築、維持するためにやれることはなにか**
  - 2019年5月 **AWS Summit Tokyo**
    **コンテナ移行ってこんなに大変？ ～「家族アルバム みてね」を支えるインフラの裏側～**

# アジェンダ

1. 会社とサービスの紹介
2. どのようにクラウドを活用しているのか
3. クラウドにおけるセキュリティの考慮事項
4. コンテナ化におけるセキュリティ
5. 新機能や新サービス開発におけるセキュリティ
6. 今後の展望

# アジェンダ

1. 会社とサービスの紹介
2. どのようにクラウドを活用しているのか
3. クラウドにおけるセキュリティの考慮事項
4. コンテナ化におけるセキュリティ
5. 新機能や新サービス開発におけるセキュリティ
6. 今後の展望

# ミクシィグループの事業

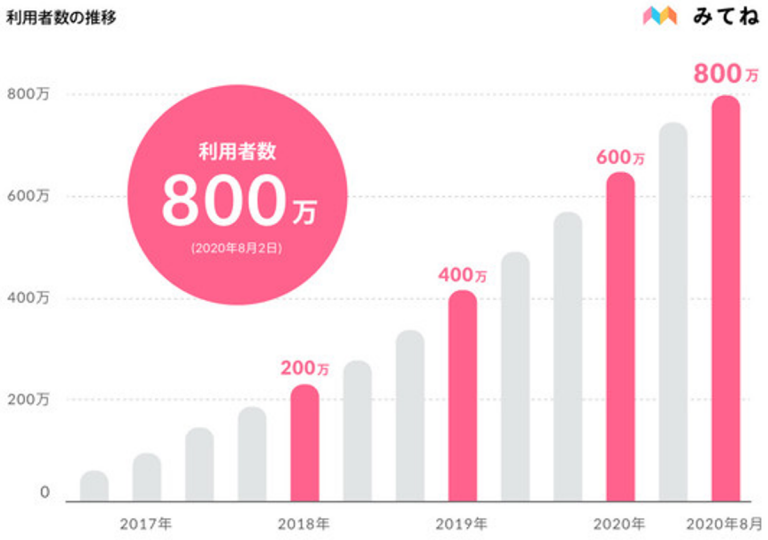
- **デジタルエンターテインメント事業**
  - スマホゲームを中心としたゲームの提供
    - モンスターストライク、コトダマン、スタースマッシュ
- **スポーツ事業**
  - プロスポーツチーム運営及び公営競技ビジネスの推進
    - TIPSTAR、チャリロト、netkeirin、netkeiba.com、千葉ジェッツ、Unlim
- **ライフスタイル事業**
  - インターネットを活用した人々の生活に密着したサービスの提供
    - **家族アルバム みてね**、minimo、mixi、FIND JOB!

# ミクシィのCSIRT 「mixirt（ミクサート）」

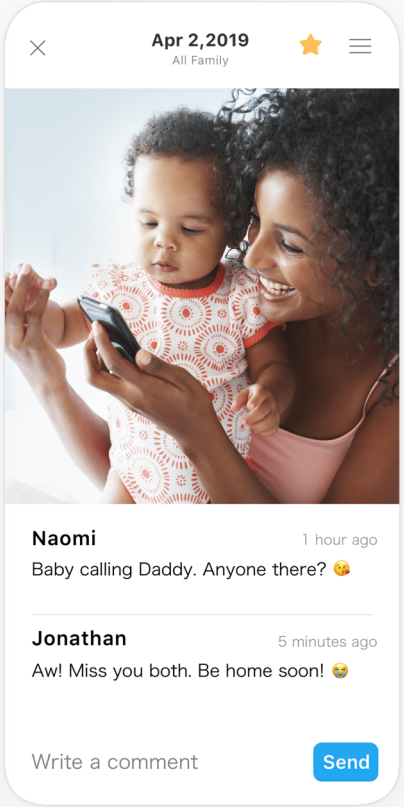
- FIND JOB! が DDoS 攻撃を受けた事故をきっかけに、2008 年 2 月 1 日に発足
- 情報漏洩等の事故が発生した際、速やかな対応行動を行い、被害を最小限に止めるための支援組織
  - 事業部門のセキュリティ担当者と、社内の情報システム部門を中心に、広報、法務などのコーポレート部門の担当者などから構成されるCISO(Chief Information Security Officer)直下の組織
- メールやSlackなどでいつでも相談、連絡ができる
- 連絡後のインシデント対応の流れ
  - 体制の整備
  - 対応方針の作成
  - 対応
  - 復旧の確認・報告
  - 事後対応
- レポートライン、ハンドリング、対応記録、ナレッジ共有などを支援

