FINATEXT HOLDINGS

技育CAMPアカデミア

事例を通して学ぶ
AWSクラウドアーキテクチャ入門

2024-07-25

株式会社Finatextホールディングス

松崎稔矢 @tm8619_pro

© 2024 Finatext Holdings Ltd.

自己紹介

© 2024 Finatext Holdings Ltd.

自己紹介

松崎稔矢
Toshiya Matsuzaki

1996-06-19生(28歳)
X(Twitter): @tm8619_pro

職種: バックエンドエンジニア
趣味: ボウリング/ゲーム/ダーツ/ポーカー/乗馬　など
その他:
・競技プログラミング経験あり　好きなアルゴリズムは乱択
・ISUCON2回出場　参加記をテックブログに書いてます
・ジャンル問わず何でもやりたい好奇心旺盛な性格

経歴
2019年1月インターン開始
2021年4月新卒入社
入社後、3つのプロダクト新規開発のリードを行う
2024年4月開発チームリーダーになる
2024 Japan AWS Jr.Champions 選出

© 2024 Finatext Holdings Ltd.

- AWSのベストプラクティスの簡単な紹介
- EC2だけで作った構成を紹介
- 構成を良くしていきながら、
  使用するAWSサービスを紹介する
- 要件に応じてどう変えていくのかを考える

© 2024 Finatext Holdings Ltd.

AWSのベストプラクティスの簡単な紹介

© 2024 Finatext Holdings Ltd.

AWS Well-Architected Framework

AWSの出している設計指針

日本語: https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/framework/welcome.html

英語: https://docs.aws.amazon.com/en_us/wellarchitected/latest/framework/welcome.html

信頼性が高く、安全で、効率的で、費用対効果が高く、持続可能なシステムを設計して運用するための、アーキテクチャに関するベストプラクティスを学ぶことができる。

若干AWSの公式ドキュメントは独特な言い回しが多いことがあり初見では難しいので、英語がある程度読める人は、英語のほうが読みやすいかも。

一般的な設計原則と、以下6つの柱で構成されている。

- 運用上の優秀性
- セキュリティ
- 信頼性
- パフォーマンス効率
- コスト最適化
- サステナビリティ

© 2024 Finatext Holdings Ltd.

一般的な設計原則

https://docs.aws.amazon.com/ja_jp/wellarchitected/latest/framework/general-design-principles.html

- キャパシティニーズの推測が不要
  - 最初に買ったサーバーがでかすぎてもったいなかった・小さすぎてまたすぐ買い足さないといけなかった…などを考えず、柔軟にスケーリングできるシステム
- 本稼働スケールでシステムをテストする
  - 簡単に環境をコピーできるようにする
- アーキテクチャの実験を念頭に置いた自動化
  - 変更や戻しが簡単にできるようにする
- 発展するアーキテクチャを検討する
  - サービスが成長していっても耐えられる設計にしておく
- データに基づいてアーキテクチャを駆動
- ゲームデーを利用して改善する

© 2024 Finatext Holdings Ltd.

運用上の優秀性

- ビジネス成果に基づいてチームを編成する
- オブザーバビリティを実装して実用的なインサイトを取得する
- 可能な限り安全に自動化する
- 小規模かつ可逆的な変更を頻繁に行う
- 運用手順を定期的に改善する
- 障害を想定する
- 運用上のあらゆるイベントやメトリクスから学ぶ
- マネージドサービスを使用する
    - サービスによって任せられる範囲はまちまちだが、インフラ面の保守をAWSに任せる
    - 物理的なものの管理は完全に不要

© 2024 Finatext Holdings Ltd.

セキュリティ

- 強力なアイデンティティ基盤を実装する
  - 最小限の人しか通さないようにする、最小権限のみ付与するという最小権限の原則
  - 弊社の別アカデミア回で、ここについて詳しく解説しています https://speakerdeck.com/kevinrobot34/introduction-aws-iam-3a810adc-5172-4460-9721-0041456eb2bf
- トレーサビリティの実現
  - 誰（人やシステム）がどんなアクションをしたかの履歴を残し、後から追えるようにする
- 全レイヤーでセキュリティを適用する
- セキュリティのベストプラクティスを自動化する
- 伝送中および保管中のデータを保護する
- データに人の手を入れない
- セキュリティイベントに備える

© 2024 Finatext Holdings Ltd.

信頼性

- 障害から自動的に復旧する
- 復旧手順をテストする
- 水平方向にスケールしてワークロード全体の可用性を高める
  - 一箇所に障害が起きたら全部死ぬような、単一障害点をなくそう、ということ。
  - 垂直は1つを大きくする、水平は数を増やすこと
- キャパシティーを推測することをやめる
- オートメーションで変更を管理する

© 2024 Finatext Holdings Ltd.

パフォーマンス効率

- 高度なテクノロジーを誰でも使えるようにする
  - AWSのマネージドサービスを使用することで、簡単に使える状態にする
- 数分でグローバルに展開する
- サーバーレスアーキテクチャを使用する
  - サーバーの運用負荷を下げることで、新規開発にリソースを割くことができる
- 実験の頻度を高める
- メカニカルシンパシーを検討する

© 2024 Finatext Holdings Ltd.

サステナビリティ

- 作業の影響を理解する
- 持続可能性の目標を設定する
- 使用率を最大化する
- より効率的なハードウェアやソフトウェアの新製品を予測して採用する
- マネージドサービスを使用する
- クラウドワークロードのダウンストリームの影響を軽減する

簡単に言うと、AWSが大規模に効率的なオペレーションを考えてくれているので任せてくれると結果効率的な事が多いのと、コストを削減する事（効率的にリソースを使用すること）がそのままサステナビリティに繋がる

FINATEXT HOLDINGS

© 2024 Finatext Holdings Ltd.

ざっくり設計で意識することまとめ

- 最小権限の原則
- トレーサビリティ（追跡可能性）を高める
- スケーラビリティ（拡張性）を高める
- オブザーバビリティ（可観測性）を高める
- アベイラビリティ（可用性）を高める
- デュラビリティ（耐久性）を高める
- 単一障害点を無くす
- 運用負荷を減らす
- マネージドサービスを使う

© 2024 Finatext Holdings Ltd.

EC2だけで作った構成を紹介

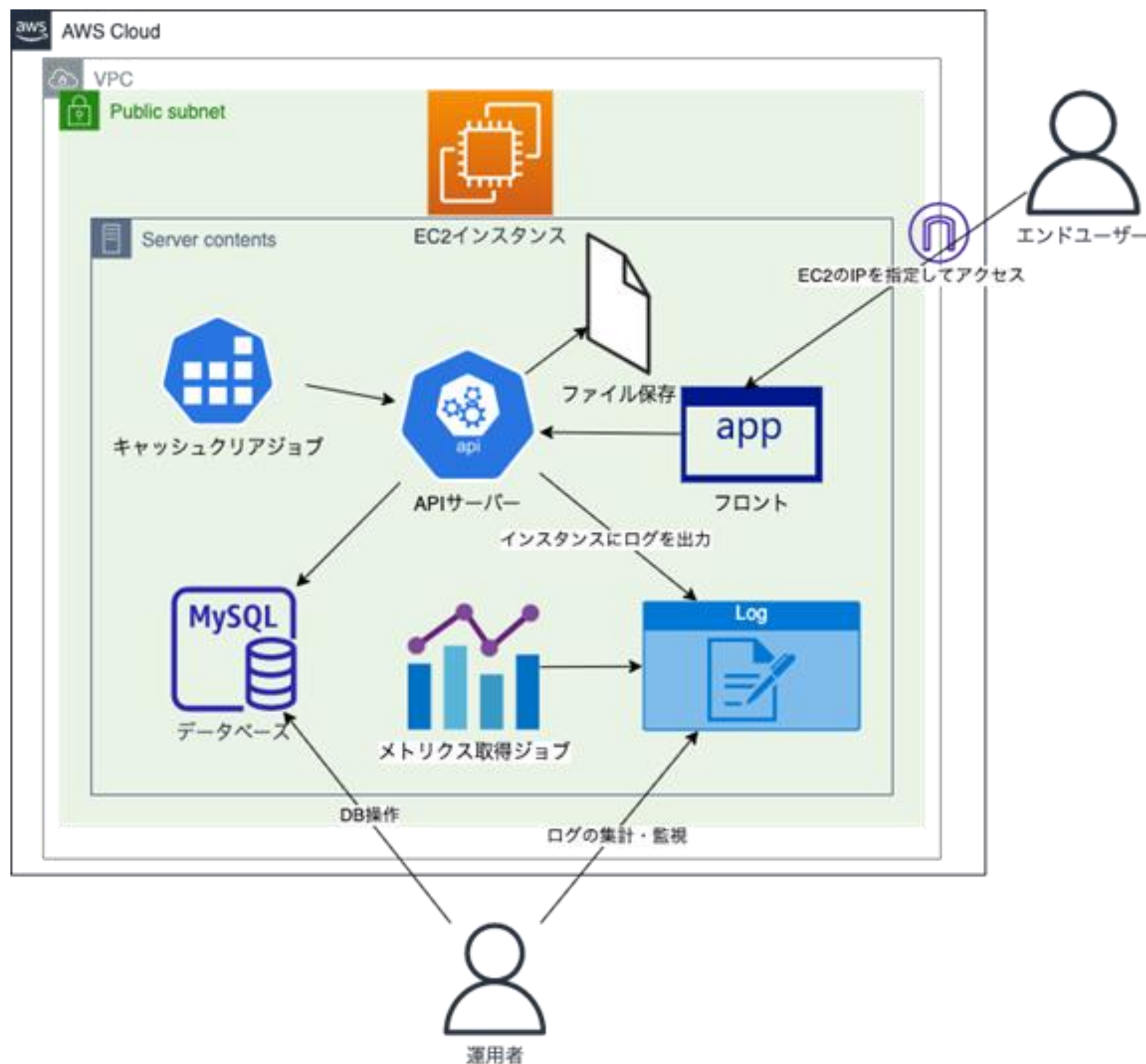
© 2024 Finatext Holdings Ltd.

一般的なwebアプリケーションを作りたい　要件

- インターネットに一般公開するアプリフロント・サーバーがある
- メトリクス（メモリ使用量など）を定期的に取得し、監視を行う必要がある
- アプリケーションにはファイルアップロード機能があり、個人情報を登録する可能性がある
- 永続化が必要なデータがある
- 永続化不要だが、高速化のためキャッシュ実装が必要
- 運用者がDB操作やアプリケーションログ、メトリクスの監視を行う
- ローカルでほぼシステムを構築出来たが、これをAWSにデプロイしたい

© 2024 Finatext Holdings Ltd.

