デベロッパーガイド

AWS App Runner

AWS App Runner: デベロッパーガイド

Table of Contents

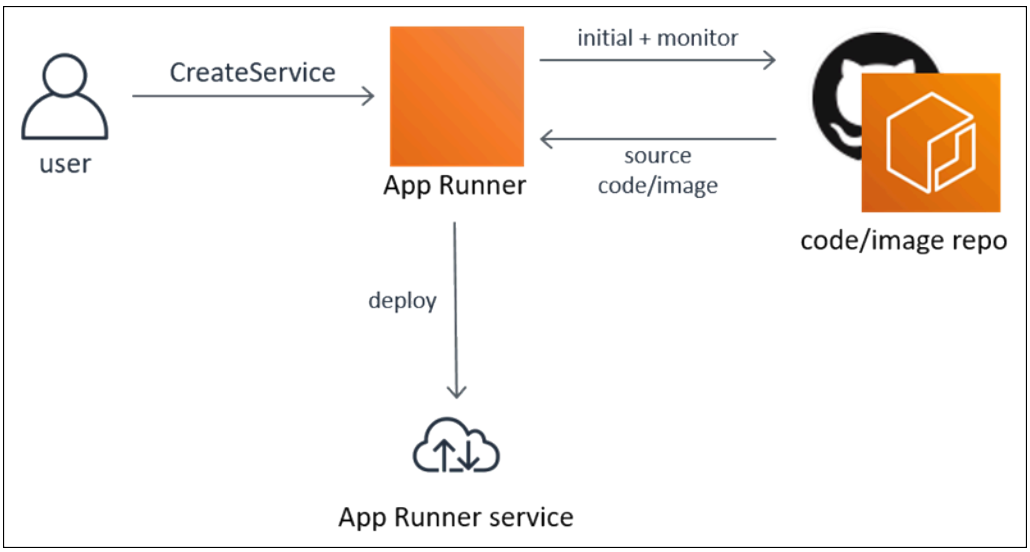
AWS App Runner は、2026 年 4 月 30 日以降、新規のお客様に公開されなくなります。App Runner を使用する場合は、その日付より前にサインアップします。既存のお客様は、通常どおりサービスを引き続き使用できます。詳細については、「AWS App Runner 可用性の変更」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。

とは AWS App Runner

AWS App Runner は、ソースコードまたはコンテナイメージから AWS クラウド内のスケーラブルで安全なウェブアプリケーションに直接デプロイするための、高速でシンプルで費用対効果の高い方法を提供する AWS サービスです。新しいテクノロジーを学習したり、使用するコンピューティングサービスを決定したり、 AWS リソースをプロビジョニングして設定する方法を知っている必要はありません。

App Runner は、コードまたはイメージリポジトリに直接接続します。フルマネージドオペレーション、ハイパフォーマンス、スケーラビリティ、セキュリティを備えた自動統合および配信パイプラインを提供します。



App Runner とは

開発者は、App Runner を使用して、コードまたはイメージリポジトリの新しいバージョンをデプロイするプロセスを簡素化できます。

オペレーションチームの場合、App Runner は、コミットがコードリポジトリにプッシュされるたびに、または新しいコンテナイメージバージョンがイメージリポジトリにプッシュされるたびに、自動デプロイを有効にします。

App Runner へのアクセス

App Runner サービスのデプロイは、次のいずれかのインターフェイスを使用して定義および設定できます。

- App Runner コンソール – App Runner サービスを管理するためのウェブインターフェイスを提供します。
- App Runner API – App Runner アクションを実行するための RESTful API を提供します。詳細については、「AWS App Runner API リファレンス」を参照してください。
- AWS コマンドラインインターフェイス (AWS CLI) – Amazon VPC を含む幅広い AWS サービスのコマンドを提供し、Windows、macOS、Linux でサポートされています。詳細については、「AWS Command Line Interface」を参照してください。
- AWS SDKs – 言語固有の APIsを提供し、署名の計算、リクエストの再試行の処理、エラー処理など、接続の詳細の多くを処理します。詳細については、AWS SDKを参照してください。

App Runner の料金

App Runner は、アプリケーションを実行するコスト効率の高い方法を提供します。App Runner サービスが消費するリソースに対してのみ料金が発生します。リクエストトラフィックが低い場合、サービスはスケールダウンしてコンピューティングインスタンス数を減らします。プロビジョニングされたインスタンスの最小数と最大数、およびインスタンスが処理する最大負荷のスケーラビリティ設定を制御できます。

App Runner 自動スケーリングの詳細については、「」を参照してくださいthe section called “Auto scaling”。

料金情報については、「AWS App Runner の料金」を参照してください。

次のステップ

以下のトピックで App Runner の使用を開始する方法について説明します。

- 設定 – App Runner を使用するための前提条件ステップを完了します。
- 開始方法 – 最初のアプリケーションを App Runner にデプロイします。

AWS App Runner 可用性の変更

慎重に検討した結果、2026 年 4 月 30 日から新規顧客 AWS App Runner を閉鎖することになりました。既存の AWS App Runner お客様は、新しいリソースやサービスの作成など、通常どおりサービスを引き続き使用できます。 AWS は引き続きセキュリティと可用性に投資しますが AWS App Runner、新機能を導入する予定はありません。

移行時に Amazon Elastic Container Service (Amazon ECS) Express Mode を試すことをお勧めします AWS App Runner。Amazon ECS Express Mode は、App Runner の運用上のシンプルさを維持しながら、より広範な Amazon ECS 機能セットへのアクセスを提供します。1 回の API コールで、コンテナイメージと 2 つの IAM ロールを提供し、Amazon ECS は、Fargate の ECS サービス、Application Load Balancer、自動スケーリング、ネットワークなど、完全なアプリケーションスタックを AWS アカウントにプロビジョニングします。Amazon ECS Express Mode の使用には追加料金はかかりません。アプリケーションを実行するために作成された基盤となる AWS リソースに対してのみ料金が発生します。

このガイドでは、既存の App Runner サービスを ECS Express Mode に移行し、DNS ルーティングを使用してトラフィックを徐々に移行する方法について説明します。

移行の概要

このガイドでは、DNS 加重ルーティングを使用した Blue/Green デプロイアプローチを使用して、App Runner から ECS Express Mode にトラフィックを移行します。両方のサービスは移行中に同時に実行されます。Amazon Route 53 (または DNS プロバイダー) を使用して、トラフィックを App Runner サービスから ECS Express Mode サービスに徐々に移行します。最初はわずかな割合から始まり、時間の経過とともに増加します。このアプローチによりダウンタイムが最小限に抑えられ、問題が発生した場合は DNS の重みを調整してロールバックできます。

一般的な移行には、次のステップが含まれます。

1. 既存の App Runner サービスの設定を確認する
2. 同じコンテナイメージを使用して ECS Express Mode サービスを作成する
3. カスタムドメインを使用する場合、ECS Express Mode サービスに同じカスタムドメインを設定する
4. DNS ルーティングを使用して App Runner から ECS Express Mode にトラフィックを移行する
5. 移行を完了し、不要になった App Runner サービスを削除する

前提条件

開始する前に、以下があることを確認してください。

- Amazon ECS、、Amazon Route 53 AWS App Runner、Application Load Balancer リソースを作成および管理するための適切な AWS Identity and Access Management アクセス許可を持つ AWS アカウント
- AWS CLI AWS アカウントの認証情報を使用してインストールおよび設定されている
- ECS Express Mode にデプロイするために Amazon Elastic Container Registry (または別のコンテナレジストリ) に保存されているコンテナイメージ
- ECS Express Mode に必要な IAM ロール: Amazon ECS タスク実行`ecsTaskExecutionRole`用および ECS Express Mode インフラストラクチャプロビジョニング`ecsInfrastructureRoleForExpressServices`用

移行中に既存のカスタムドメインを保持する場合は、以下も必要です。

- Amazon Route 53 またはサードパーティーのドメインレジストラを使用して`app.example.com`、 など、ユーザーが管理する登録済みドメイン名
- カスタムドメインに一致する AWS Certificate Manager (ACM) の SSL/TLS 証明書。リソースをデプロイ AWS リージョン するのと同じ でパブリック ACM 証明書をリクエストします。App Runner と Amazon ECS Express Mode の両方で、カスタムドメインで HTTPS アクセスを有効にするには ACM 証明書が必要です。

[開始する前に]