「家族アルバム みてね」

# 世界中の家族の
# “こころのインフラ”をつくる








# 全世界800万人以上の利用者

※ (株) ミクシィ調べ。 iOS・Androidアプリ登録者数、ブラウザ版登録者数の合計

# 直近の新機能リリース 🎉

- みてね 出張撮影（9月）
- みてね ウィジェット（9月）
- みてね 年賀状（10月）
- 高画質版フォトブック（10月）
  - ライト（従来のタイプ）
  - NEW スタンダード
  - NEW ハードカバー
- 写真プリント（11月）
  - オリジナル
  - ましかく、L版（1月）
- みてね カレンダー（11月）
- みてね ギフト（11月）


高画質版フォトブック


写真プリント



# アジェンダ

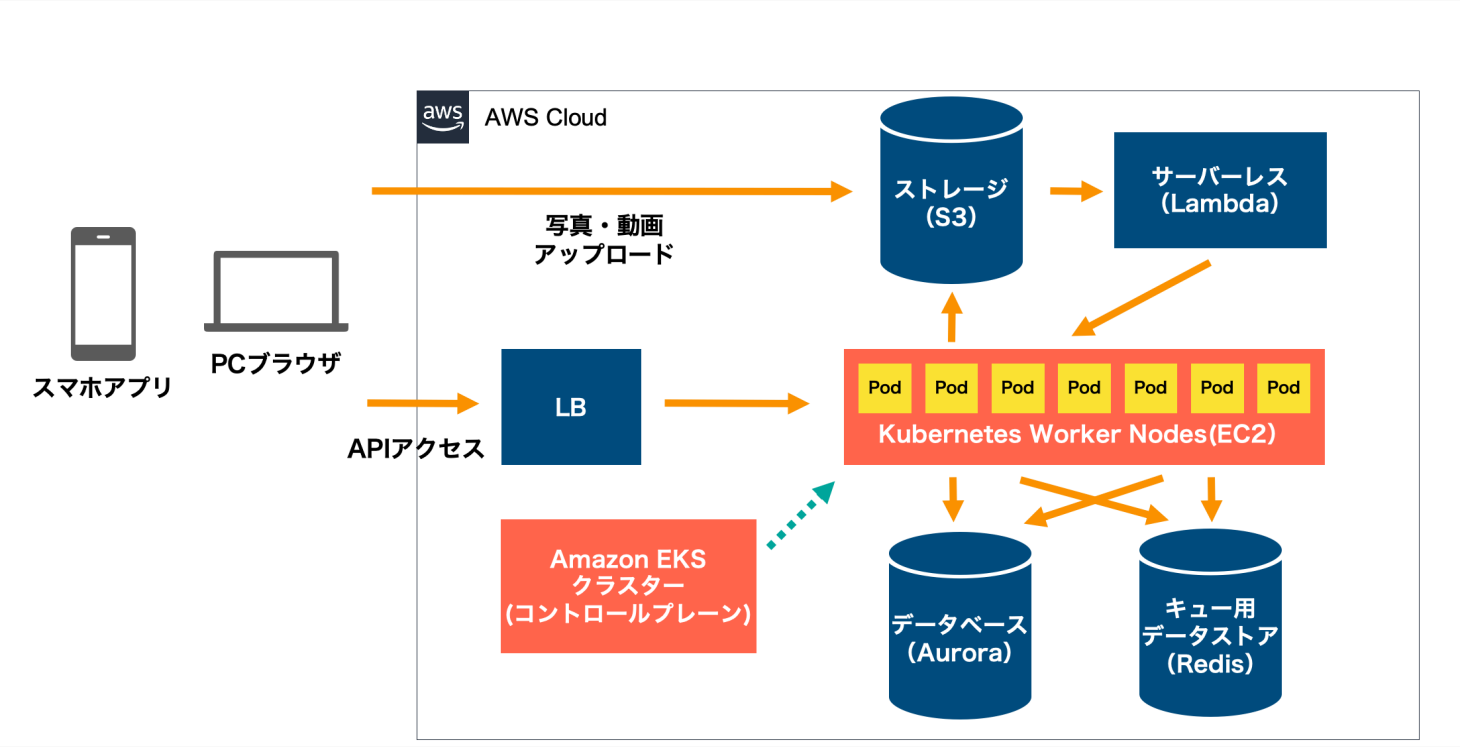
1. 会社とサービスの紹介
2. どのようにクラウドを活用しているのか
3. クラウドにおけるセキュリティの考慮事項
4. コンテナ化におけるセキュリティ
5. 新機能や新サービス開発におけるセキュリティ
6. 今後の展望