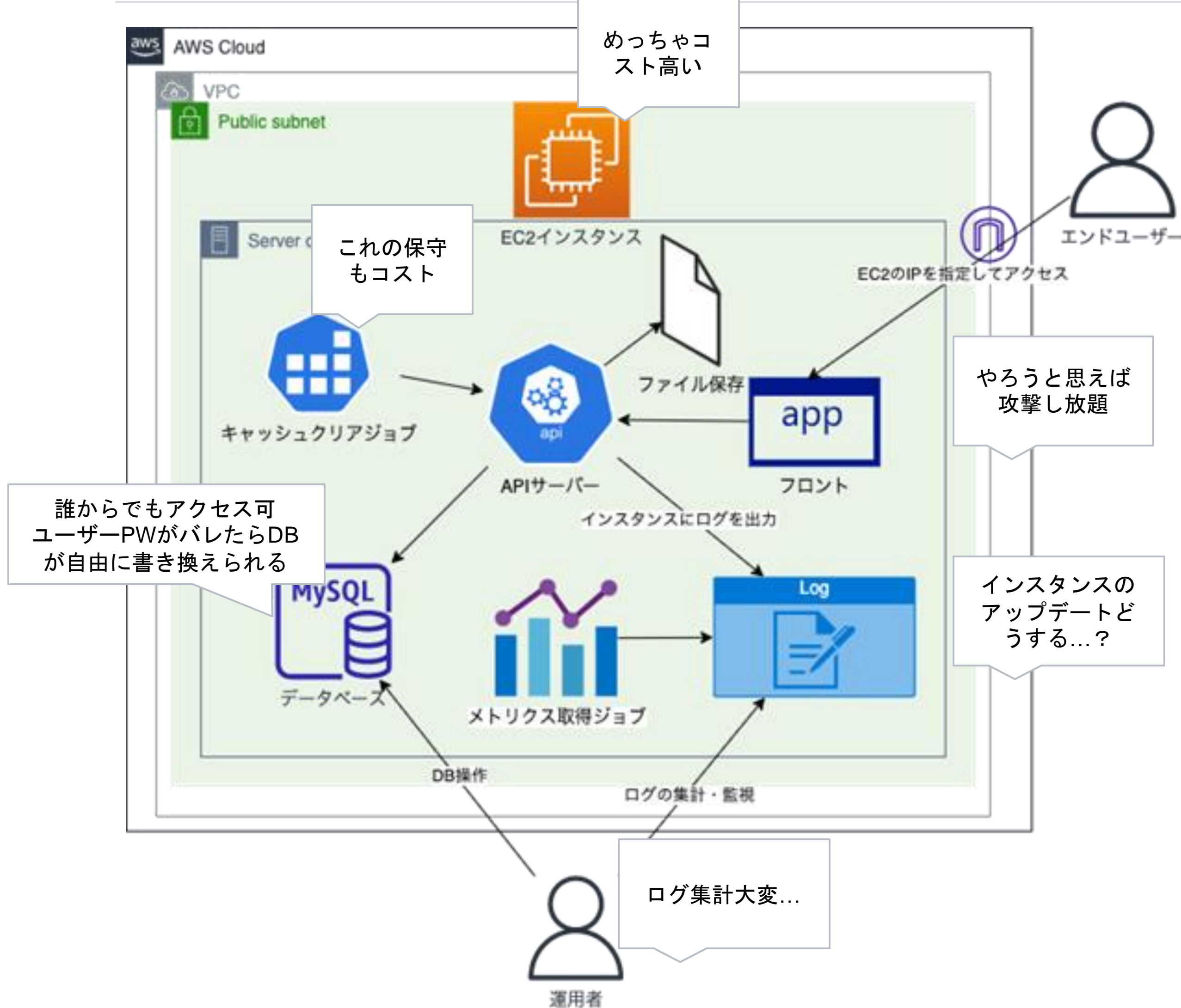
EC2だけで作った構成を紹介



- ローカルで作った環境を再現するため、1つのEC2インスタンスに全てを乗せる構成にした。
- 過去、一瞬だけアクセス過多で落ちた経験から、大きなEC2インスタンスを立てている。
- セキュリティグループは、全てのポートでインバウンド・アウトバウンドも0.0.0.0/0で許可した。
- アプリのインメモリキャッシュの実装があり、1時間を過ぎたデータはcronで起動されるキャッシュクリアジョブ（サーバーのAPIを一つ叩くのみ）で削除される。
- 運用者は、アプリのログとメトリクス取得ジョブが作成したログを集計し、監視する。
- DBは、ローカルからMySQLに作成したユーザーPWを使用しログインする。
- サーバーがファイルをインスタンス内に保存している。
- サーバーはDockerで起動している。

© 2024 Finatext Holdings Ltd.

EC2だけで作った構成を紹介



問題点

- セキュリティ上のリスクが大きい
- 運用負荷が高い
- インスタンスを大きくしたり小さくしたりすることが難しい
- DBだけ性能上げたい要望に答えられない
- インスタンスが仮に壊れた時に全データが失われるリスクがある

やりたいこと

- EC2が単一障害点になるので改善したい
- 最小権限の原則を適用して改善したい
- 運用負荷を減らしたい
- 拡張性・可観測性・追跡可能性を高めたい

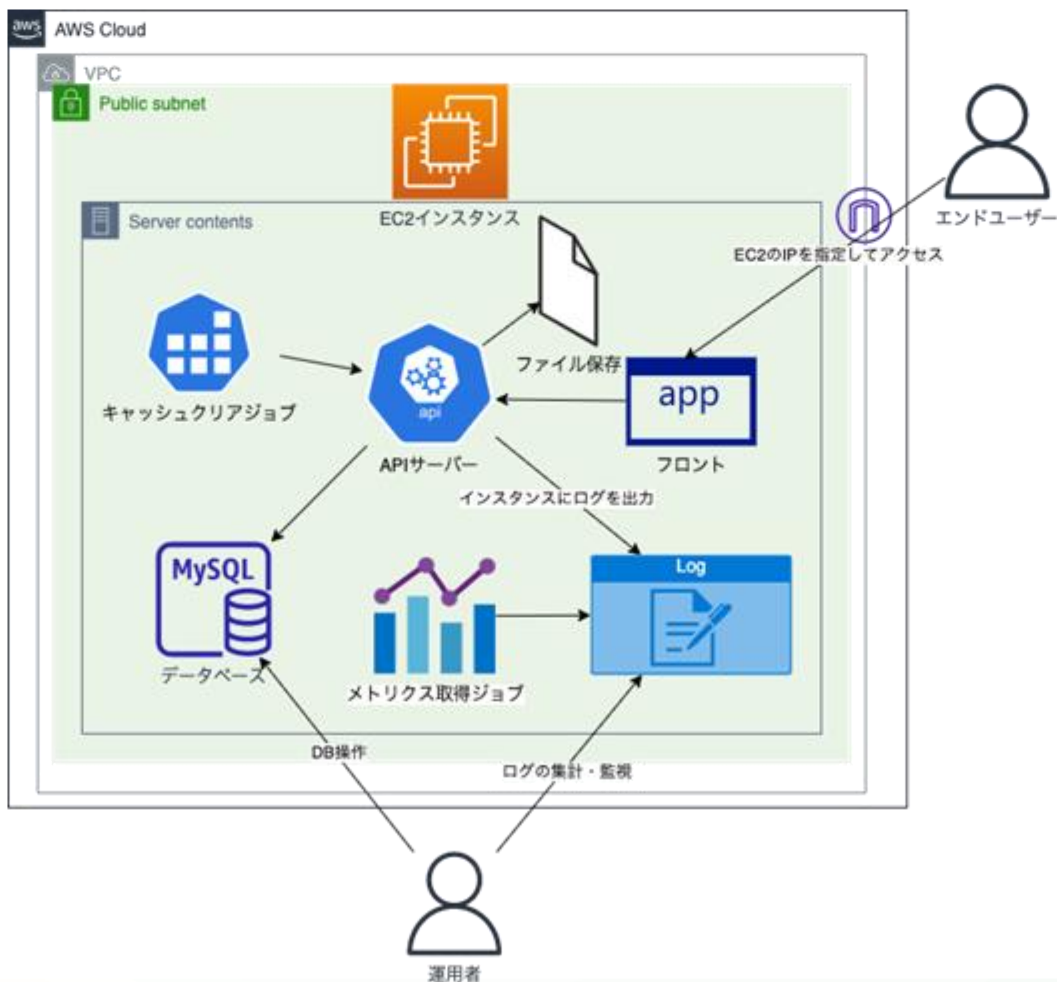
© 2024 Finatext Holdings Ltd.

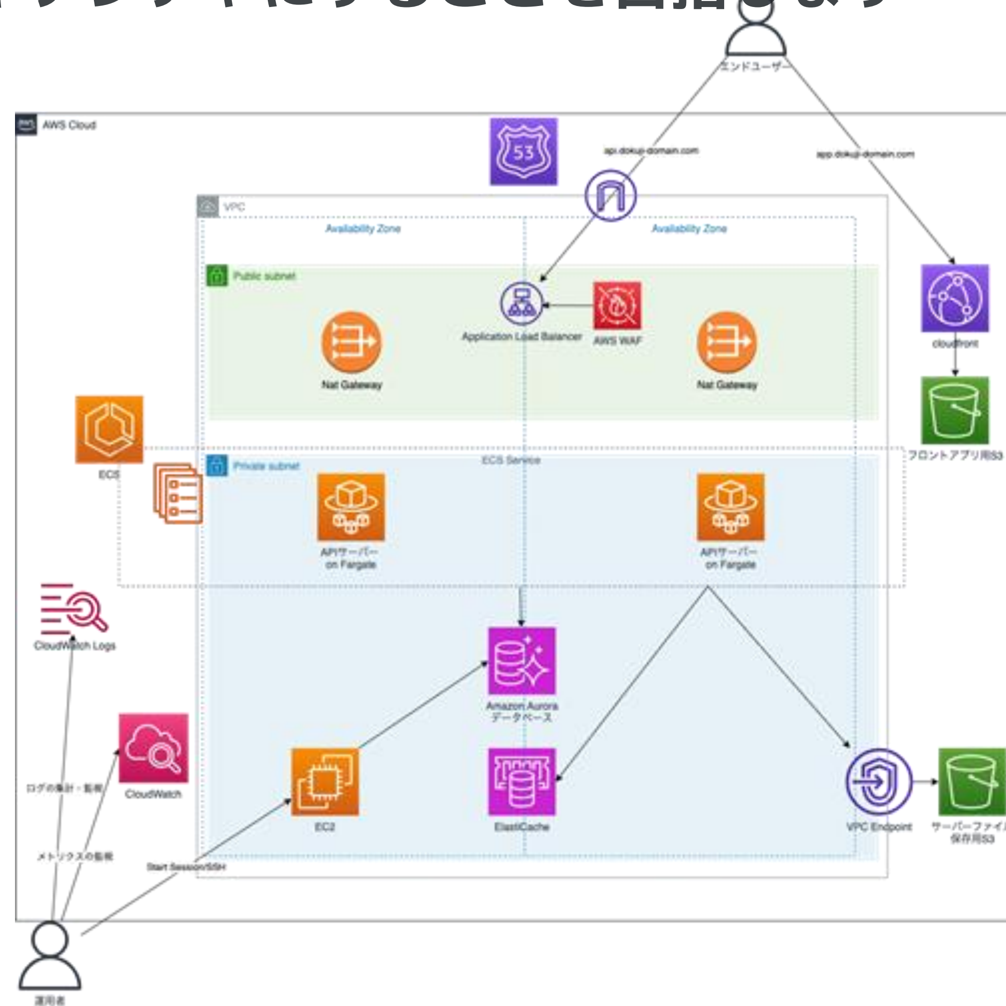
構成を良くしていきながら、使用するAWSサービスを紹介する

アーキテクチャを改善していく

© 2024 Finatext Holdings Ltd.