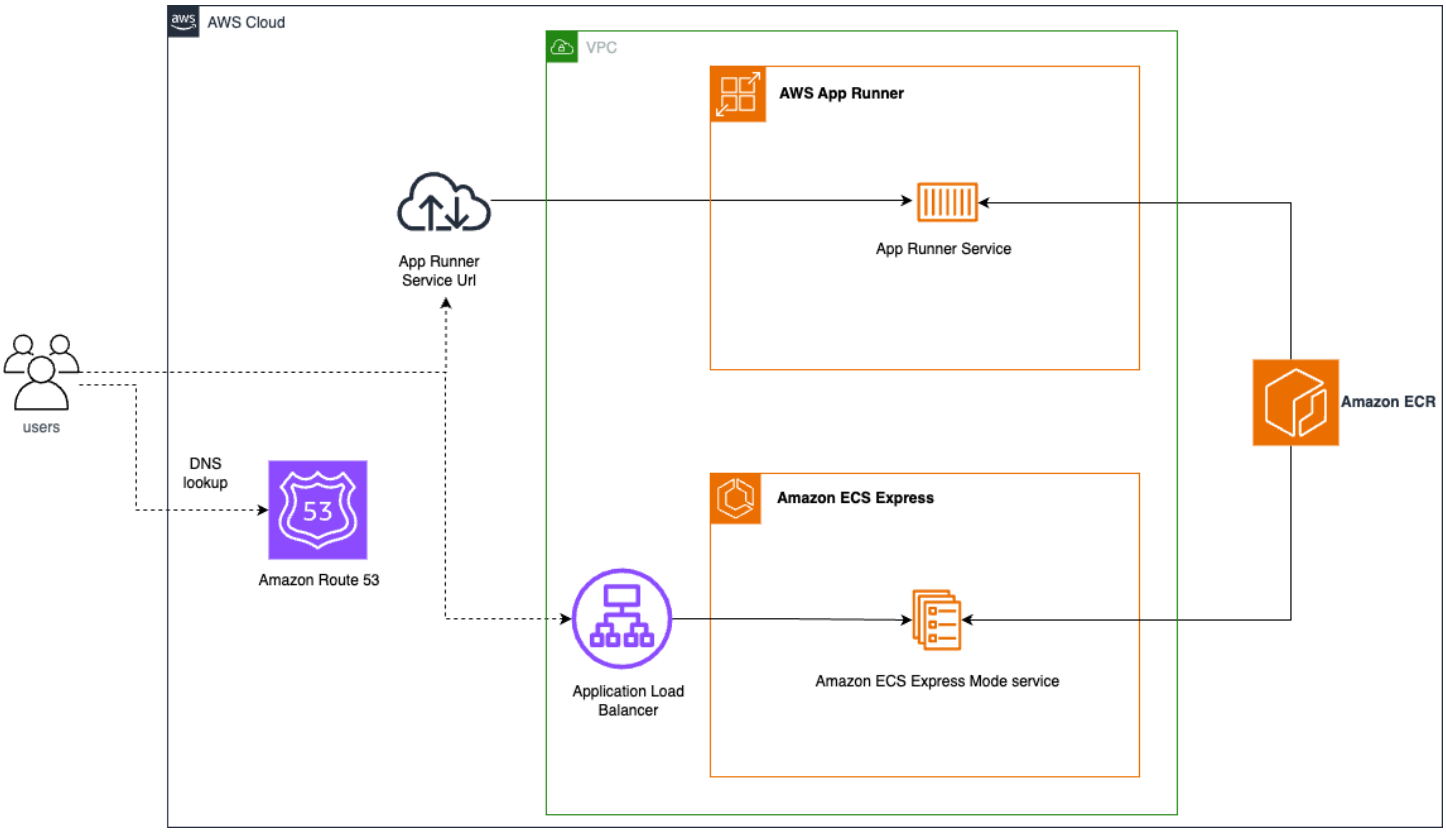
- コンテナイメージの要件 — ECS Express Mode はコンテナイメージをデプロイします。App Runner サービスがソースコードからデプロイされている場合は、まずコンテナイメージを作成し、Amazon Elastic Container Registry などのレジストリにプッシュするビルドステップを追加します。次に、そのイメージを ECS Express Mode にデプロイします。ソースベースのデプロイの移行の詳細については、ソースベースのデプロイの移行「」を参照してください。
- ドメインの動作 — App Runner サービスが などのカスタムドメインを既に使用している場合は、移行中に同じホスト名を再利用し`app.example.com`、DNS を更新して App Runner と ECS Express Mode の間でトラフィックを徐々に移行できます。

  App Runner サービスがデフォルトの App Runner サービス URL のみを使用している場合、ECS Express Mode サービスには異なるエンドポイントがあります。この場合、段階的なトラフィック

シフトに使用できる共有ホスト名はありません。ECS Express Mode サービスを作成して検証し、新しいエンドポイントを使用するようにクライアントまたは DNS を更新する必要があります。

移行のチュートリアル

次の図は、Route 53 を使用して App Runner サービスと ECS Express Mode サービス間で DNS レコードを移行する方法を示しています。



ステップ 1: 既存の App Runner 設定を確認する

App Runner コンソールで、既存のサービスを確認し、続行する値を書き留めます。少なくとも、次の点に注意してください。

- コンテナイメージ
- アプリケーションポート
- 環境変数
- 設定されている場合、カスタムドメイン名
- 設定されている場合、カスタムドメインに関連付けられた ACM 証明書

新しいサービスに引き継ぐその他のランタイム設定を確認することもできます。

カスタムドメインの詳細については、「」を参照してくださいthe section called “カスタムドメイン名”。

ステップ 2: ECS Express Mode サービスを作成する

App Runner サービスで使用されるのと同じコンテナイメージを使用して ECS Express Mode サービスを作成します。または AWS マネジメントコンソール を使用してサービスを作成できますAWS CLI。

CLI コマンドの例:

```
aws ecs create-express-gateway-service \
    --execution-role-arn arn:aws:iam::123456789012:role/ecsTaskExecutionRole \
    --infrastructure-role-arn arn:aws:iam::123456789012:role/
ecsInfrastructureRoleForExpressServices \
    --primary-container '{
        "image": "123456789012.dkr.ecr.us-east-1.amazonaws.com/my-app:latest",
        "containerPort": 8080,
        "environment": [{
            "name": "ENV_VAR_NAME",
            "value": "value"
        }]
    }' \
    --service-name "my-application" \
    --health-check-path "/" \
    --scaling-target '{"minTaskCount":1,"maxTaskCount":4}' \
    --monitor-resources
```

イメージ、ポート、環境変数、スケーリング値を App Runner サービスの値に置き換えます。

このコマンドは、Fargate の ECS サービス、ターゲットグループとヘルスチェックを備えた Application Load Balancer、自動スケーリングポリシー、セキュリティグループとネットワーク設定、デフォルト URL など、 AWS アカウントの完全なアプリケーションスタックをプロビジョニングします。

プロビジョニングには通常 3～5 分かかります。Amazon ECS コンソールのリソースタブで進行状況を追跡できます。

完了したら、コンソールに表示されるデフォルトの URL を使用して ECS Express Mode サービスをテストします。トラフィックシフトに進む前に、アプリケーションが正しく動作することを確認します。

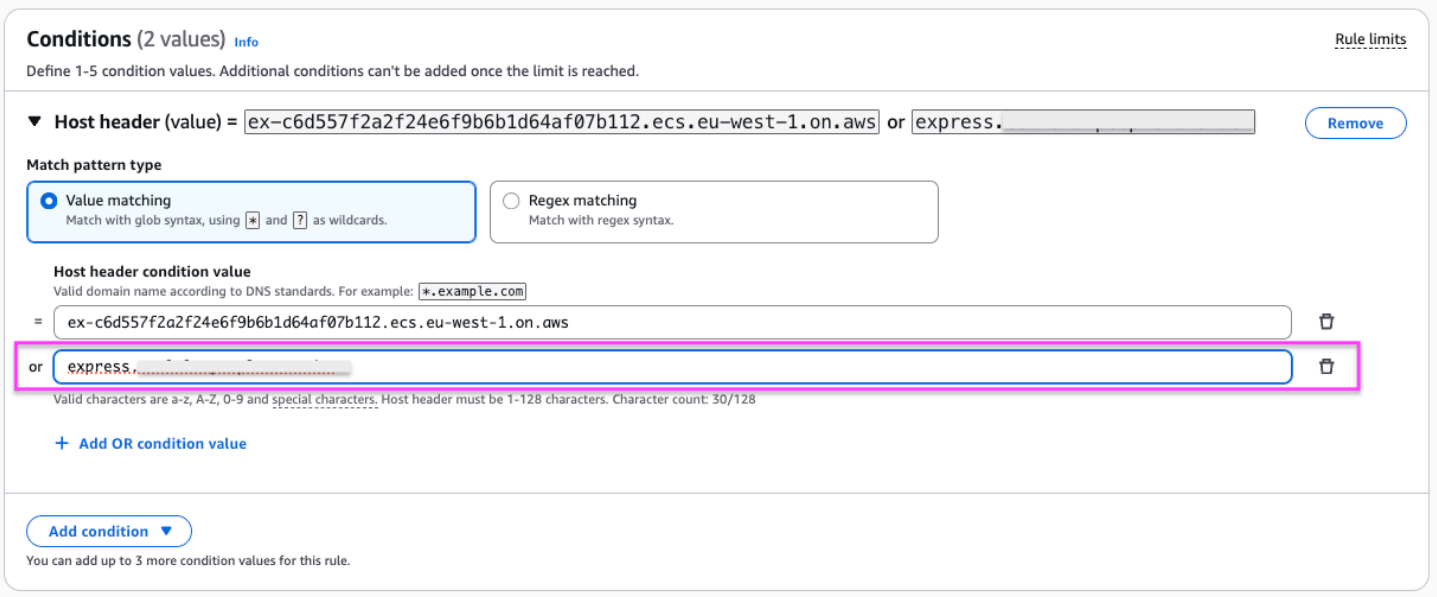
ステップ 3: ECS Express Mode のカスタムドメインを設定する

App Runner サービスがカスタムドメインを使用している場合は、トラフィックを移行する前に ECS Express Mode サービスに同じカスタムドメインを設定します。このステップでは、ECS Express Mode サービス用に作成された Application Load Balancer を設定して、ドメインのトラフィックを受け入れ、HTTPS に ACM 証明書を使用します。

- Application Load Balancer リスナールールで、カスタムドメインをホストヘッダー条件として追加します。App Runner サービスに関連付けたのと同じドメイン名を使用します (例: `app.example.com`)。これにより、ドメインから ECS Express Mode ターゲットグループにトラフィックをルーティングするように Application Load Balancer に指示します。
- Application Load Balancer HTTPS リスナーに SSL 証明書を追加します。ステップ 1 でメモした ACM 証明書を HTTPS リスナーに追加します。

詳細な手順については、「Amazon ECS デベロッパーガイド」の「サービスへのカスタムドメインの追加」を参照してください。

次の図は、Application Load Balancer リスナールールでホストヘッダー条件を設定する例を示しています。



ステップ 4: Route 53 加重ルーティングを使用してトラフィックをシフトする

App Runner サービスが既にカスタムドメインを使用している場合は、Route 53 加重ルーティングを使用してトラフィックを ECS Express Mode サービスに徐々に移行できます。加重ルーティングを使用すると、同じホスト名のトラフィックを複数のエンドポイントにルーティングできます。各エンドポイントは、独自の重みを持つ個別の DNS レコードとして定義され、Route 53 はそれらの重みに従ってリクエストを分散します。

> **Note**
>
> このガイドでは、Route 53 を例として使用します。別の DNS プロバイダーを使用する場合は、プロバイダーのトラフィック管理機能を使用して同等の DNS 変更を行います。

既存の App Runner レコードを加重レコードに変換します。

1. Route 53 コンソールを開きます。
2. ホストゾーンを選択し、ドメインのホストゾーンを選択します。
3. 現在 App Runner を指しているホスト名の既存のレコード (など`app.example.com`) を見つけます。
4. レコードを編集し、そのルーティングポリシーを加重に変更します。
5. Weight を に設定します `100` (これにより、すべての初期トラフィックが App Runner に送信されます) 。
6. レコード ID に、 などのわかりやすい識別子を入力します`app-runner-service`。
7. [Save changes] (変更の保存) をクリックします。