# クラウド採用の背景

- **なぜクラウドを使うのか**
  - 弊社は元々オンプレが基本だった
  - スモールスタートな新規事業にはオンプレは不向き
  - 必要な分だけ、最小限のコストで使える良さ
  - マネージドサービスによる運用負荷の軽減（セキュリティ関連のサービス、機能も充実）

- **AWS（Amazon Web Services）の採用**
  - 社内・社外（ブログや書籍、記事など）にAWSのノウハウが多くあった
  - 機能が豊富で、各社での利用実績がある安心感

# 「家族アルバム みてね」におけるクラウド（AWS）活用



# 「家族アルバム みてね」におけるクラウド（AWS）活用

- **サーバーアプリケーション（Ruby on Rails）**
  - Amazon EC2、Amazon Aurora、Amazon ElastiCache
  - オーケストレーションにはOpsWorksを利用していたが現在はAmazon EKS

- **オブジェクトストレージ**
  - Amazon S3
  - 写真や動画の保存先として圧倒的な信頼性（99.999999999%の耐久性）
  - 豊富なアクセス制御、バージョン管理

# アジェンダ

1. 会社とサービスの紹介
2. どのようにクラウドを活用しているのか
3. **クラウドにおけるセキュリティの考慮事項**
4. コンテナ化におけるセキュリティ
5. 新機能や新サービス開発におけるセキュリティ
6. 今後の展望

# クラウドにおけるセキュリティの基本

- **責任共有モデル**
- AWSの例

  **“AWS がホストオペレーティングシステムと仮想化レイヤーから、サービスが運用されている施設の物理的なセキュリティに至るまでの要素を AWS が運用、管理、および制御する**ことから、お客様の運用上の負担を軽減するために役立ちます。

  **お客様には、ゲストオペレーティングシステム (更新とセキュリティパッチを含む)、その他の関連アプリケーションソフトウェア、および AWS が提供するセキュリティグループファイアウォールの設定に対する責任と管理を担っていただきます”**

  https://aws.amazon.com/jp/compliance/shared-responsibility-model/

  **AWSにとっては「クラウドのセキュリティ」**

  **ユーザーにとっては「クラウドにおけるセキュリティ」**

# クラウドにおけるセキュリティのプラクティス（例）

- **クラウドに用意されているセキュリティーソリューションの活用**
  - 監査ログ、脅威検出、ファイアウォール、設定変更履歴管理など用意されたものを積極利用する
- **ネットワークの分離**
  - パブリックとプライベートを分ける
- **不要なポートを開けない**
  - SSHのポートを開けないでもターミナル利用できるソリューションも
- **最小限のポリシー**
  - 不必要な権限は与えない
- **定期的なアップデート**
  - 脆弱性を放置しない、deprecatedなバージョンを使い続けない