最終的には左のアーキテクチャを右のアーキテクチャにすることを目指します





© 2024 Finatext Holdings Ltd.

Amazon Elastic Container Serviceを使用して、APIサーバーの管理を簡潔に

AWS Elastic Container Registry (ECR)

AWSが用意したDocker Registry

AWS Elastic Container Service(ECS)

コンテナ化されたアプリケーションを簡単にデプロイ、管理、スケーリングするためのフルマネージドサービス

つまり、いい感じにコンテナを管理してくれるサービス

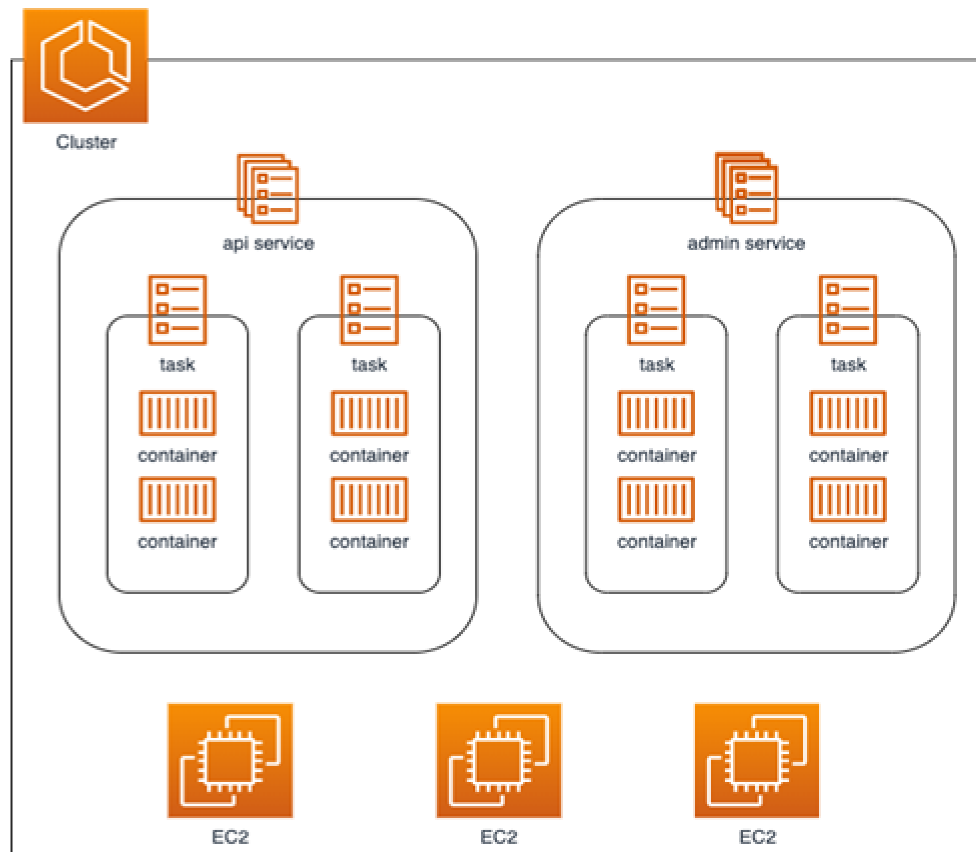
できること

- コンテナ管理の自動化
- オートスケーリング
- ECR(Elastic Container Registry)を利用し、イメージのデプロイを簡潔に
- 柔軟なデプロイ
  - ダウンタイムなしで可能なローリングアップデートなど

今回は、拡張性、運用負荷軽減、マネージドサービス使用の観点から、APIサーバーの管理をAWSに寄せるために使用します。

© 2024 Finatext Holdings Ltd.

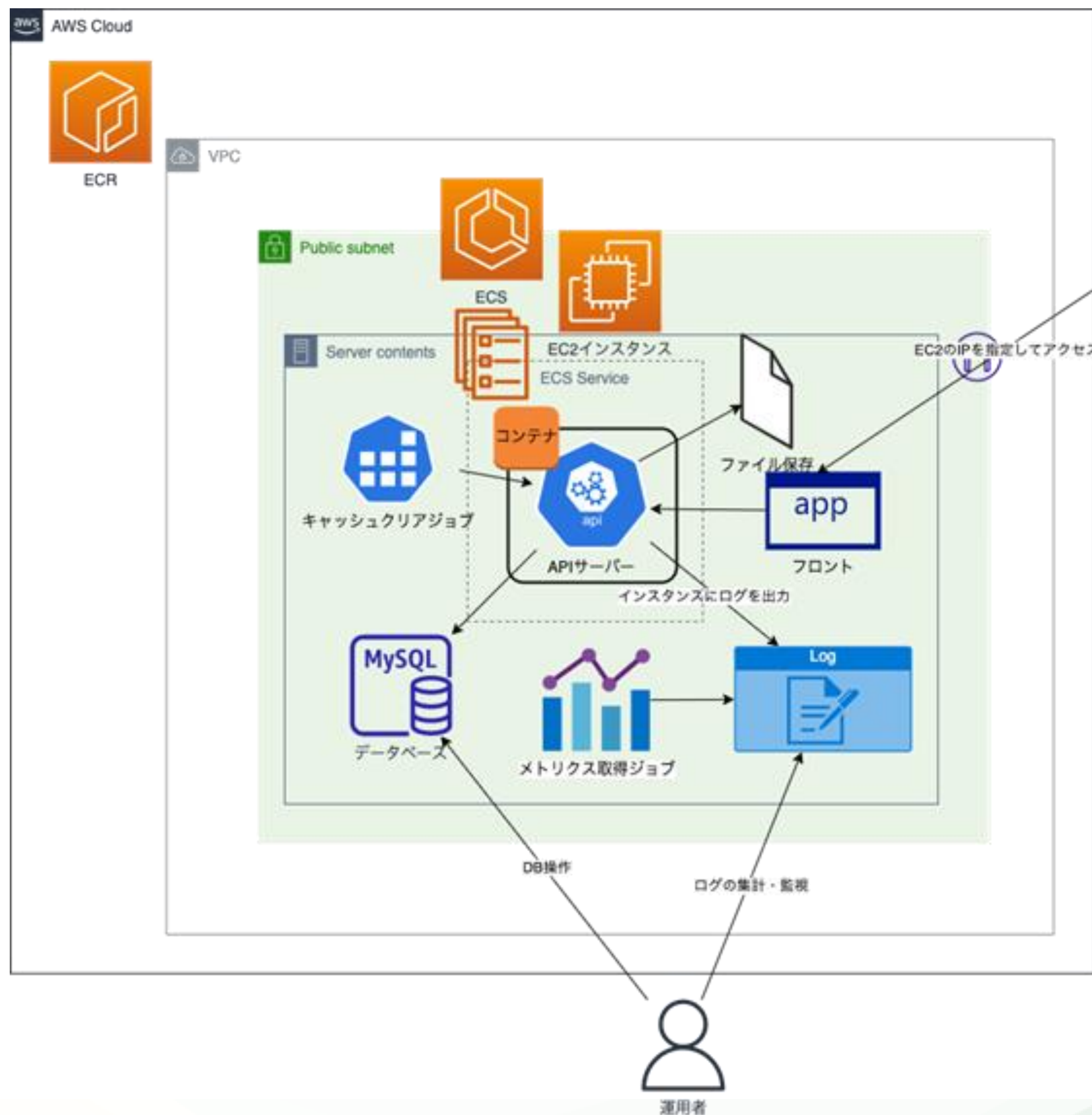
Amazon Elastic Container Serviceの簡単な概念説明



- タスク定義
  - コンテナの起動設定を管理する
- タスク
  - コンテナ群のこと
  - タスク定義の設定に基づき、コンテナを立てる
- サービス
  - タスクを管理するもの　数の増減や入れ替えなど
- クラスタ
  - サービスやタスクを実行する基盤・集まり

クラスタにはEC2が紐づいていて、タスクがEC2の場所を確保する。確保した場所にコンテナを立てる。

© 2024 Finatext Holdings Ltd.



やったこと

- APIサーバーをECSタスク経由で起動するようにした
- Docker RegistryとしてECRを使用するようにした

嬉しいこと

- APIサーバーのアップデートが楽になった
  - EC2内での作業が不要
- どのバージョンのAPIなのかが、タスク定義やECR Image経由で分かるようになった
- 運用負荷軽減・拡張性の向上

まだ出来ないこと

- ローリングアップデート
- ≒ APIサーバーを複数台立てること

© 2024 Finatext Holdings Ltd.