ECS Express Mode の加重レコードを作成します。

1. 同じホストゾーンに新しいレコードを作成します。
2. 同じレコード名を使用します (例: `app.example.com`)。
3. 同じレコードタイプを使用します。
4. ルーティングポリシーを加重に設定します。
5. Route traffic to で、Alias to Application and Classic Load Balancer を選択します。

6. ドロップダウンから ECS Express Mode Application Load Balancer を選択します。
7. Weight を に設定します `0` (明示的にウェイトを増やすまで、トラフィックは ECS Express Mode に流れません)。
8. レコード ID に、 などのわかりやすい識別子を入力しますecs-express-service。
9. [レコードを作成] を選択します。

トラフィックを徐々にシフトします。

DNS レコードを設定したら、ECS Express Mode の重みを増やしながら、App Runner の重みを比例的に減らしてトラフィックの移行を開始します。推奨されるアプローチ:

- ECS Express Mode を 10 に設定/App Runner を 90 に設定
- リクエストを正常に処理するサービスをモニタリングおよび検証する
- を 25 / 75 に増やす
- を 50 / 50 に増やす
- を 75 / 25 に増やす
- 100 / 0 で完了

各ステップで、追加のトラフィックを移行する前にアプリケーションをテストします。いずれかの時点で問題が発生した場合は、重みを以前の値に調整してロールバックします。

> **Important**
>
> App Runner サービスを検証期間 (24～48 時間など) にわたって実行し、DNS の変更がグローバルに伝達されたことを確認し、必要に応じてロールバックオプションを提供します。問題が発生した場合は、Route 53 の重みをすばやく App Runner に戻すことができます。

ステップ 5: 移行を完了する

ECS Express Mode サービスが本番トラフィックを正しく処理し、検証期間が経過したことを確認したら、移行を完了します。

- Route 53 で、App Runner を指す加重レコードを削除します (または重みを 0 に設定します)。
- App Runner サービスからカスタムドメインの関連付けを削除します。

App Runner サービスを削除します。

```
aws apprunner delete-service --service-arn your-app-runner-service-arn
```

また、不要になったリソースを削除することを検討してください。

- App Runner の Route 53 加重ルーティングレコード
- Amazon Elastic Container Registry の未使用のコンテナイメージ
- 不要になった場合に App Runner 専用に作成された IAM ロール

> **Note**
>
> サービスが本番環境で実行されている場合は、ECS Express Mode サービス、Application Load Balancer、または関連するリソースを削除しないでください。

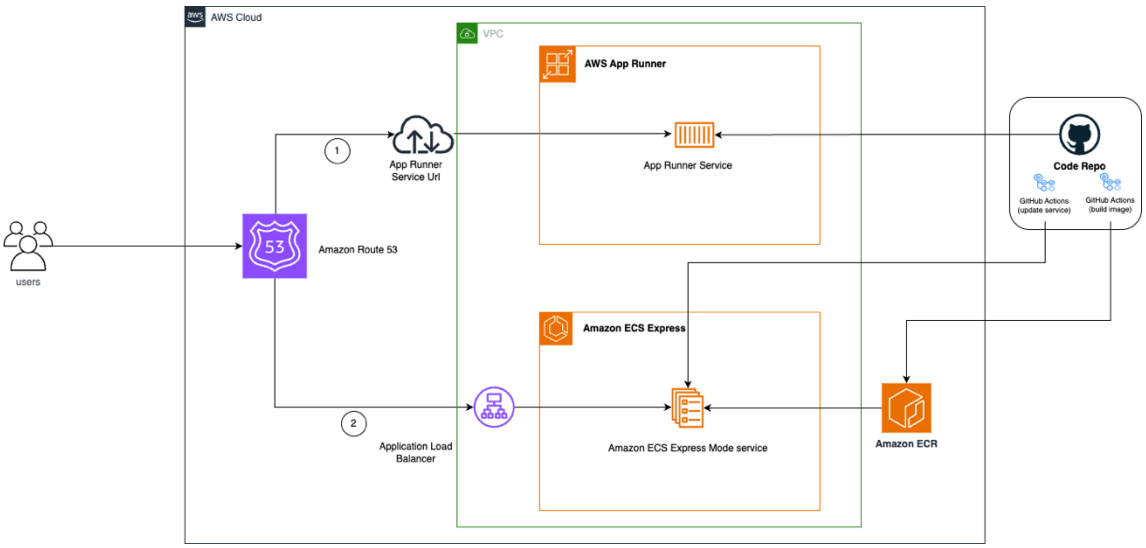
ソースベースのデプロイの移行

既存の App Runner サービスがコンテナイメージではなくソースコードからデプロイされている場合は、ECS Express Mode にデプロイする前にコンテナ化ステップを追加する必要があります。App Runner とは異なり、ECS Express Mode にはコンテナイメージが必要です。ただし、GitHub Actions などの CI/CD ツールと Amazon ECS Deploy Express Service GitHub Action を使用して、App Runner の自動デプロイエクスペリエンスをレプリケートできます。

移行ワークフローには 3 つのステージがあります。

1. Dockerfile を使用してコンテナイメージを構築する
2. Amazon Elastic Container Registry などのコンテナレジストリにイメージをプッシュする
3. ECS Express Mode にイメージをデプロイする

次の図は、このワークフローが GitHub Actions を使用してどのように機能するかを示しています。



アプリケーションをコンテナ化する

アプリケーションに Dockerfile がまだない場合は、リポジトリルートに作成します。Dockerfile は、ソースコードを構築してコンテナイメージにパッケージ化するための設計図として機能します。

リポジトリ構造には以下を含める必要があります。

```
your-app/
### src/                    # Application source code
### Dockerfile              # Container build instructions
### package.json            # Dependencies and scripts
### .github/                # GitHub configuration
    ### workflows/          # GitHub Actions workflows
        ### deploy.yml      # ECS Express Mode deployment workflow
```

自動デプロイ用に GitHub Actions を設定する

コードプッシュ時に App Runner の自動デプロイをレプリケートするには、以下を使用して GitHub Actions を設定します。

- OpenID Connect (OIDC) プロバイダーを作成して、GitHub Actions が IAM ロールを引き受けることを許可する
- ECS Express Mode と Amazon Elastic Container Registry のアクセス許可を持つ IAM ロールを作成する https://docs.aws.amazon.com/AmazonECR/latest/userguide/ECR_on_ECS.html
- ECS Express Mode に必要な 2 つの IAM ロールを作成する

- ECS リソースの GitHub 環境変数を作成する: `ECS_SERVICE`、`ECS_CLUSTER`、`AWS_REGION`、`AWS_ACCOUNT_ID`、および `ECR_REPOSITORY`

GitHub Actions ワークフローの例

でワークフローファイルを作成します`.github/workflows/deploy.yml`。

```
name: Build and Deploy to ECS

on:
  push:
    branches: [ main ]

env:
  AWS_REGION: ${{ vars.AWS_REGION }}
  AWS_ACCOUNT_ID: ${{ vars.AWS_ACCOUNT_ID }}
  ECR_REPOSITORY: ${{ vars.ECR_REPOSITORY }}
  ECS_SERVICE: ${{ vars.ECS_SERVICE }}
  ECS_CLUSTER: ${{ vars.ECS_CLUSTER }}

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    environment: production
    permissions:
      id-token: write
      contents: read

    steps:
    - name: Checkout
      uses: actions/checkout@v6

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v5
      with:
        aws-region: ${{ env.AWS_REGION }}
        role-to-assume: arn:aws:iam::${{ env.AWS_ACCOUNT_ID }}:role/github-actions-ecs-
role
        role-session-name: GitHubActionsECSDeployment

    - name: Login to Amazon ECR
      id: login-ecr
```

```
      uses: aws-actions/amazon-ecr-login@v2

    - name: Get short commit hash
      run: echo "IMAGE_TAG=${GITHUB_SHA:0:7}" >> $GITHUB_ENV

    - name: Build, tag, and push image to Amazon ECR
      id: build-image
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      uses: docker/build-push-action@v6
      with:
        context: .
        push: true
        tags: ${{ env.ECR_REGISTRY }}/${{ env.ECR_REPOSITORY }}:latest,
${{ env.ECR_REGISTRY }}/${{ env.ECR_REPOSITORY }}:${{ env.IMAGE_TAG }}

    - name: Deploy to ECS Express Mode
      uses: aws-actions/amazon-ecs-deploy-express-service@v1
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      with:
        service-name: ${{ env.ECS_SERVICE }}
        image: ${{ env.ECR_REGISTRY }}/${{ env.ECR_REPOSITORY }}:${{ env.IMAGE_TAG }}
        execution-role-arn: arn:aws:iam::${{ env.AWS_ACCOUNT_ID }}:role/
ecsTaskExecutionRole
        infrastructure-role-arn: arn:aws:iam::${{ env.AWS_ACCOUNT_ID }}:role/
ecsInfrastructureRoleForExpressServices
        cluster: ${{ env.ECS_CLUSTER }}
        container-port: 8080
        environment-variables: |
          [
            {"name": "ENV", "value": "Prod"}
          ]
        cpu: '1024'
        memory: '2048'
        health-check-path: /health
        min-task-count: 1
        max-task-count: 4
        auto-scaling-metric: AVERAGE_CPU
        auto-scaling-target-value: 70
```

メインブランチにコード変更をプッシュすると、GitHub Actions は自動的に新しいコンテナイメージを構築し、Amazon Elastic Container Registry にプッシュして、ECS Express Mode サービスにデ

プロイします。これにより、App Runner での自動デプロイエクスペリエンスがレプリケートされます。

ECS Express Mode サービスが実行されたら、移行チュートリアルのステップ 3～5 に従ってカスタムドメインを設定し、DNS ルーティングを使用してトラフィックをシフトし、移行を完了します。

その他のリソース

- を使用して最初の高速モードサービスを作成する AWS CLI
- コンソールで最初の Amazon ECS Express Mode サービスを作成する
- Express Mode 以外のリソースの更新
- the section called “カスタムドメイン名”
- Amazon Route 53 加重ルーティング

App Runner のセットアップ

新規の AWS お客様は、 の使用を開始する前に、このページに記載されているセットアップの前提条件を完了してください AWS App Runner。

これらのセットアップ手順では、 AWS Identity and Access Management (IAM) サービスを使用します。IAM の詳細については、以下の資料を参照してください。