# 手作業でのクラウド利用における問題

# Infrastructure as Code（IaC）

- **手作業でのクラウド利用における問題**
  - レビューなしでの設定は抜け漏れが起きやすい。セキュリティ上の問題にもなりやすい。
  - 手順書はあっという間に古くなる。更新されなくなった手順は使えない。
  - 人間はミスをしやすい。ただし、ペア作業である程度防げる。
- **インフラをコード化することの意味（IaC）**
  - 作業記録、手順がコードとして表現される
  - Gitなどのバージョン管理ツールによって履歴管理（誰が、いつ、何を）が自動的におこなわれる
  - コードレビューによって第三者による確認がおこなえる
  - 変数の利用、ロジックの実装、モジュール化などができる
  - 適用した内容を戻したり、繰り返すことが楽におこなえる
  - 自動化しやすい

# Infrastructure as Code（IaC）の課題

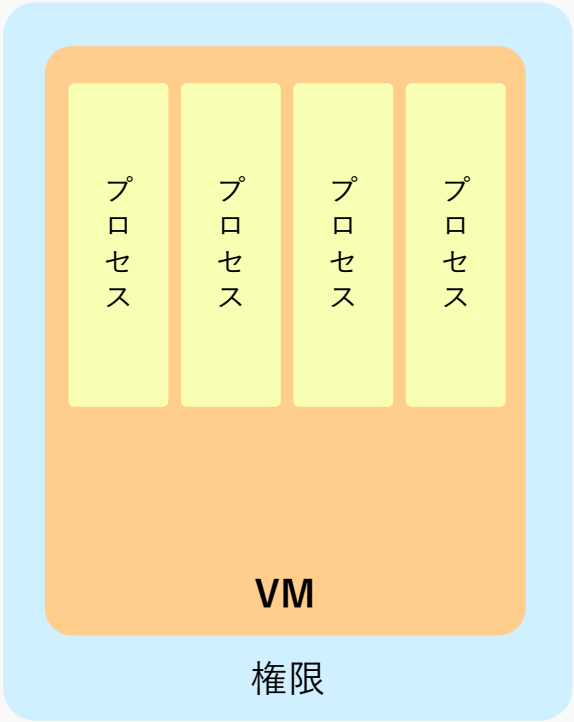
- **ツールの選定**
  - CloudFormationやTerraformなど様々なツールが存在する
  - 使っているクラウドや規模、組織などに合ったツールを選ぶ
- **ツールの実行環境**
  - Terraformのようにどこでも実行できるツールの場合、どこで実行するのが安全かを考える必要がある
  - 管理対象のリソース次第では強い権限が必要になる
  - ツール実行用の権限をどのように渡すのか、ポリシーの管理や認証方法をよく考える
  - コード化によるメリットに反して、セキュリティのリスクが増えてしまわないように設計をする

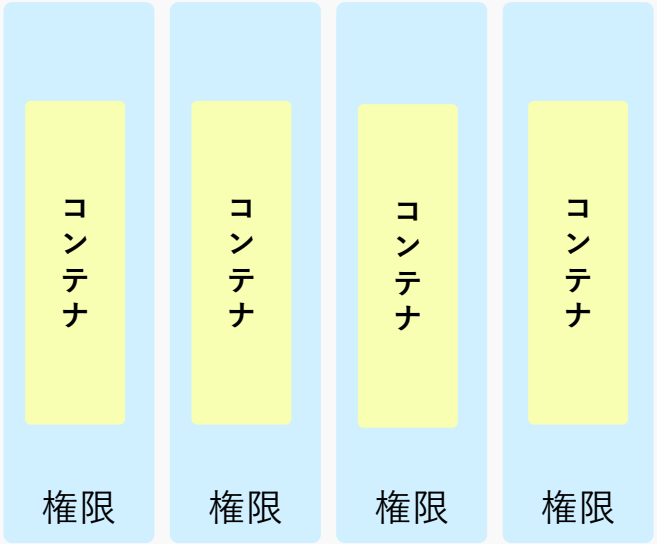
# アジェンダ

1. 会社とサービスの紹介
2. どのようにクラウドを活用しているのか
3. クラウドにおけるセキュリティの考慮事項
4. コンテナ化におけるセキュリティ
5. 新機能や新サービス開発におけるセキュリティ
6. 今後の展望