Amazon Elastic Load Balancingを使用して、スケーラブルに

Elastic Load Balancing(ELB)

ロードバランシングサービスで、サーバーの負荷の分散を行う。3種類ある。

- Application Load Balancer（ALB）
- Network Load Balancer（NLB）
- Classic Load Balancer（CLB）

できること

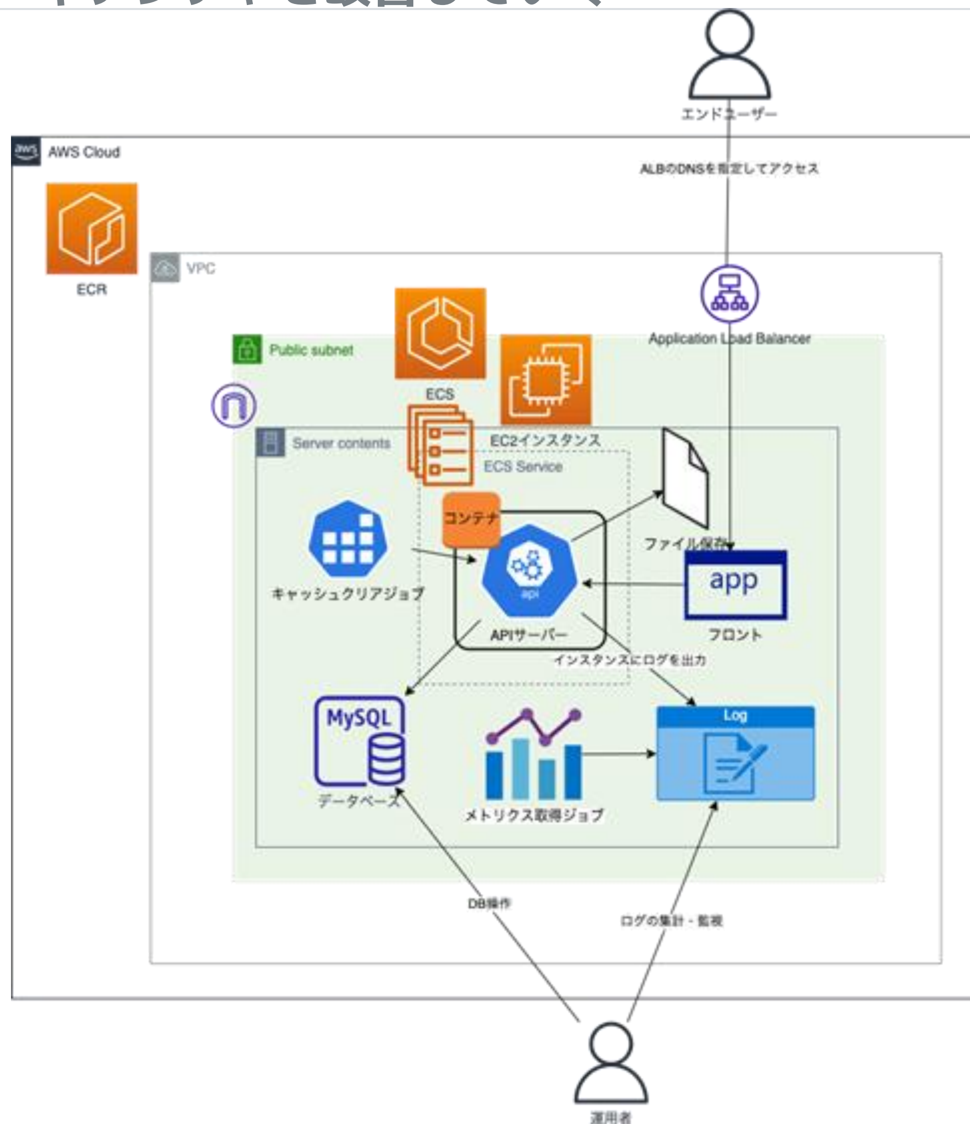
- パスやホストでのルーティング
- セキュリティ強化
- トラフィックを複数のEC2・コンテナに分散する
  - ECSサービスに紐づける事で、動的ポートマッピングを利用可能
- ヘルスチェック

今回は、動的ポートマッピングというALBの機能を使いたいことと、アプリへはALBを通した後にEC2にアクセスするようにします。

拡張性、単一障害点の回避に繋がります。

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やったこと

- フロントが参照するAPIを、ローカルではなくALB経由にするようにした。
- APIサーバーの前にALBを置いた

嬉しいこと

- APIサーバーのコンテナの数を自由に増やせるようになった
- ローリングアップデートにより、ダウンタイムなしにAPIサーバーを更新できるようになった

やりたいこと

- ログ周り、集計も監視もたくさんコマンド打たないといけないし簡単にしたいな…

© 2024 Finatext Holdings Ltd.

Amazon CloudWatchで監視をAWSに任せよう！

AWS CloudWatch

AWSのリソースとアプリケーションの監視および管理を行うためのサービス

できること

- メトリクスの収集とモニタリング
- ログの収集と分析
- アラームの設定
- ダッシュボードの作成
- イベントの監視と自動化

今回は、可観測性の向上、運用負荷低減の観点で、ログとメトリクス収集を置き換え、更にダッシュボードを作る事にします。

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく

AWS CloudWatch

CloudWatch > アラーム

アラーム (7)

Auto Scaling アラームを非表示

名前	状態	最終状態の更新 (ローカル)	条件
	アラーム状態	2024-07-24 15:52:24	15 分内の15データポイントの ConsumedReadCapacityUnits < 30
	OK	2024-07-24 15:37:21	2 分内の2データポイントの ConsumedReadCapacityUnits > 42
	OK	2024-06-13 14:24:41	5 分内の1データポイントの AccountManagerInvalidResourcePosted > 0
	OK	2024-03-07 14:21:54	1 日内の1データポイントの SMSMonthToDateSpentUSD > 40
	OK	2024-03-07 14:21:31	1 日内の1データポイントの SMSMonthToDateSpentUSD >= 40
	OK	2022-08-02 11:26:17	15 分内の3データポイントの ProvisionedReadCapacityUnits > 1
	OK	2022-08-02 11:21:16	15 分内の3データポイントの ProvisionedReadCapacityUnits < 1

- アラームを設定したり、ダッシュボードを設定したりすることで、監視を集約できる。
- リソースの使用状況や、ALBのStatusCodeの件数など、様々なメトリクスを取得可能。

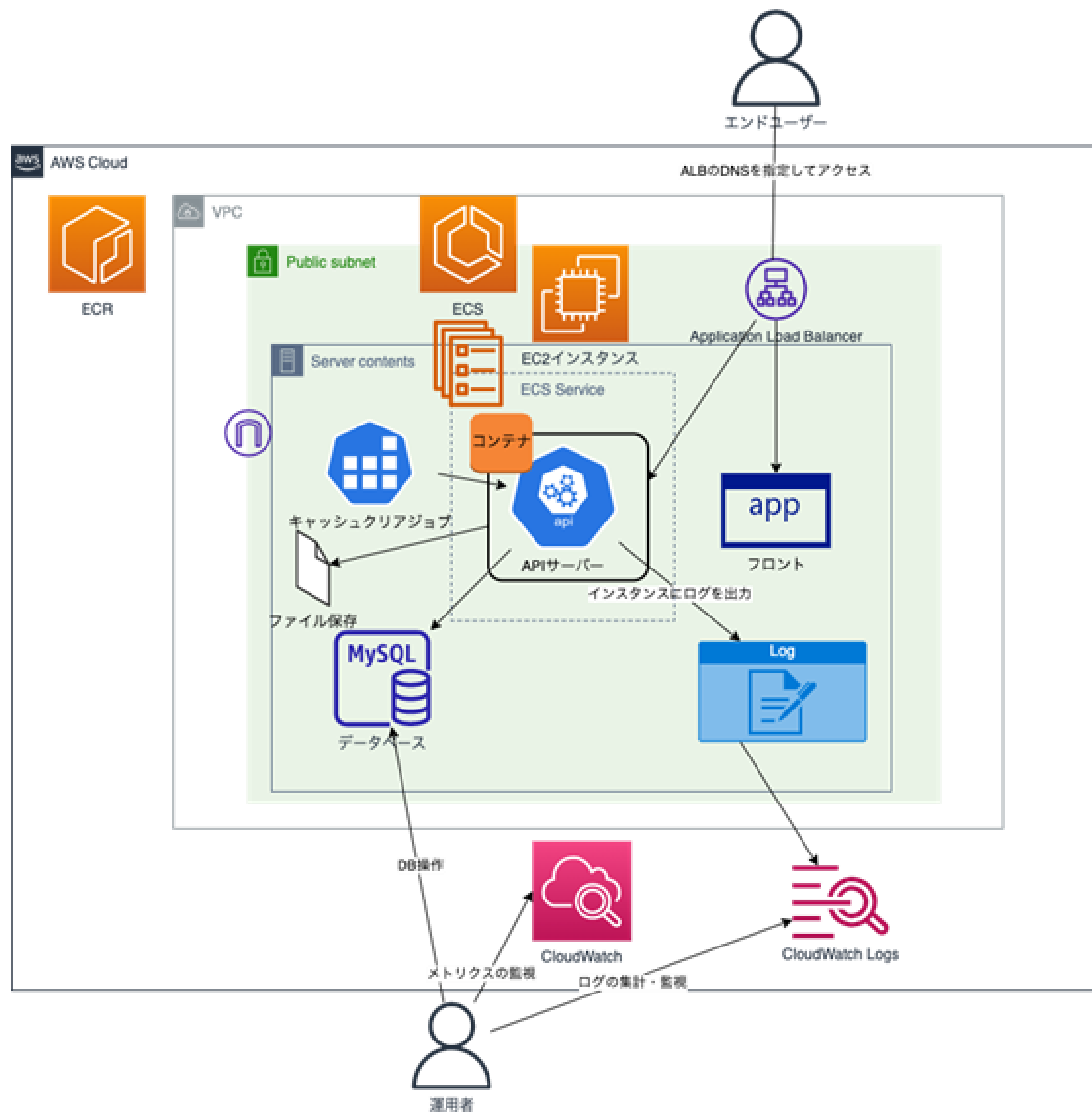
© 2024 Finatext Holdings Ltd.

Amazon CloudWatchLogs

AWSにログの管理を集約！JSONにすると検索性も上がります。
ALBのアクセスログも設定で集められます。

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やったこと

- ログをCloudWatchLogsに置き換えた
- メトリクス監視をCloudWatchに置き換えた
- CloudWatch Logsを有効活用できるよう、ログ出力をjson形式にフォーマットを整えた

嬉しいこと

- ログの調査が直感的になった
- 監視が集約され、今まで見てなかったものも一緒に見れるようになった

やりたいこと

- 仮にEC2インスタンスが壊れたらデータが無くなってサービス終了するので改善したいな…

© 2024 Finatext Holdings Ltd.