- AWS Identity and Access Management (IAM)
- IAM ユーザーガイド

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 を保護し AWS IAM アイデンティティセンター、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS マネジメントコンソールとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、「AWS サインイン ユーザーガイド」の「ルートユーザーとしてサインインする」を参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの AWS アカウント 「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「AWS IAM アイデンティティセンターの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 AWS IAM アイデンティティセンター ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「アクセス許可セットを作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM アイデンティティセンター ユーザーガイド」の「グループを追加する」を参照してください。

プログラム的なアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS マネジメントコンソール。プログラムによるアクセスを許可する方法は、 がアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラムによるアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	(推奨) コンソール認証情報を一時的な認証情報として使用して AWS CLI、、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、AWS Command Line Interface「 ユーザーガイド」のAWS「ローカル開発のための口グイン」を参照してください。

プログラムによるアクセス権を必要とするユーザー	目的	方法
		• AWS SDKs「 SDK およびツールリファレンスガイド」のAWS 「ローカル開発用のログイン」を参照してください。 AWS SDKs
ワークフォースアイデンティティ (IAM アイデンティティセンターで管理されているユーザー)	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 を使用する AWS CLI ように AWS IAM アイデンティティセンターを設定する」を参照してください。 • AWS SDKs、ツール、API については、 AWS APIs「 SDK およびツールリファレンスガイド」の「IAM アイデンティティセンター認証」を参照してください。 AWS SDKs
IAM	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	(非推奨) 長期認証情報を使用して、AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 AWS SDKs • API AWS APIs「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

次のステップ

前提条件のステップを完了しました。最初のアプリケーションを App Runner にデプロイするには、「」を参照してください開始方法。

App Runner の開始方法

AWS App Runner は、既存のコンテナイメージまたはソースコードを で実行中のウェブ AWS サービスに直接変換するための、高速でシンプルで費用対効果の高い方法を提供する サービスです AWS クラウド。

このチュートリアルでは、 AWS App Runner を使用してアプリケーションを App Runner サービスにデプロイする方法について説明します。ソースコードとデプロイ、サービスビルド、サービスランタイムの設定について説明します。また、コードバージョンをデプロイする方法、設定変更を行う方法、ログを表示する方法も示します。最後に、チュートリアルの手順に従って作成したリソースをクリーンアップする方法を示します。

トピック

前提条件

チュートリアルを開始する前に、必ず次のアクションを実行してください。

1. のセットアップ手順を完了します設定。
2. GitHub リポジトリと Bitbucket リポジトリのどちらを使用するかを決定します。
   - Bitbucket を使用するには、Bitbucket アカウントをまだ作成していない場合は、まず作成します。Bitbucket を初めて使用する場合は、Bitbucket クラウドドキュメントの「Bitbucket の開始方法」を参照してください。
   - GitHub を使用するには、GitHub アカウントをまだお持ちでない場合は作成します。GitHub を初めて使用する場合は、GitHub ドキュメントの「GitHub の開始方法GitHub」を参照してください。

> **Note**
>
> アカウントから複数のリポジトリプロバイダーへの接続を作成できます。したがって、GitHub リポジトリと Bitbucket リポジトリの両方からデプロイを実行する場合は、この手順を繰り返すことができます。次に、 を使用して新しい App Runner サービスを作成し、他のリポジトリプロバイダー用の新しいアカウント接続を作成します。

3. リポジトリプロバイダーアカウントにリポジトリを作成します。このチュートリアルでは、リポジトリ名 を使用しますpython-hello。次の例で指定された名前とコンテンツを使用して、リポジトリのルートディレクトリにファイルを作成します。

`python-hello` サンプルリポジトリのファイル

Example requirements.txt

```
pyramid==2.0
```

Example server.py

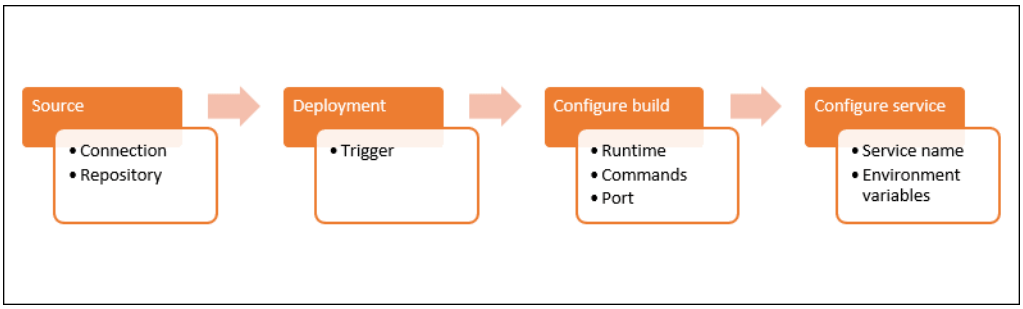
```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
import os

def hello_world(request):
    name = os.environ.get('NAME')
    if name == None or len(name) == 0:
        name = "world"
    message = "Hello, " + name + "!\n"
    return Response(message)

if __name__ == '__main__':
    port = int(os.environ.get("PORT"))
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', port, app)
    server.serve_forever()
```

ステップ 1: App Runner サービスを作成する

このステップでは、 の一部として GitHub または Bitbucket で作成したサンプルソースコードリポジトリに基づいて App Runner サービスを作成しますthe section called “前提条件”。この例には、シンプルな Python ウェブサイトが含まれています。サービスを作成するために実行する主なステップは次のとおりです。

1. ソースコードを設定します。
2. ソースデプロイを設定します。
3. アプリケーションビルドを設定します。
4. サービスを設定します。
5. 確認して確認します。

次の図は、App Runner サービスを作成する手順の概要を示しています。

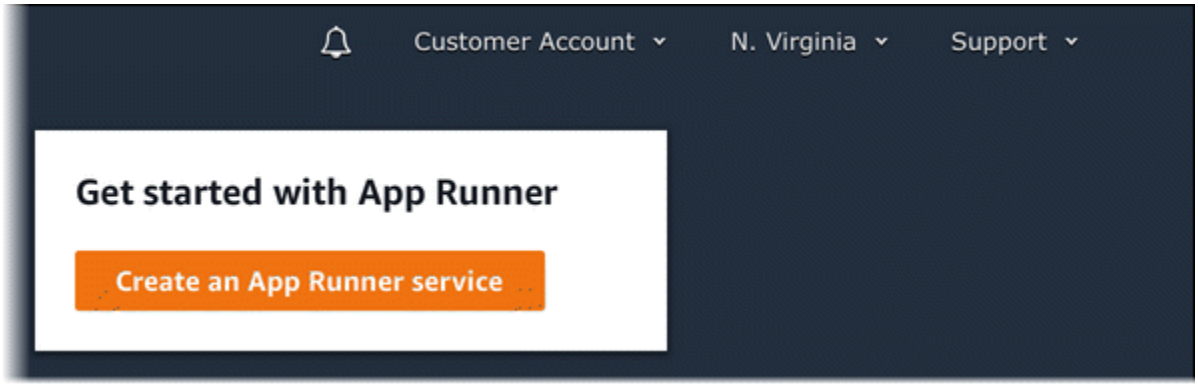


ソースコードリポジトリに基づいて App Runner サービスを作成するには

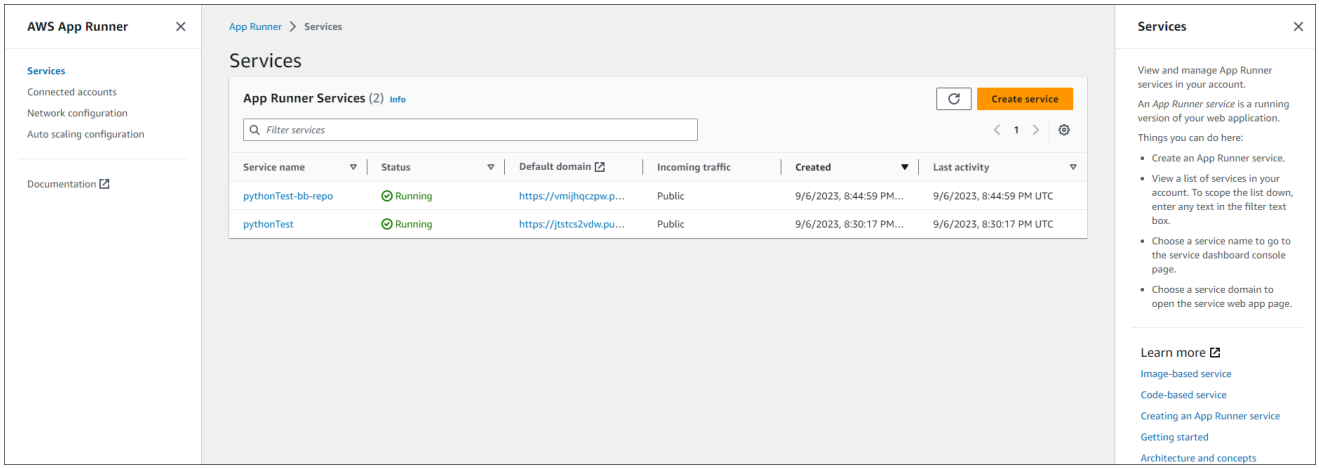
1. ソースコードを設定します。

   a. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。

   b. AWS アカウント に App Runner サービスがまだない場合は、コンソールのホームページが表示されます。App Runner サービスの作成を選択します。



に既存のサービス AWS アカウント がある場合、サービスのリストを含むサービスページが表示されます。[Create service (サービスの作成)] を選択します。



c. 「ソースとデプロイ」ページの「ソース」セクションの「リポジトリタイプ」で、「ソースコードリポジトリ」を選択します。

d. プロバイダータイプを選択します。GitHub または Bitbucket を選択します。

e. 次に、新規追加を選択します。プロンプトが表示されたら、GitHub または Bitbucket の認証情報を入力します。

f. 以前に選択したプロバイダータイプに基づいて、次の一連のステップを選択します。

> **Note**
>
> GitHub の AWS コネクタを GitHub アカウントにインストールする次の手順は、1 回限りのステップです。接続を再利用して、このアカウントのリポジトリに基づいて複数の App Runner サービスを作成できます。既存の接続がある場合は、その接続を選択し、リポジトリの選択にスキップします。
> Bitbucket アカウントの AWS コネクタにも同じことが当てはまります。App Runner サービスのソースコードリポジトリとして GitHub と Bitbucket の両方を

使用している場合は、プロバイダーごとに 1 つの AWS Connector をインストールする必要があります。その後、各コネクタを再利用して、より多くの App Runner サービスを作成できます。