# VMの運用からコンテナの運用にシフト

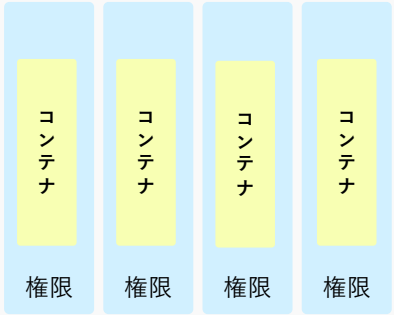
# VMとコンテナにおける権限の単位



# コンテナ化におけるセキュリティ

- **コンテナが持つ権限**
  - 従来はサーバー（インスタンス）単位で権限を付与
  - コンテナの機能単位で細かく権限設定可能に
    - Amazon ECSではタスク単位にIAMロールを付与
    - Amazon EKSではIRSA（IAM Roles for Service Accounts）を使う




- **コンテナの実行に必要な認証情報**
  - コンテナイメージ内に認証情報を持たせず、環境変数として外部から渡す
  - 秘匿情報は暗号化する
    - AWSではParameter StoreやSecret Managerがよく使われる

# コンテナ化におけるセキュリティ

- **ログ**
  - オーケストレーションツール（コントロールプレーン）側のログには、認証情報や認可情報などが記録されている。
  - ネットワークのログ。意図しないポートへのアクセスなど。
- **プロセス**
  - コンテナ内のプロセス起動にrootユーザーを使わない。

# コンテナ化におけるセキュリティ

- **コンテナイメージ**
  - 野良のイメージの危険性、Dockerfileが公開されているかどうか
  - クレデンシャル情報が含まれていないか
  - 脆弱性診断ツールの活用
- **コンテナレジストリ**
  - アクセス制御が正しくされているか（パブリック or プライベート）
- **コンテナオーケストレーションツール**
  - 権限が適切に設定されているか
- **コンテナランタイム、OS**
  - 脆弱性に対応できているか（**セキュリティアップデート**を適用し続けられる運用）

※アメリカ国立標準技術研究所（NIST）“Application Container Security Guide”（NIST SP 800-190）より

# アジェンダ

1. 会社とサービスの紹介
2. どのようにクラウドを活用しているのか
3. クラウドにおけるセキュリティの考慮事項
4. コンテナ化におけるセキュリティ
5. **新機能や新サービス開発におけるセキュリティ**
6. 今後の展望

# 新機能や新サービス開発におけるセキュリティ

**事例**

**「新機能Aは別チームがクラウドのアカウント新たに作って構築します」**

というケース。

この場合、新たなアカウントにおけるセキュリティについてどうやって対応するか。

**課題**

- チーム間でセキュリティに対する考え方が共有されていない
- 設定方法や設定内容は人それぞれ

# チームや企業間でのセキュリティガイドライン作り

- チーム間でセキュリティに対する考え方が共有されていない
  - **ガイドラインを作って各チーム、各社で共有し、理解する**
- 設定方法や設定内容は人それぞれ
  - **最低限おこなうべき設定や確認事項をガイドラインに盛り込む**

これらに対応するために

- **AWS Well-Architected Framework の活用**
- **ガイドラインを独自に作成**
  - クラウドセキュリティガイドライン
  - アプリケーションセキュリティガイドライン

# AWS Well-Architected Framework

# AWS Well-Architected Framework

“AWS Well-Architected は、クラウドアーキテクトがアプリケーションやワークロード向けに高い安全性、性能、障害耐性、効率性を備えたインフラストラクチャを構築する際に役立ちます。”
https://aws.amazon.com/jp/architecture/well-architected/ より

- **5つの柱**
  - **優れた運用効率**
  - **セキュリティ**
  - **信頼性**
  - **パフォーマンス効率**
  - **コストの最適化**

# AWS Well-Architected Framework におけるセキュリティ

- **アイデンティティとアクセスの管理**
  - 適切なユーザーが適切な条件で適切なリソースにアクセスできるようにする
- **検出**
  - 潜在的なセキュリティ設定の誤り、脅威、予期しない動作を特定する
- **インフラストラクチャの保護**
  - ネットワークの保護: ネットワーク設計を慎重に計画し管理する
  - コンピューティングの保護: 脆弱性管理、攻撃対象領域の縮小、安全なリモートアクセス、マネージドサービスの活用、ソフトウェアの整合性検証、自動化
- **データ保護**
  - 重要度と機密性にもとづいたデータの分類、保管中のデータの保護、伝送中のデータの保護
- **インシデントへの対応**
  - セキュリティインシデントが起こる前にツールとアクセス権を整備し、ゲームデー (実践訓練) を通じてインシデント対応を定期的に実施

https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/security-pillar/welcome.html