Amazon Simple Storage Serviceを利用して圧倒的な耐久性を得る

AWS Simple Storage Service(S3)

オブジェクトストレージサービス

DropBox/GoogleDriveなどのAWS版とイメージすると分かりやすい
※厳密には仕組みが違うもの

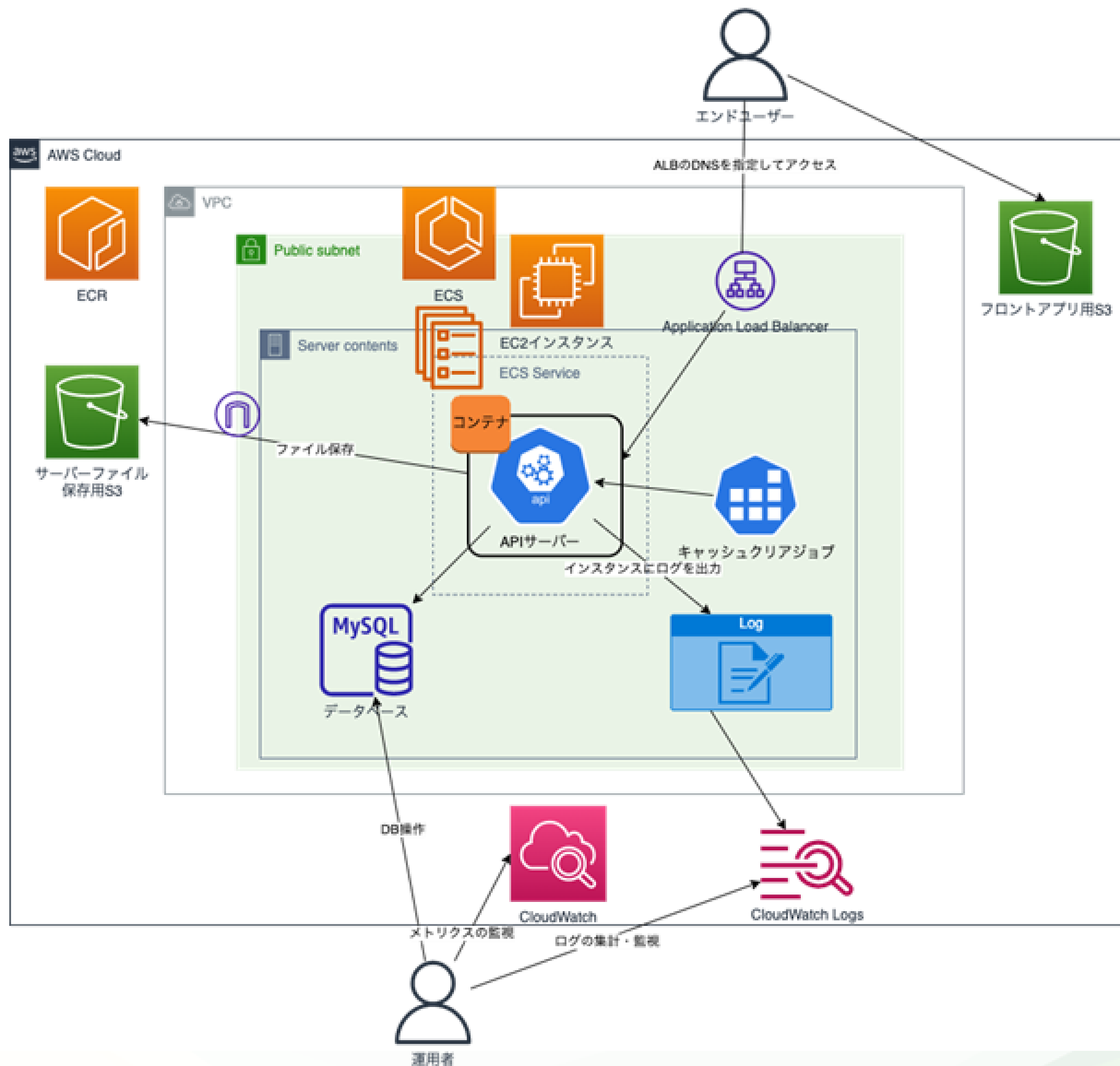
できること

- 他のAWSサービスとの連携
  - イベントをトリガーにして後続処理を起動できる
- 自動でのバックアップ
- バージョニング
  - 同じファイル名での更新履歴まで取得可能
- 静的ファイルの配信
  - フロントのデプロイ先にできる
- セキュリティ設定

今回は、障害耐久性や運用負荷低減のため、サーバーのファイル保存先を置き換え、更にフロントのデプロイ先も置き換えることにする

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やったこと

- フロントをS3に置き、パブリックアクセスを許可することで置き換えた
- サーバー用のS3バケットを作成して使うようにした

嬉しいこと

- サーバーのファイルが、EC2に入らずとも確認出来るようになり、耐久性も上がった
- フロントのデプロイが簡単になった

やりたいこと

- MySQLのデータも守りたい

© 2024 Finatext Holdings Ltd.

Amazon Relational Database Serviceを使用して、高可用性を得よう！

Amazon Relational Database Service(RDS)/ Amazon Aurora

今回はRDSの中のAurora MySQL（AWSが再設計したもので、MySQL互換で使えつつ独自機能もある）を使用します。

ざっくり、高機能になったMySQLと考えてokです。

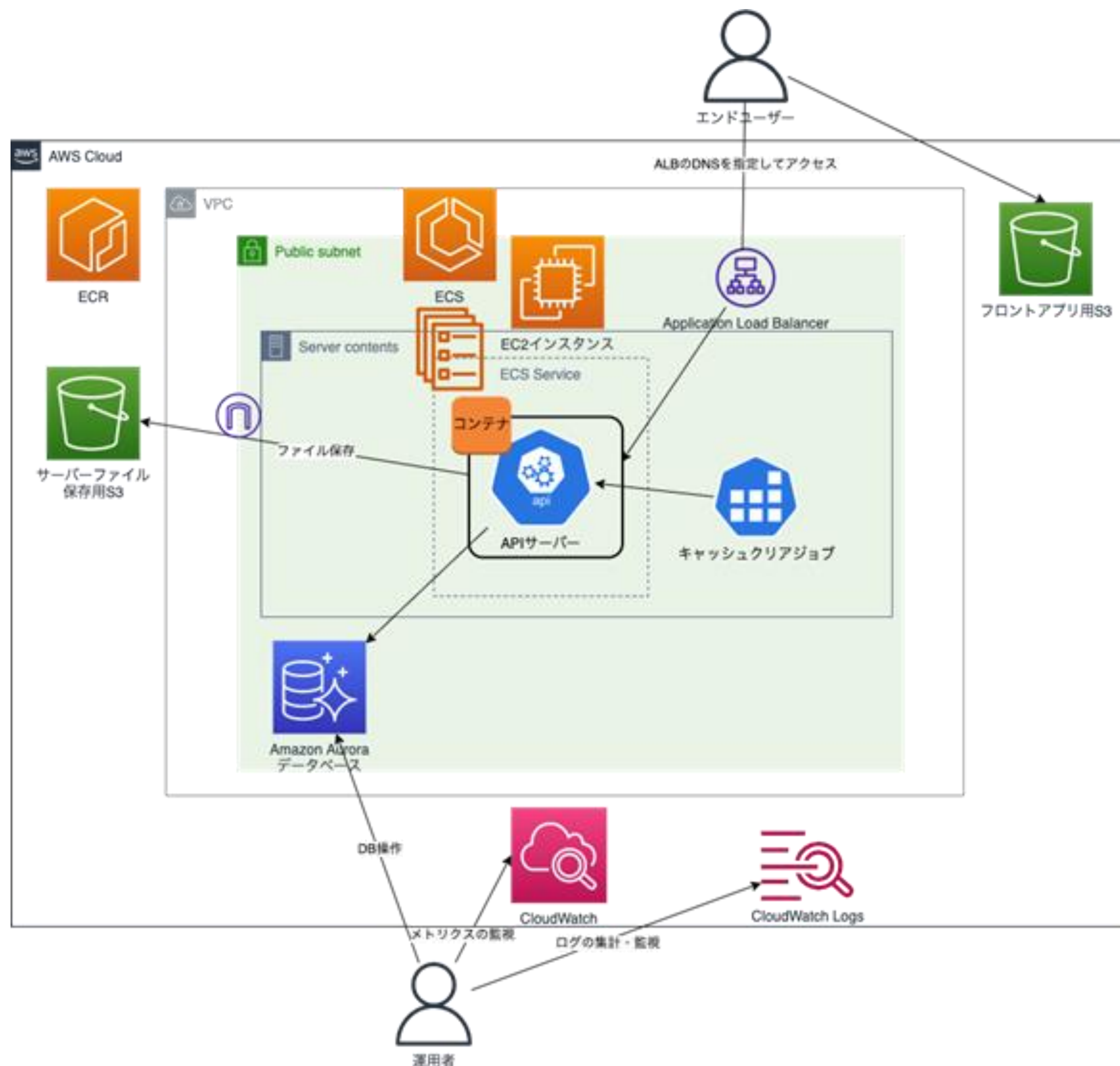
できること

- 容易なレプリケーション
- 高可用性
- 自動バックアップ
  - ある時点の状態に戻すことも可能
- フェイルオーバー
  - 主インスタンスに障害が起きたとき、自動で別のインスタンスがその役割を担う
- ストレージ自動拡張
- クエリのパフォーマンス分析

今回は、データのバックアップ機能の使用や、今後の拡張性を考えEC2インスタンスのDBをAuroraに置き換えます。

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やったこと

- EC2で立っていたMySQLを、Amazon Auroraに置き換えた
- データ移行も行った

嬉しいこと

- DBの可用性が上がった
- EC2インスタンスが壊れてもデータが守られる
- スケーラビリティが向上した
  - 簡単にレプリカを追加出来る

やりたいこと

- EC2インスタンスにAPIコンテナだけにしたい
- キャッシュ周りもAWSに載せちゃいたい

© 2024 Finatext Holdings Ltd.