- GitHub の場合は、以下の手順に従います。

  i. 次の画面で、接続名を入力します。

  ii. App Runner で GitHub を初めて使用する場合は、別の のインストール を選択します。

  iii. AWS Connector for GitHub ダイアログボックスで、プロンプトが表示されたら、GitHub アカウント名を選択します。

  iv. AWS Connector for GitHub を承認するように求められたら、AWS 接続の承認を選択します。

  v. Install AWS Connector for GitHub ダイアログボックスで、インストールを選択します。

  アカウント名は、選択した GitHub アカウント/組織として表示されます。アカウントでリポジトリを選択できるようになりました。

  vi. リポジトリ で、作成したサンプルリポジトリ を選択しますpython-hello。Branch では、リポジトリのデフォルトのブランチ名 (main など) を選択します。

  vii. Source ディレクトリはデフォルト値のままにします。ディレクトリのデフォルトはリポジトリルートです。前の前提条件ステップのリポジトリルートディレクトリにソースコードを保存しました。

- Bitbucket の場合は、以下の手順に従います。

  i. 次の画面で、接続名を入力します。

  ii. App Runner で Bitbucket を初めて使用する場合は、別の をインストールする を選択します。

  iii. AWS CodeStar リクエストアクセスダイアログボックスで、ワークスペースを選択し、Bitbucket 統合 AWS CodeStar のために へのアクセスを許可できます。ワークスペースを選択し、アクセス権の付与を選択します。

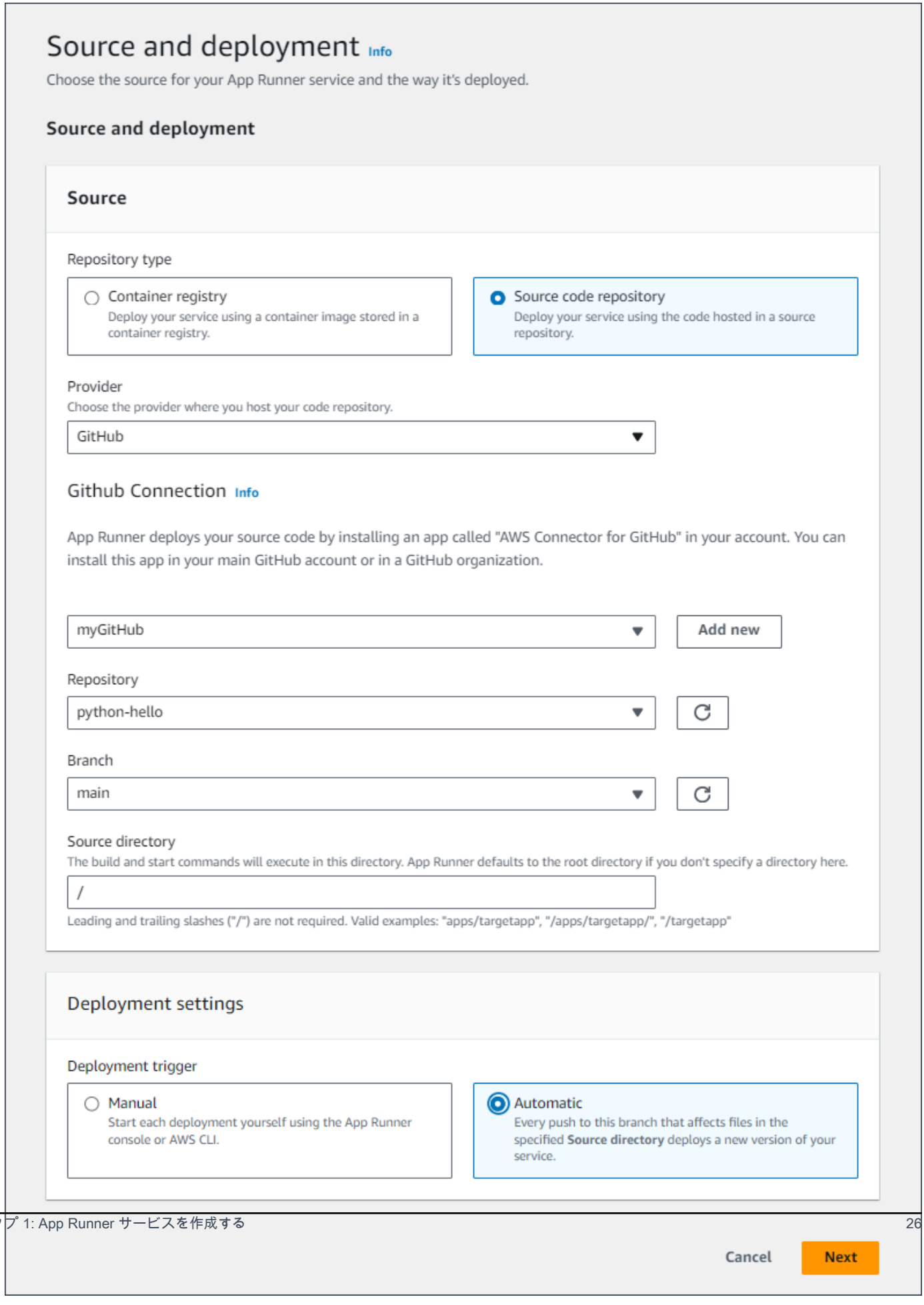
iv. 次に、 AWS コンソールにリダイレクトされます。Bitbucket アプリケーションが正しい Bitbucket ワークスペースに設定されていることを確認し、次へを選択します。

v. リポジトリ で、作成したサンプルリポジトリ を選択しますpython-hello。Branch では、リポジトリのデフォルトのブランチ名 (main など) を選択します。

vi. Source ディレクトリはデフォルト値のままにします。ディレクトリのデフォルトはリポジトリルートです。前の前提条件ステップのリポジトリルートディレクトリにソースコードを保存しました。

2. デプロイを設定する: デプロイ設定セクションで、自動 を選択し、次へ を選択します。

> **Note**
>
> 自動デプロイでは、リポジトリソースディレクトリへの新しいコミットごとに、サービスの新しいバージョンが自動的にデプロイされます。


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source and deployment
Source
Repository type
Container registry
Deploy your service using a container image stored in a container registry.
Source code repository
Deploy your service using the code hosted in a source repository.
Provider
Choose the provider where you host your code repository.
GitHub
Github Connection Info
App Runner deploys your source code by installing an app called "AWS Connector for GitHub" in your account. You can install this app in your main GitHub account or in a GitHub organization.
myGitHub
Add new
Repository
python-hello
Branch
main
Source directory
The build and start commands will execute in this directory. App Runner defaults to the root directory if you don't specify a directory here.
/
Leading and trailing slashes ("/") are not required. Valid examples: "apps/targetapp", "/apps/targetapp/", "/targetapp"
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
Every push to this branch that affects files in the specified Source directory deploys a new version of your service.
Cancel
Next

3. アプリケーションビルドを設定します。

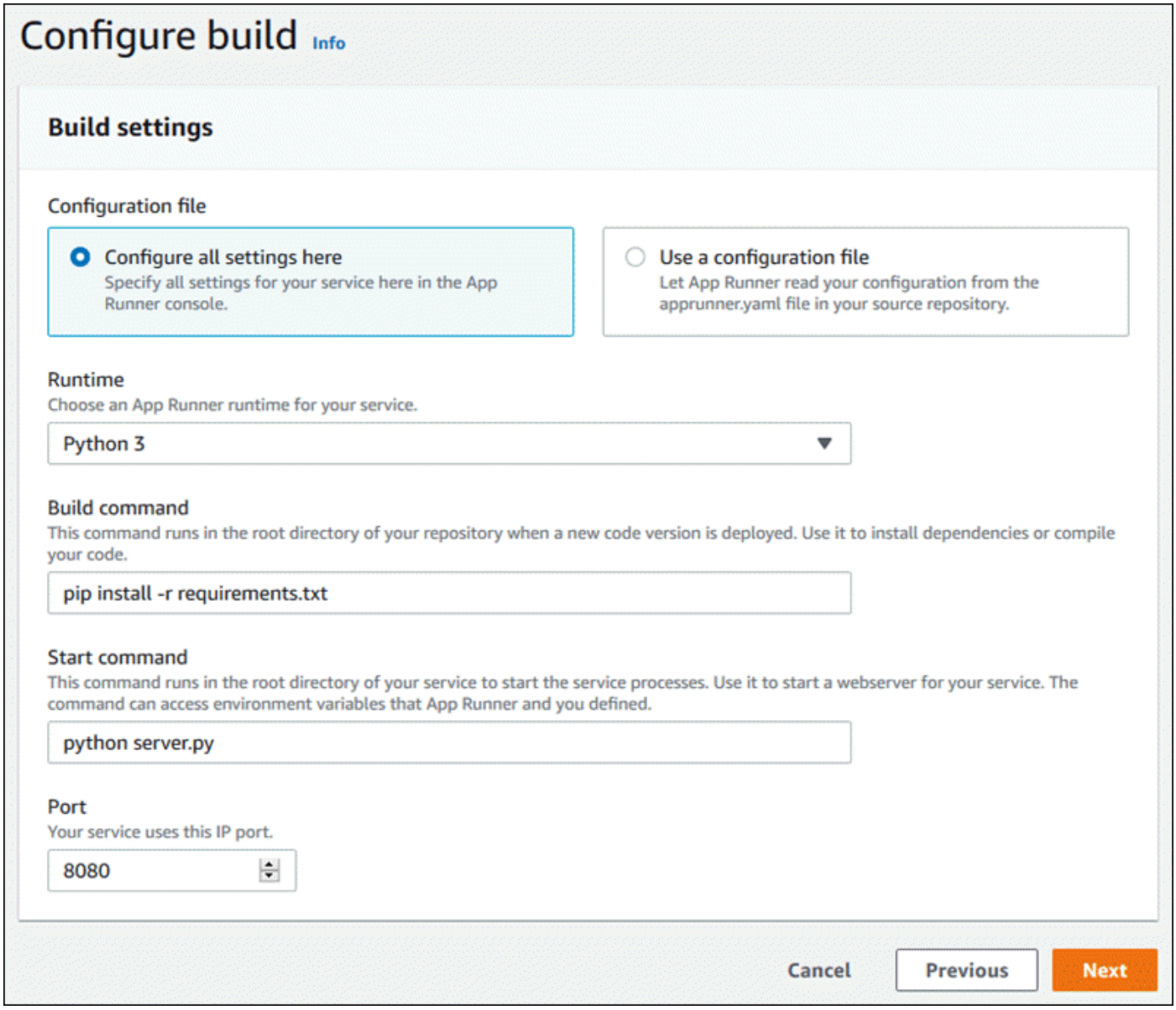
   a. 「ビルドの設定」ページの「設定ファイル」で、「ここですべての設定を設定する」を選択します。

   b. 次のビルド設定を指定します。

      - ランタイム – Python 3 を選択します。
      - ビルドコマンド – と入力します**`pip install -r requirements.txt`**。
      - Start command – と入力します**`python server.py`**。
      - ポート – と入力します**`8080`**。

   c. [次へ] を選択します。

> **Note**
>
> Python 3 ランタイムは、基本 Python 3 イメージとサンプル Python コードを使用して Docker イメージを構築します。次に、このイメージのコンテナインスタンスを実行するサービスを起動します。