# 独自のガイドライン

# 独自のクラウドセキュリティガイドラインの例（AWSの場合）

- マネジメントコンソールにログインするユーザー全員MFAが有効となっているか。
- Amazon EC2などに不要な権限が付いていないか。
- アクセスキーを必要としている箇所があるか。
- 本番、ステージング、開発環境を相互にアクセスできる権限がないか（クロスアカウントを含む）
- パブリック公開されたS3バケットはあるか。ブロックパブリックアクセスは有効化されているか。
- AdministratorAccessポリシーを利用している箇所があるか。
- インターネット公開されたポートは80や443以外にあるか。
- DBのアクセス元はVPC内のアプリケーション以外に存在するか。
- DBへのログイン権限は、誰が、どこまでアクセスできるようになっているか。
- Amazon GuardDuty、AWS CloudTrail、AWS Configで脅威検出や監査ができるようになっているか。

# 独自のアプリケーションセキュリティガイドラインの例 ①

- ソースコードはどうやって管理されているか。レビュー体制はどうなっているか。
- ユーザーのセッショントークンの取り扱い
  - 有効期限
  - 保存場所はどこか
- 一般的なXSS、CSRFの対策はできているか
- APIのパラメータを直接変更して、他人のデータを見ることはできないか
- ソースコードやアプリケーション内部に秘匿情報を持っていないかどうか
- ユーザーのパスワード
  - 平文で保存していないか
  - 平文で送信（メールなど）していないか
  - パスワードの強度（文字数、文字種など）は適切か
  - パスワードリマインダの手法はどんなものか

# 独自のアプリケーションセキュリティガイドラインの例 ②

- クライアント・サーバー間の通信にTLSv1.2以降のプロトコルを使っているか。
- クライアントに保持されるデータ
  - ライフタイムはどのくらいか。
  - データの保存場所はどこか。
- ログの内容はどんなものか。ユーザーを特定できる形で残しているか。
- ログの保存期間はどのくらいか。検索可能か。
- ログ上の個人情報（氏名、メールアドレス、住所など）はマスク処理しているか。
- パスワード総当り攻撃を検知可能か。
- アプリ側に脆弱性が見つかったとき等、アップデートを強制できるようになっているか。
- 決済サービスを使っているか。
  - どのサービスで、どんな実装になっているか。

# アジェンダ

1. 会社とサービスの紹介
2. どのようにクラウドを活用しているのか
3. クラウドにおけるセキュリティの考慮事項
4. コンテナ化におけるセキュリティ
5. 新機能や新サービスにおけるセキュリティ
6. **今後の展望**

# 今後の展望

- ガイドラインをアップデートし続ける
  - 最初から完璧なものは作れない。フィードバックを取り入れ続けることが大事。
  - 作られたガイドラインはできるだけGitHubなどで公開し、多くの組織で役立てていきたい。
- IaCの改善
  - クラウドのリソースの多くはTerraform で管理されているが、まだまだ改善の余地がある。
- SSO（Single Sign-On）の導入
  - ユーザー管理を一元化したい。入退職処理の抜け漏れ防止。
- ログの集約、分析、通知の改善
  - 異常に気づいたときの原因調査の効率化
  - 対応が明確なアラート

# まとめ

- 「家族アルバム みてね」では初期からパブリッククラウドをフル活用してきた
- VMからコンテナへの移行にあわせて、コンテナならではのセキュリティ対策を実施
- インフラのコード化（Infrastructure as Code）はセキュリティ対策に効果的
  - ただし、実行環境のセキュリティ設計は非常に重要
- クラウドが推奨するセキュリティガイドラインの活用、独自のガイドラインの整備
- 新機能や新サービスにおいてガイドラインを適用することで、ステークホルダー間でセキュリティの共通認識を持つ
- これからも改善し続けることが大事

mixi, Inc.