Amazon ElastiCacheを使用して、管理の手間を省き高可用性を得よう！

できること

- 可用性やレプリケーション周りはRDSと同じ！
- Redis/MemcachedをAWSのマネージドで使える
    - サーバー等を立てたり、スケーリングに悩む必要がない

Amazon ElastiCache

Redis/Memcachedを使用可能

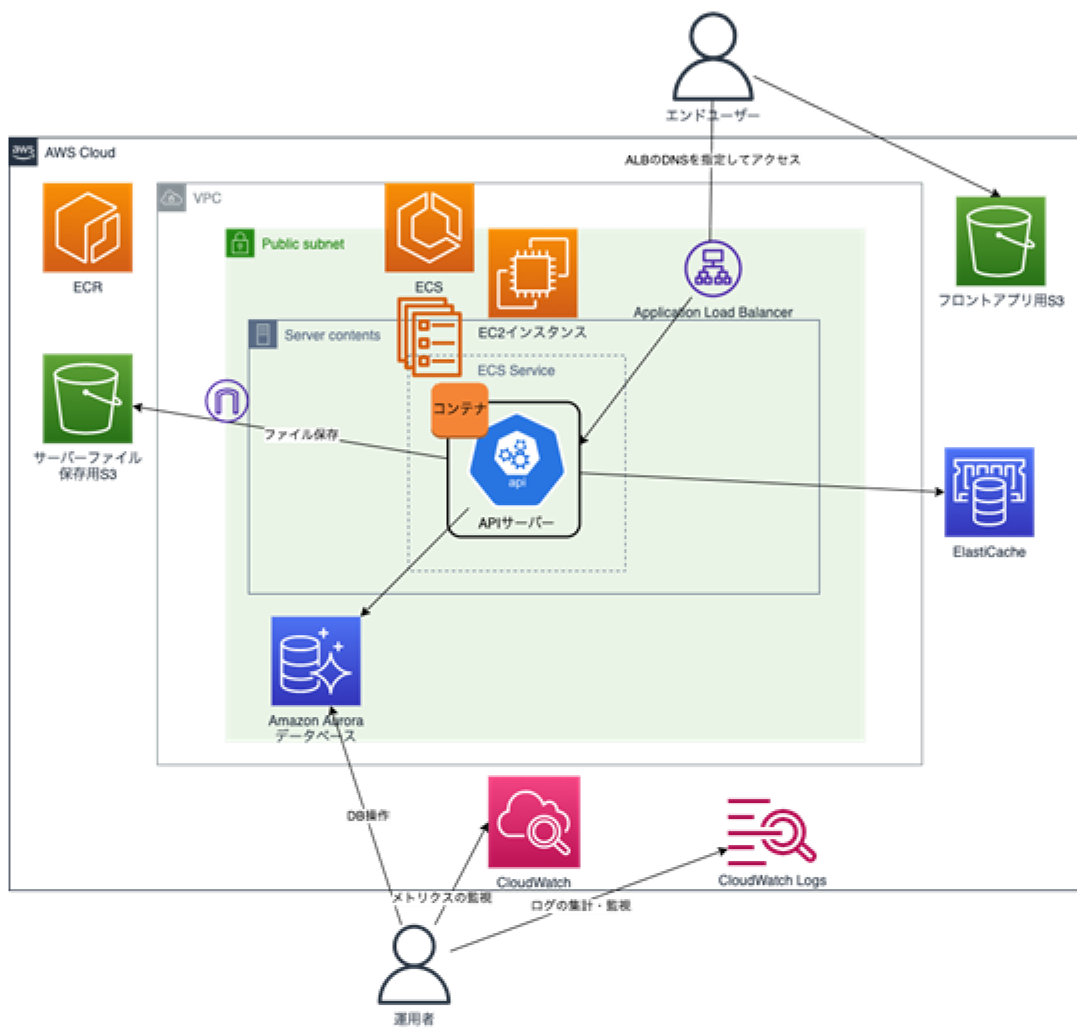
パフォーマンスが高く、サイズを動的に変更可能でスケーラビリティが高い。

拡張性と、マネージドサービス使用の観点で使用します

今回は実際には、言語のキャッシュライブラリ等を使用する事でEC2のインメモリキャッシュを実装も可能ではあるが、AWSのサービスに任せる方針で置き換える。

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やったこと

- キャッシュ管理をElastiCacheに置き換え、TTL(Time To Live)を1時間に設定した
- キャッシュクリアバッチが不要になったので消した

嬉しいこと

- キャッシュクリアバッチを消せた
- EC2インスタンスの内部の構成が非常にシンプルになった

やりたいこと

- EC2のメンテナンスも面倒なので管理をやめたい

© 2024 Finatext Holdings Ltd.

Amazon Fargateを使用して、EC2の管理からも開放されよう！

Amazon Fargate

コンテナ向けサーバーレスコンピューティングサービス

できること

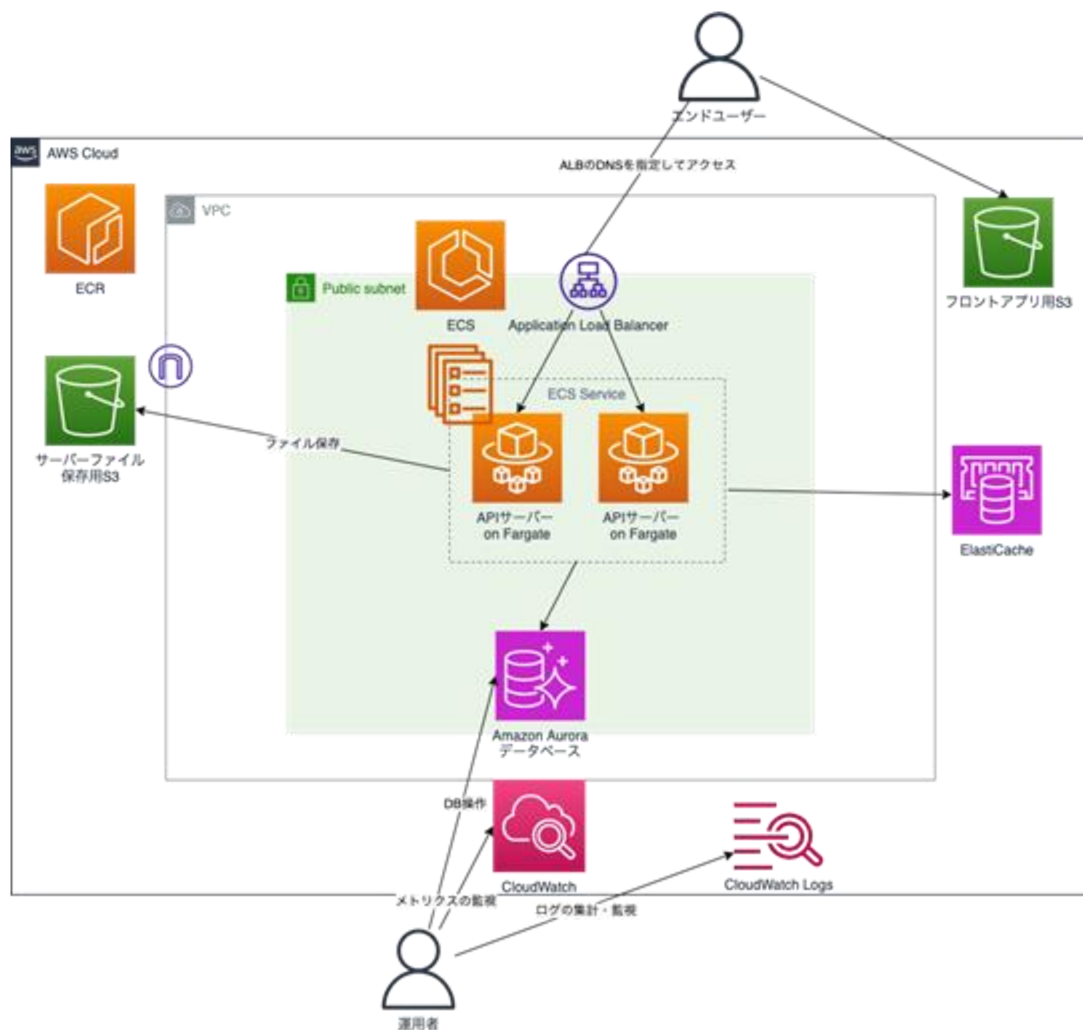
- インフラなしに、コンテナを起動できる
- 必要になった時に必要な分のリソースだけを使用できる
- 今まではEC2の中に収まる量しかコンテナを配置できなかった（増やしたければEC2を追加で立てたり大きくする必要があった）が、それが実質無制限になる

運用負荷軽減、拡張性向上、マネージドサービスを利用する観点で使用します。

構成の改善により、EC2はAPIサーバーのコンテナしか立っていない状態になったので、コンピューティングのインフラもAWSに任せることにします。

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やったこと

- コンテナはFatgate起動にした

嬉しいこと

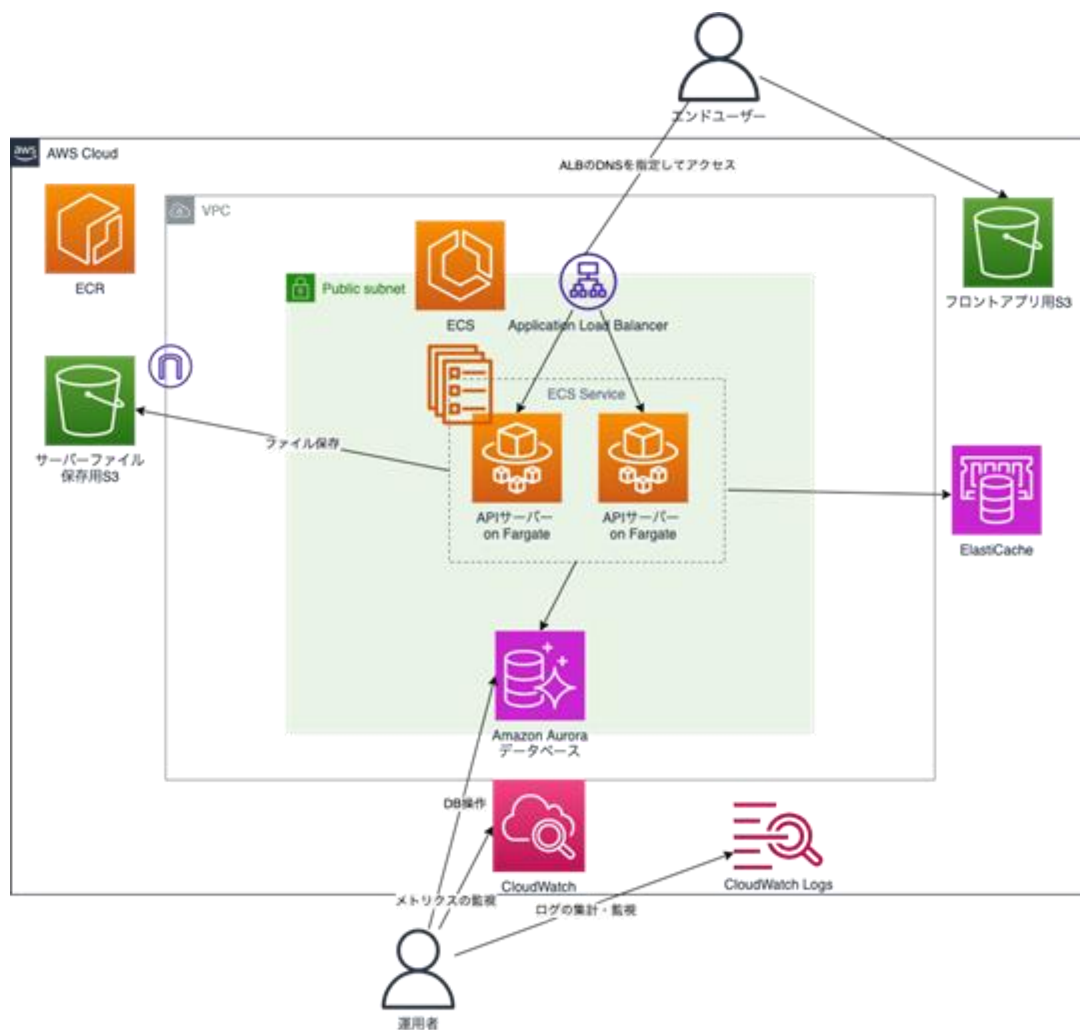
- EC2の管理が不要になった
    - 急にたくさん増やしたいとき、EC2の上限を気にしなくて良い
    - EC2のインスタンスが不調のようなことを気にしなくて良い