4. サービスを設定します。

   a. 「サービスの設定」ページの「サービス設定」セクションに、サービス名を入力します。

   b. 環境変数 で、環境変数の追加 を選択します。環境変数に次の値を指定します。

      - ソース – プレーンテキストを選択する
      - 環境変数名 – **NAME**
      - 環境変数値 – 任意の名前 (名など) 。

> **Note**
>
> サンプルアプリケーションは、この環境変数で設定した名前を読み取り、ウェブページに名前を表示します。

c. [次へ] を選択します。

Configure service Info

Service settings

Service name

python-test

Virtual CPU & memory

1 vCPU | 2 GB

Environment variables — *optional* Info

Add environment variables in plain text or reference them from Secrets Manager and SSM Parameter Store. Update IAM Policies using the IAM Policy template given below to securely reference secrets and configurations as environment variables.

No environment variables have been configured.

Add environment variable

You can add up to 50 more items.

▶ IAM policy templates

▶ Auto scaling Info

Configure automatic scaling behavior.

▶ Health check Info

Configure load balancer health checks.

▶ Security Info

Specify an Instance role and an AWS KMS encryption key

▶ Networking Info

Configure the way your service communicates with other applications, services, and resources.

▶ Observability

Configure observability tooling.

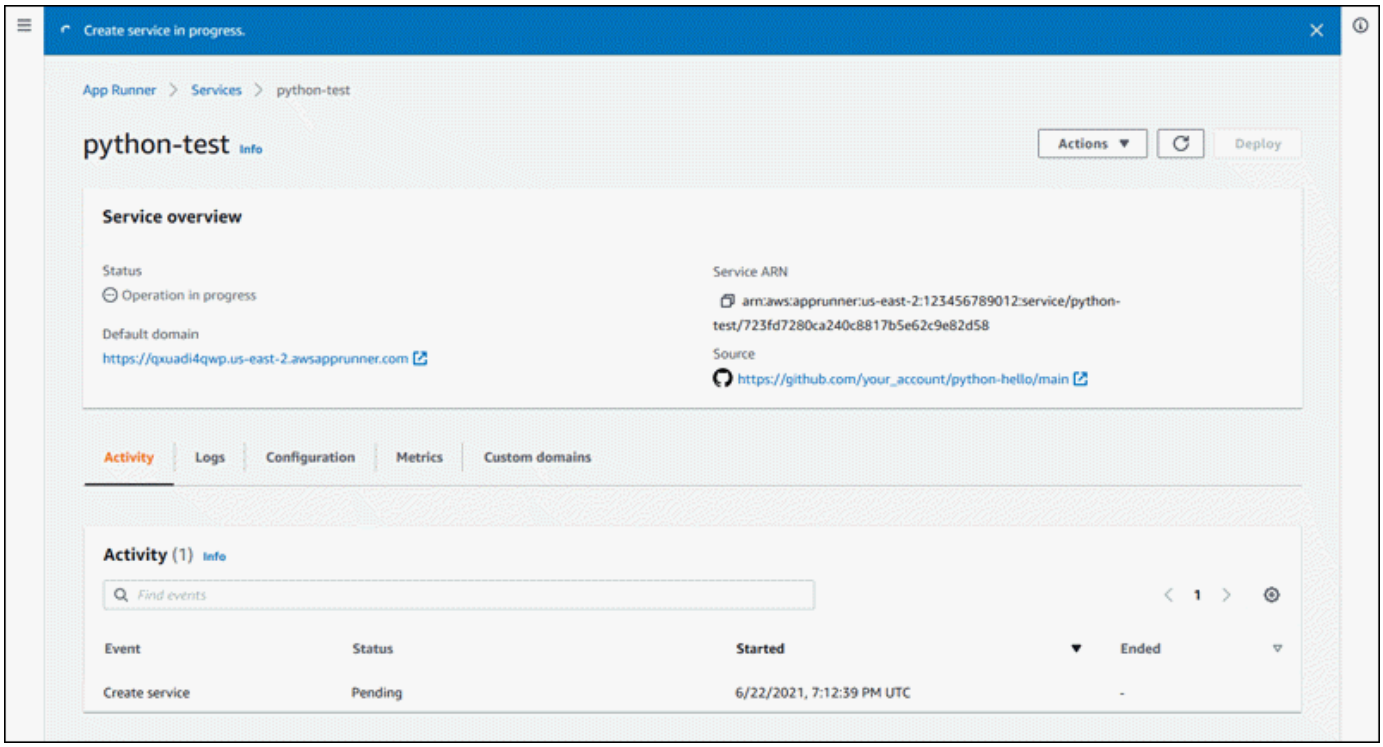
▼ Tags Info

Use tags to search and filter your resources, track your AWS costs, and control access permissions.

Tags — *optional*

5. 確認と作成ページで、入力したすべての詳細を確認し、作成とデプロイを選択します。

   サービスが正常に作成されると、コンソールにサービスダッシュボードが表示され、新しいサービスの概要が表示されます。



6. サービスが実行されていることを確認します。

   a. サービスダッシュボードページで、サービスのステータスが実行中になるまで待ちます。

   b. デフォルトのドメイン値を選択します。これはサービスのウェブサイトへの URL です。

> **Note**
>
> App Runner アプリケーションのセキュリティを強化するために、*.awsapprunner.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化するために、App Runner アプリケーションのデフォルトのドメイン名に機密 Cookie を設定する必要がある場合は、`__Host-` プレフィックス付きの Cookie を使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

ウェブページが表示されます: Hello, ***your name***!

ステップ 2: サービスコードを変更する

このステップでは、リポジトリソースディレクトリのコードを変更します。App Runner CI/CD 機能は、変更を自動的にビルドしてサービスにデプロイします。

サービスコードを変更するには

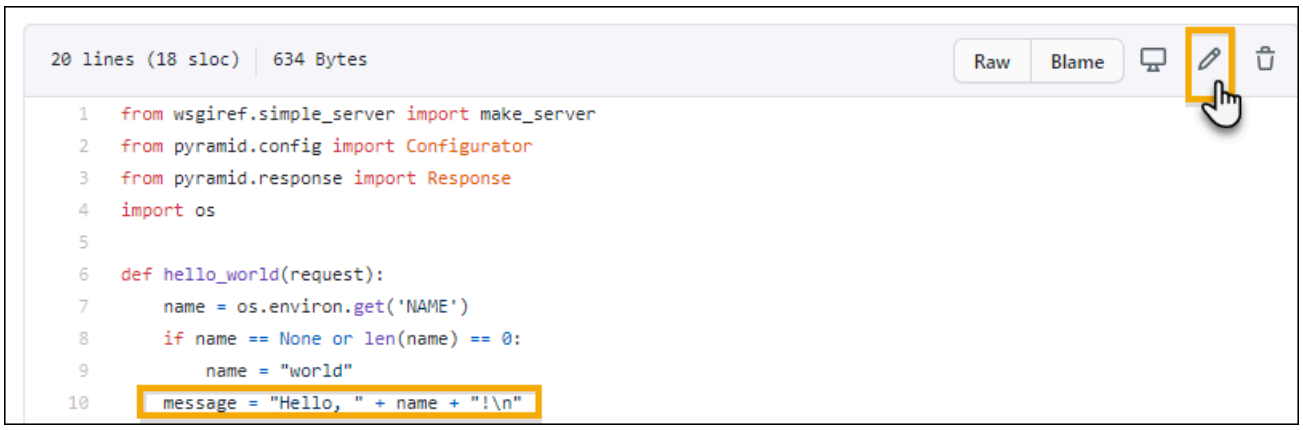
1. サンプルリポジトリに移動します。
2. という名前のファイルを編集しますserver.py。
3. 変数 に割り当てられた式でmessage、テキストを に変更HelloしますGood morning。
4. 変更を保存してリポジトリにコミットします。
5. 次の手順は、GitHub リポジトリのサービスコードの変更を示しています。

    a. サンプル GitHub リポジトリに移動します。

    b. ファイル名server.pyを選択して、そのファイルに移動します。

    c. このファイルの編集 (鉛筆アイコン) を選択します。

    d. 変数 に割り当てられた式でmessage、テキストを に変更HelloしますGood morning。



e. [Commit changes] (変更のコミット) を選択します。

6. 新しいコミットが App Runner サービスのデプロイを開始します。サービスダッシュボードページで、サービスのステータスが進行中のオペレーションに変わります。

   デプロイが終了するのを待ちます。サービスダッシュボードページで、サービスのステータスが実行中に戻ります。

7. デプロイが成功したことを確認する: サービスのウェブページが表示されるブラウザタブを更新します。

   このページには、変更後のメッセージ「おはよう、**##**」が表示されるようになりました。

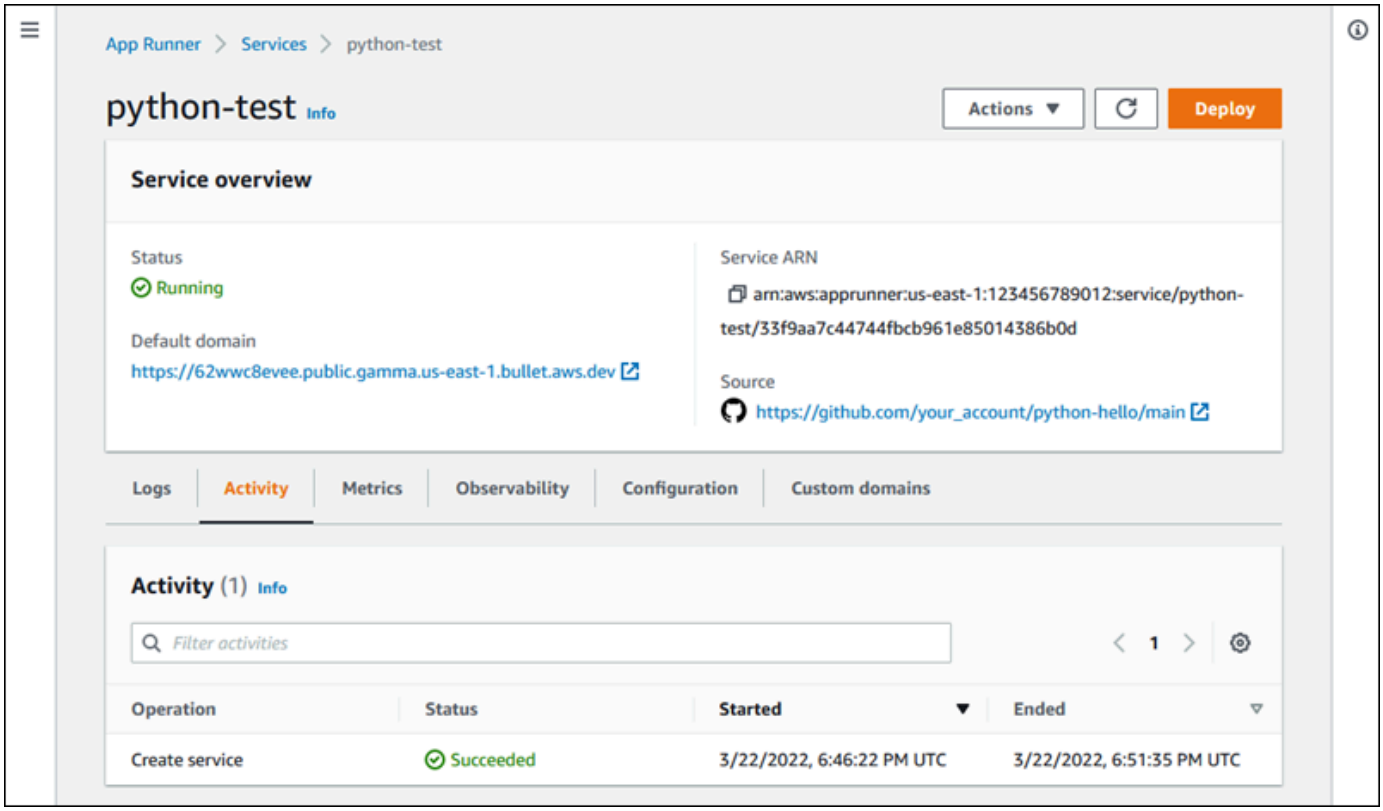
ステップ 3: 設定を変更する

このステップでは、**NAME**環境変数値を変更して、サービス設定の変更を示します。

環境変数の値を変更するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

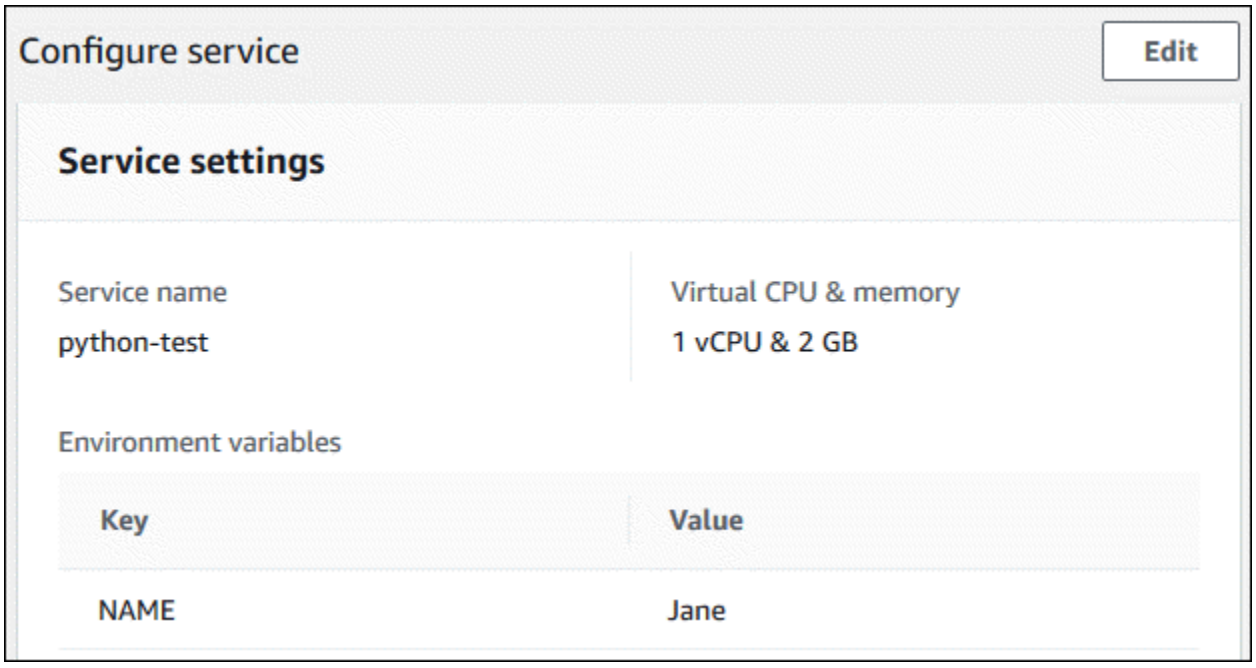
   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. サービスダッシュボードページで、設定タブを選択します。

   コンソールには、サービス設定がいくつかのセクションに表示されます。

4. サービスの設定セクションで、編集を選択します。



5. キー を持つ環境変数の場合**NAME**、値を別の名前に変更します。

6. [Apply changes (変更の適用)] を選択します。

   App Runner は更新プロセスを開始します。サービスダッシュボードページで、サービスのステータスが進行中のオペレーションに変わります。

7. 更新が終了するのを待ちます。サービスダッシュボードページで、サービスのステータスが実行中に戻ります。

8. 更新が成功したことを確認します。サービスのウェブページが表示されるブラウザタブを更新します。

   このページに、変更後の名前「おはよう、**#####**」が表示されるようになりました。

ステップ 4: サービスのログを表示する

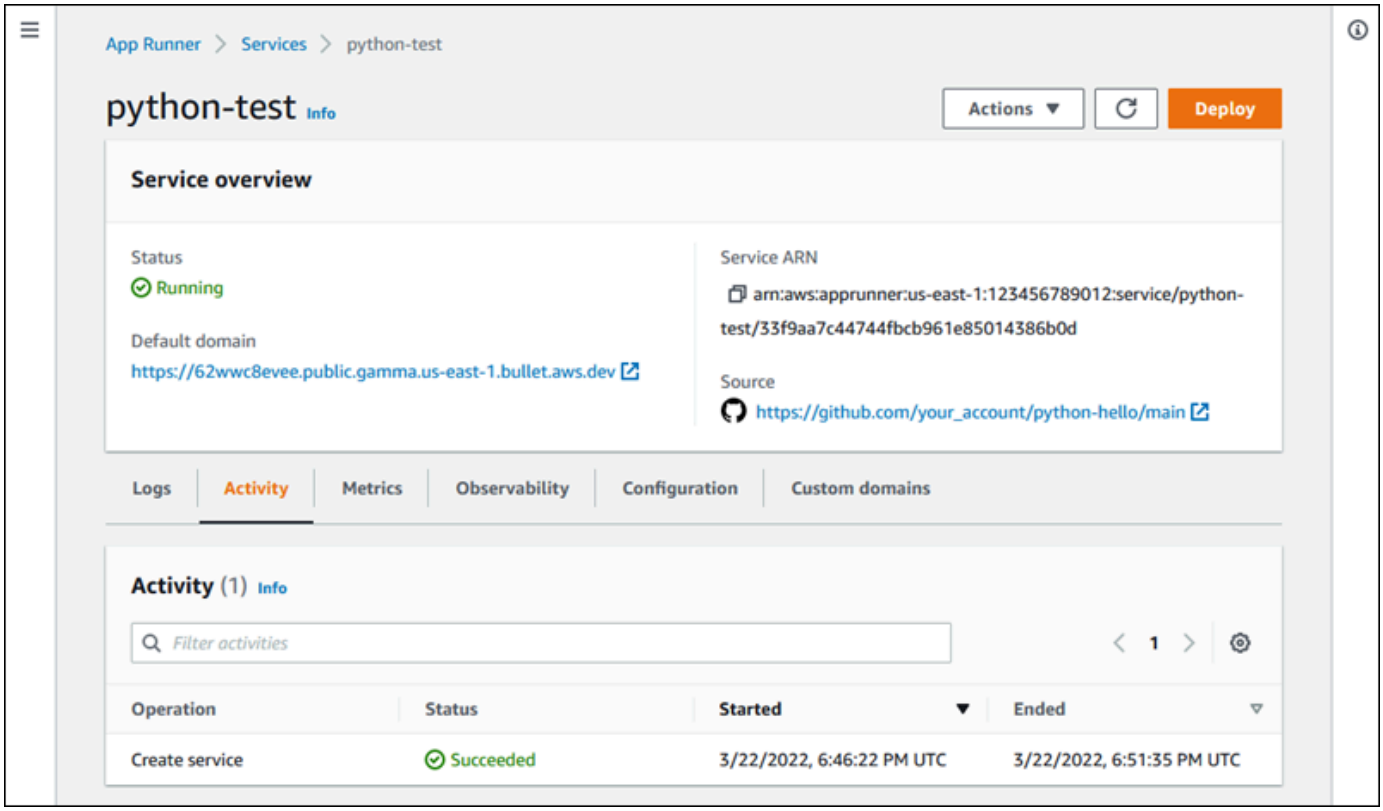
このステップでは、App Runner コンソールを使用して App Runner サービスのログを表示します。App Runner はログを Amazon CloudWatch Logs (CloudWatch Logs) にストリーミングし、サービスのダッシュボードに表示します。App Runner ログの詳細については、「」を参照してくださいthe section called “ログ (CloudWatch Logs)”。