やりたいこと

- インフラが動かしやすい構成になったので、ネットワーク周りを整備することでセキュリティ強化

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やりたいことは、アクセス許可を最小限にする

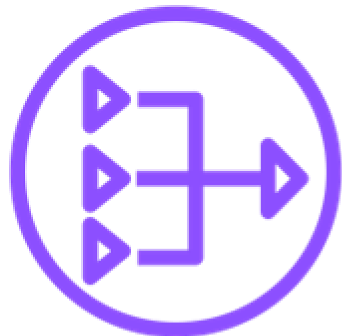
- APIサーバーはALBから受け取る
- DBはAPIサーバー、運用者からのみアクセス
- ElastiCacheはAPIサーバーからのみ
- サーバーファイル保存用S3はインターネットを通したくない

このあたりは外からアクセス出来ない場所に置くようにする（=プライベートサブネット内に置く）

しかし、逆にプライベートサブネット内からもインターネットにアクセス出来なくなる

© 2024 Finatext Holdings Ltd.

NAT GatewayとVPC Endpoint



NAT Gateway

ネットワークアドレス変換ができるAWSサービス

簡単に言うと、プライベートサブネット内のリソースがインターネットにアクセスできるようにしつつ、インターネットからのアクセスは防ぐ事ができる

これを利用することで、プライベートサブネットからインターネットアクセスが可能になる

VPC Endpoint

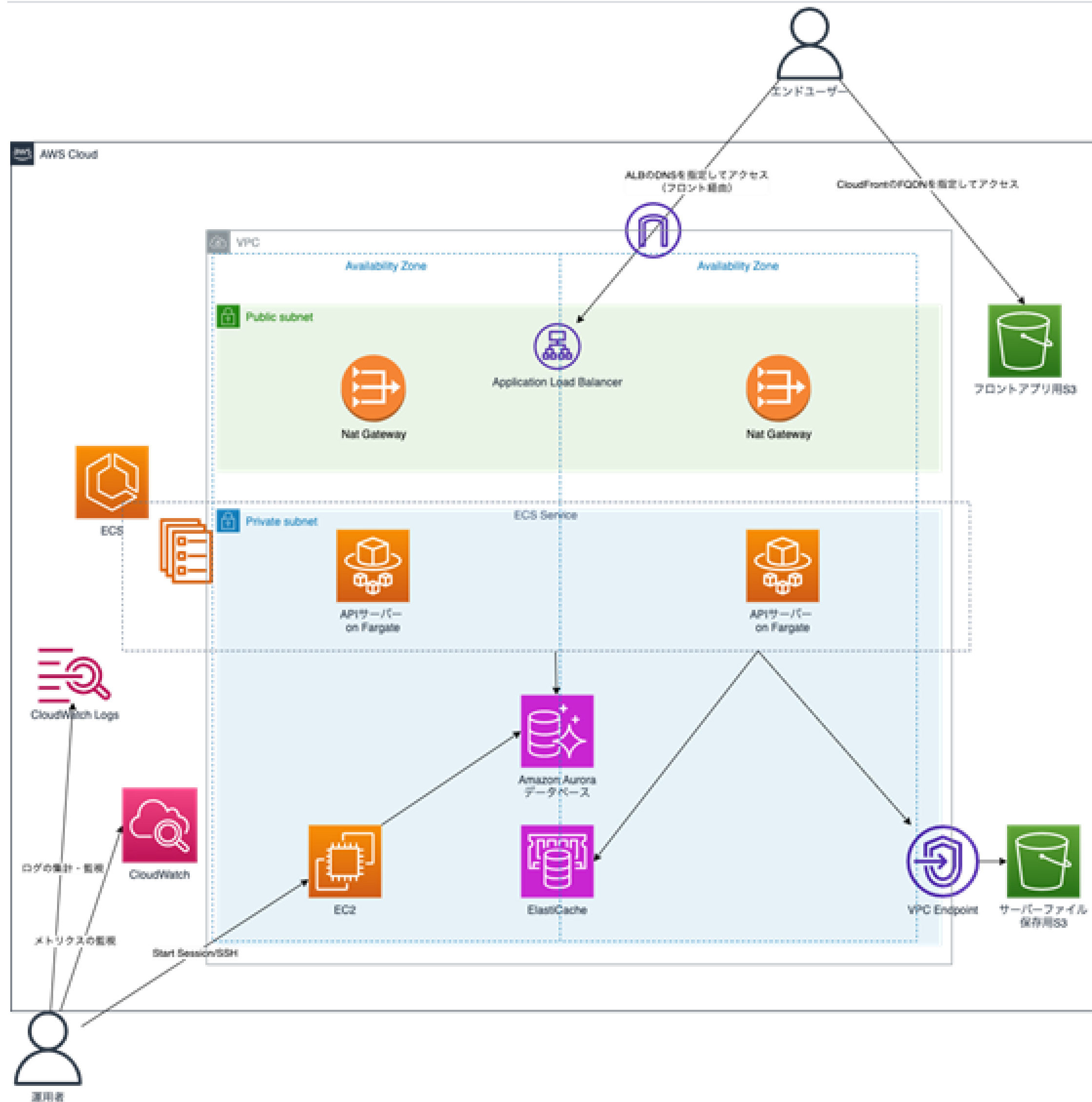
インターネットに出さずにAWS内のサービスにアクセス出来る道を開けてくれるAWSサービス

VPC内の通信を振り分けるルートテーブルの設定を行う必要はあるが、用意されたエンドポイントにアクセスした場合、その通信をAmazon S3に向けてくれるようになり、アクセスできるようになる

これを利用することで、インターネットを経由せずS3を使用することができるようになる

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく

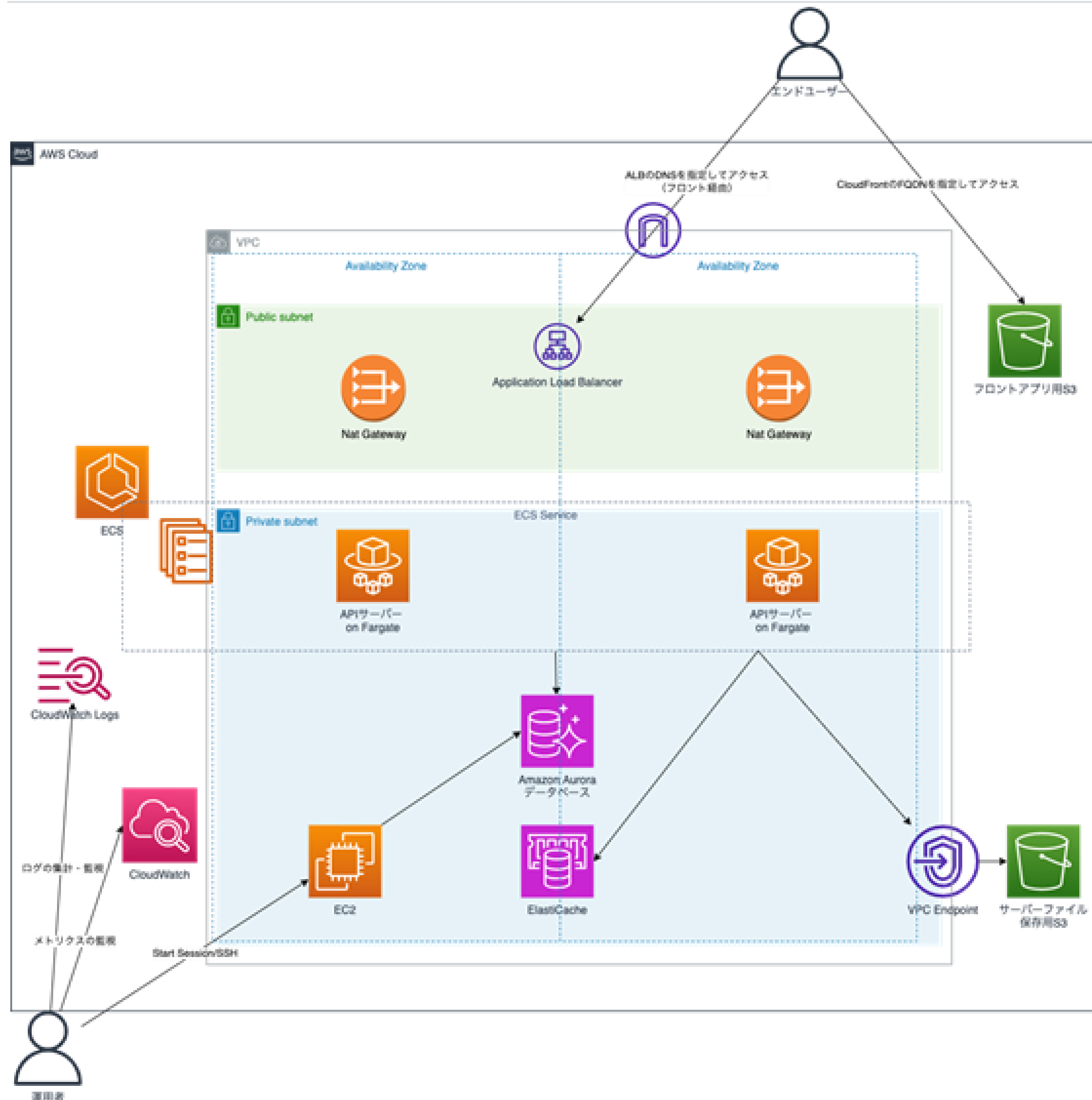


やったこと

- API・DB・キャッシュをプライベートサブネットに置いた
- ALBのみパブリックに置き、サーバーはALB経由でのみ許可をするようにした
- 運用者がDBを触れなくなってしまったので、必要に応じて都度踏み台EC2サーバーを立てる
- DBはサーバーと、踏み台EC2からのセキュリティグループを許可
- 踏み台のEC2は、運用者のアカウントからのみ入れる（StartSession）ようにした
- サーバー用S3は、サーバーが置いてあるVPCからのアクセスのみに制御した
- ネットワーク構成の変更のついでに、マルチAZ構成とした

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



嬉しいこと

- DBに接続できる人がシステムと運用者のみに限られた
- サーバーにインターネットから直接アクセスすることが出来なくなった
  - ALBにファイアウォールを置けば全リクエスト通すことができる
- サーバーファイル用のS3もシステムと運用者しかアクセス出来なくなった
- AZ障害が起きても、システムが落ちなくなった

やりたいこと

- さらなるセキュリティ向上のため、ファイアウォールを設置したい
- フロントのコンテンツ配信を早めたい
- 独自ドメインにしたい

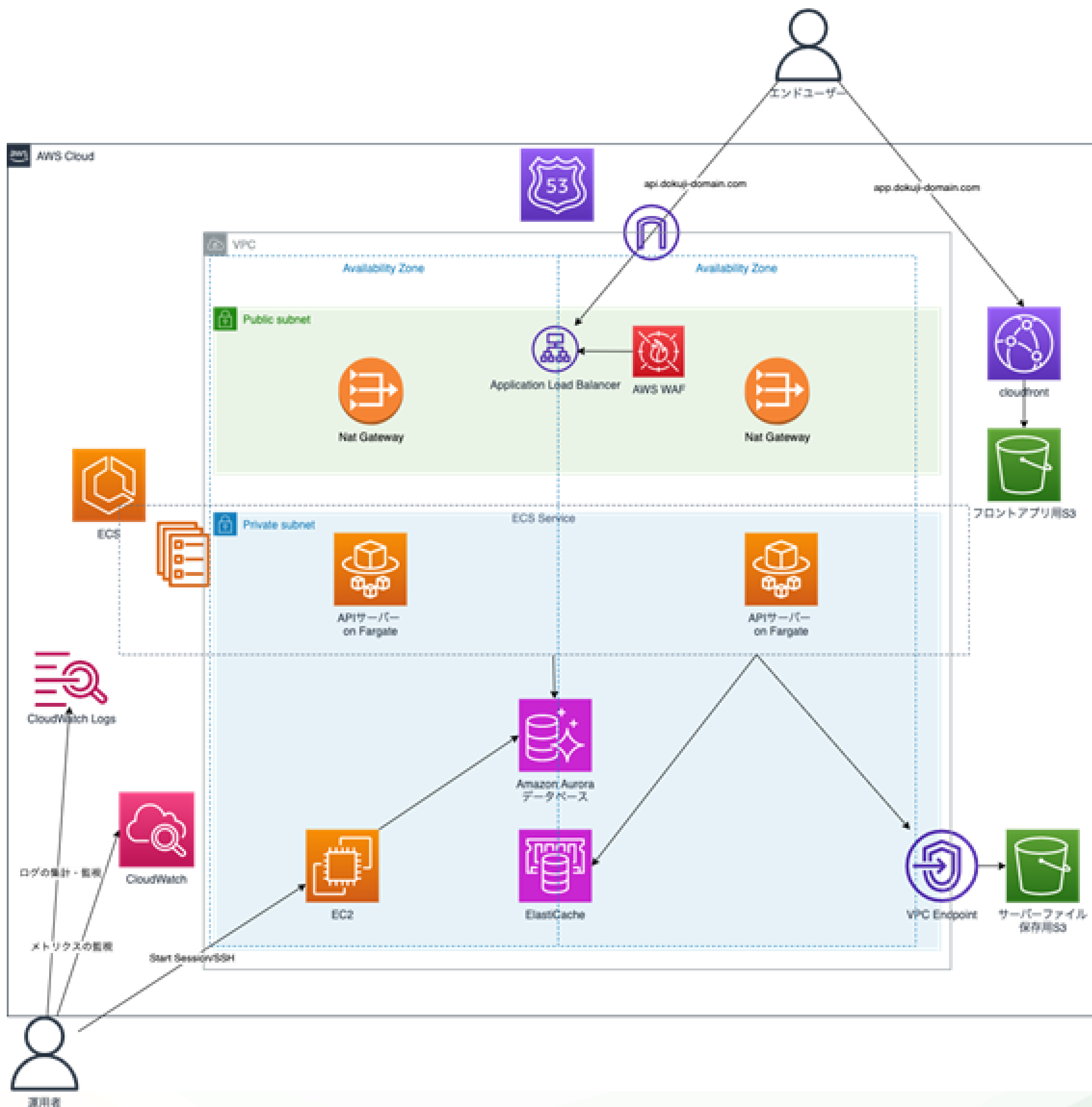
© 2024 Finatext Holdings Ltd.

AWS WAF・Amazon CloudFront・Amazon Route 53

AWS WAF	Amazon CloudFront	Amazon Route 53
ファイアウォールサービス	コンテンツ配信ネットワーク(CDN)	ドメインネームサービス（DNS）
条件に従ってリクエストをブロックすることが出来る	高可用性、スケーラビリティ、セキュリティを兼ね備えている	ドメインを購入したり、持っているドメインを登録してルーティング
ブロックのルールはAWSマネージドのものがあるので、それを使用するだけでも十分な効果あり	コンテンツをよりユーザーから物理的に近いところから配信することで、高速な配信が可能になる	S3/CloudFront/ALB等へのルーティングも容易に可能

© 2024 Finatext Holdings Ltd.

アーキテクチャを改善していく



やったこと

- フロント配信をCloudFront経由にした
- ドメインを購入し、Route53に登録した
- ALBにWAFを噛ませた

嬉しいこと

- コンテンツ配信が全世界に高速になった
- 覚えやすいURLでフロントサービスにアクセス出来るようになった
- SQL Injectionなどのリクエストを機械的にブロックしたり、IPの制限をかけられるようになった

一旦基本的な構成としては完成！

© 2024 Finatext Holdings Ltd.

要件に応じて出来ること

- 高トラフィックを捌きたい
  - APIサーバーの台数を増やす
  - Auroraリードレプリカ数を上げる
  - 非同期アーキテクチャを考える
    - Amazon SQS等
  - DDoSが原因ならAWS Shield
- 開発者のアクセスを監視したい
  - CloudTrailというAPIコールを確認できるサービスを使用して、ログを確認
  - EC2 StartSessionのログ収集
  - Aurora MySQLの監視ログ収集
- 監視結果に応じてSlackにログ通知したい
  - CloudWatchのAlarmからAmazon Lambda(サーバレスコンピューティング)を実行
- コストを抑えたい
  - AZ障害を許容して、単AZ構成にする
  - リソースを全体的に小さいサイズにする
  - インスタンスのまとめ買いをする

当初の構成では到底対応出来なかった要件も、自然にサービス追加で拡張したり、今ある機能だけで実現可能に！

© 2024 Finatext Holdings Ltd.

AWSをフルに利用して、運用面で柔軟で、簡単なシステムを構築しよう！

- 最小権限の原則
- トレーサビリティ（追跡可能性）を高める
- スケーラビリティ（拡張性）を高める
- オブザーバビリティ（可観測性）を高める
- アベイラビリティ（可用性）を高める
- デュラビリティ（耐久性）を高める
- 単一障害点を無くす
- 運用負荷を減らす
- マネージドサービスを使う

AWS Well-Architectedの観点に従ってAWSのサービスに置き換えていったところ…

要件への対応で分かる通り、当初の構成でより明らかに柔軟性が増したシステムに！

実際にはまだまだ紹介しきれていないサービスも

一口にDBと言ってもAurora以外にたくさんある

AWSのサービス間の比較が全く出来ていないが、

それを経験できるプログラムが…

© 2024 Finatext Holdings Ltd.

Finatext HD のサマーインターン

実施概要

申し込み
↓フォーム

毎年サマーインターン(有償)を実施しています！ぜひご参加ください！
2024年の実施概要は以下のとおりです。

実施期間　：　2024年8月26日(月)〜2024年8月30日(金)
要項ページ：https://hd.finatext.com/news/20240529/

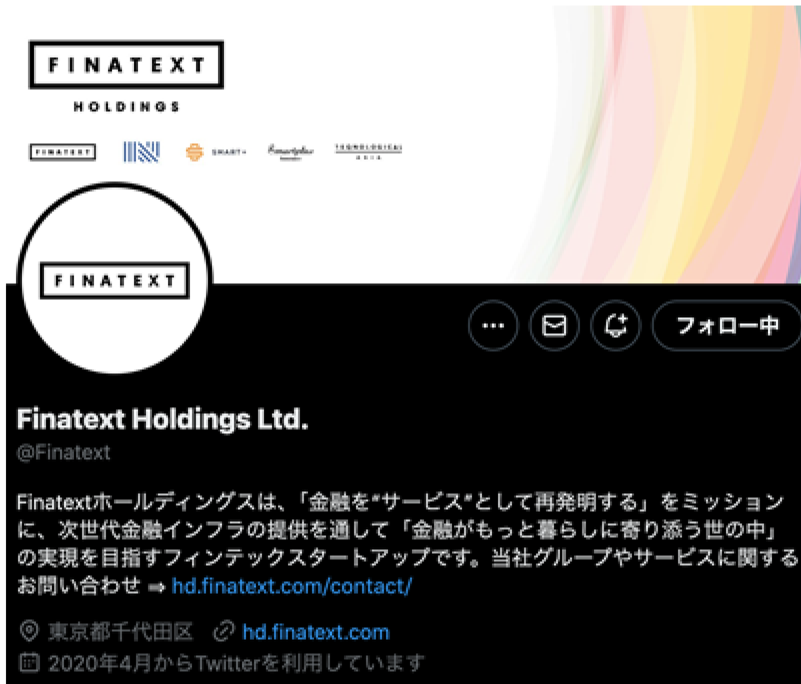
【ソフトウェアエンジニア】：
- Go言語とAWSを使ってシステムの開発やAWSの技術選定をやっていただきます
- AWSの方がゲストで来ます
- **今日の話を活かして実践出来ます！**

【データサイエンティスト】：
- Python/SQLで実践的ビッグデータ分析！
- 普段触れない、POSやクレカのデータを利用して実際の分析業務を体験できます

5 DAYS SUMMER INTERNSHIP PROGRAM 2024
FINATEXT HOLDINGS
エンジニア・データサイエンティスト職
26卒向け
証券システム開発のリアルやクラウドを利用したデータ分析を体験したい方必見!

Finatext HD の各種リンク

公式Xやテックブログもぜひご覧ください！



生成AIを用いた業務改善アイデアソンを開催しました

DBT Incremental Strategy and Idempotency

Rust関連のアウトプットの紹介

システムのエントロピーをコントロールすることの大切さ

公式X→

テックブログ→

© 2024 Finatext Holdings Ltd.

HOLDINGS