サービスのログを表示するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。

2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

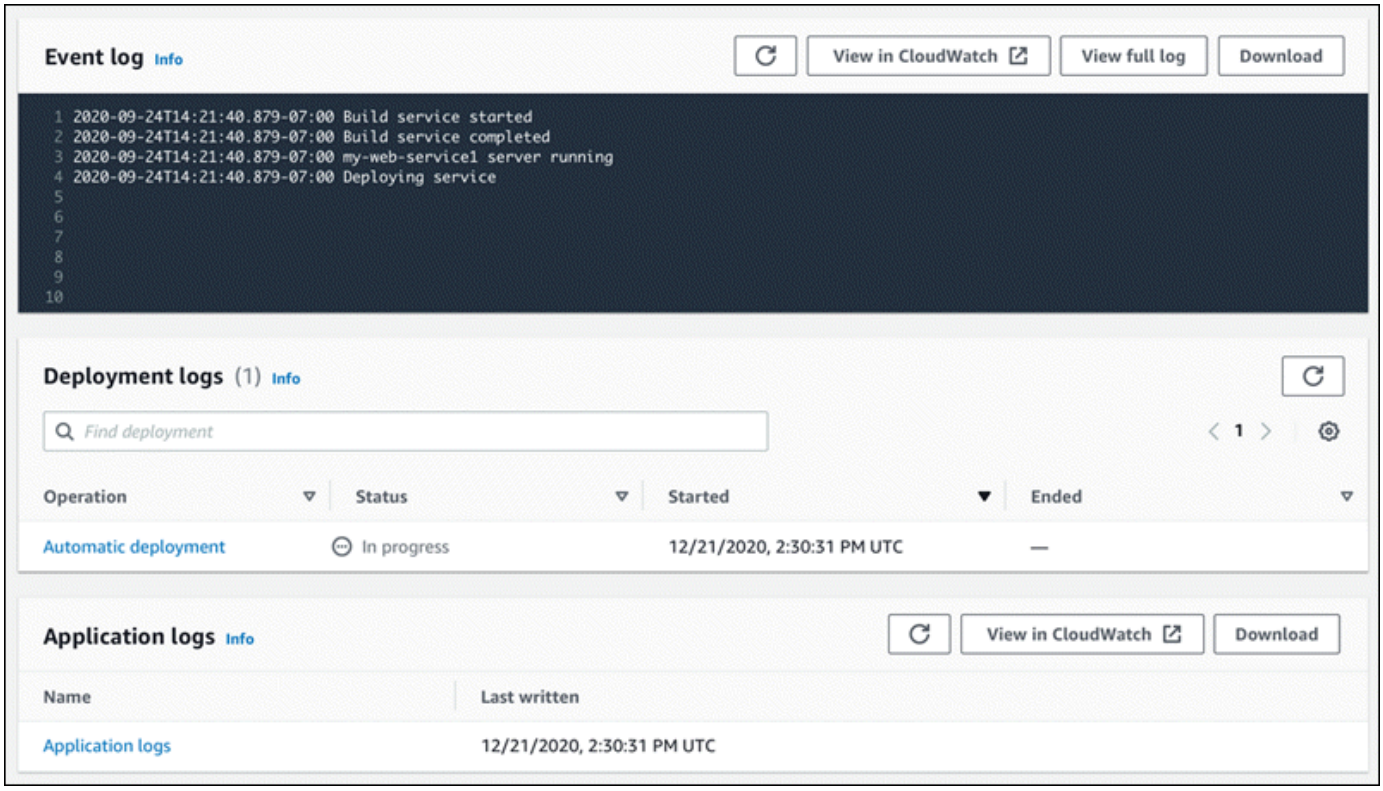
   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. サービスダッシュボードページで、ログタブを選択します。

   コンソールには、いくつかのセクションにいくつかのタイプのログが表示されます。

   - イベントログ – App Runner サービスのライフサイクルにおけるアクティビティ。コンソールに最新のイベントが表示されます。
   - デプロイログ – App Runner サービスへのソースリポジトリのデプロイ。コンソールには、デプロイごとに個別のログストリームが表示されます。
   - アプリケーションログ – App Runner サービスにデプロイされたウェブアプリケーションの出力。コンソールは、実行中のすべてのインスタンスからの出力を 1 つのログストリームに結合します。



4. 特定のデプロイを検索するには、検索語を入力してデプロイログリストの範囲を絞り込みます。テーブルに表示される任意の値を検索できます。
5. ログの内容を表示するには、フルログの表示 (イベントログ) またはログストリーム名 (デプロイログとアプリケーションログ) を選択します。
6. ダウンロードを選択してログをダウンロードします。デプロイログストリームの場合は、まずログストリームを選択します。
7. CloudWatch で表示 を選択して CloudWatch コンソールを開き、その完全な機能を使用して App Runner サービスログを調べます。デプロイログストリームの場合は、まずログストリームを選択します。

> **Note**
>
> CloudWatch コンソールは、結合されたアプリケーションログではなく、特定のインスタンスのアプリケーションログを表示する場合に特に便利です。

ステップ 5：クリーンアップ

これで、App Runner サービスの作成、ログの表示、およびいくつかの変更を行う方法を学習しました。このステップでは、サービスを削除して、不要になったリソースを削除します。

サービスを削除するには

1. サービスダッシュボードページで、アクションを選択し、サービスの削除を選択します。
2. 確認ダイアログで、リクエストされたテキストを入力し、削除を選択します。

   結果: コンソールはサービスページに移動します。削除したサービスのステータスが DELETING と表示されます。しばらくすると、リストから消えます。

このチュートリアルの一部として作成した GitHub および Bitbucket 接続の削除も検討してください。詳細については、「the section called “Connections”」を参照してください。

次のステップ

最初の App Runner サービスをデプロイしたので、以下のトピックで詳細を確認してください。

- アーキテクチャと概念 – App Runner に関連するアーキテクチャ、主要な概念、 AWS リソース。
- イメージベースのサービス および コードベースのサービス – App Runner がデプロイできる 2 種類のアプリケーションソース。
- App Runner の開発 – App Runner にデプロイするアプリケーションコードを開発または移行するときに知っておくべきこと。
- App Runner コンソール – App Runner コンソールを使用してサービスを管理およびモニタリングします。
- サービスの管理 – App Runner サービスのライフサイクルを管理します。
- オブザーバビリティ – メトリクスのモニタリング、ログの読み取り、イベントの処理、サービスアクション呼び出しの追跡、HTTP 呼び出しなどのアプリケーションイベントのトレースを行うことで、App Runner サービスオペレーションを可視化します。
- App Runner 設定ファイル – App Runner サービスのビルドおよびランタイム動作のオプションを指定する設定ベースの方法です。
- App Runner API – App Runner アプリケーションプログラミングインターフェイス (API) を使用して、App Runner リソースを作成、読み取り、更新、削除します。

- セキュリティ – App Runner やその他の サービスの使用中に と がクラウドセキュリティ AWS を確保するさまざまな方法。

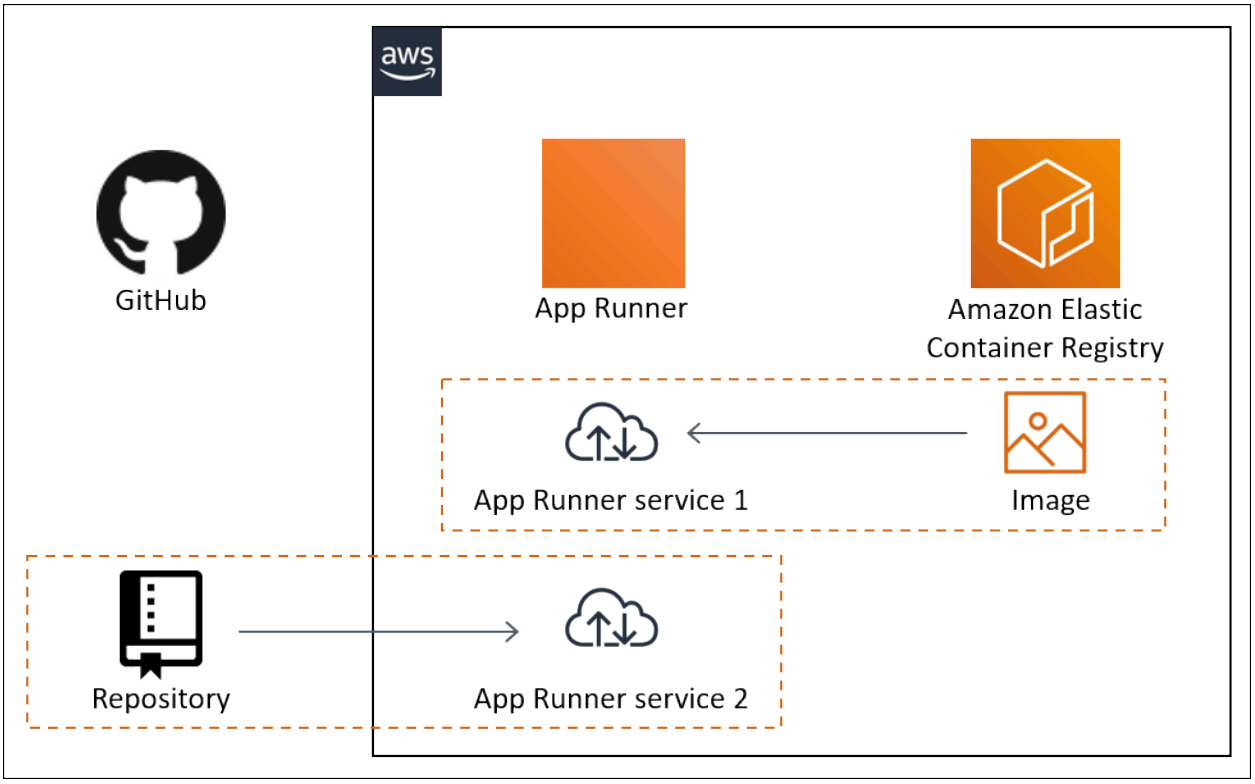
App Runner のアーキテクチャと概念

AWS App Runner は、リポジトリからソースコードまたはソースイメージを取得し、 で実行中のウェブサービスを作成して維持します AWS クラウド。通常、サービスを作成するには、App Runner アクション CreateService を 1 つだけ呼び出す必要があります。

ソースイメージリポジトリでは、App Runner がウェブサービスを実行するためにデプロイできる ready-to-useコンテナイメージを提供します。ソースコードリポジトリでは、ウェブサービスを構築して実行するためのコードと手順を提供し、特定のランタイム環境をターゲットにします。App Runner は、複数のプログラミングプラットフォームをサポートしています。各プラットフォームには、プラットフォームメジャーバージョンの 1 つ以上のマネージドランタイムがあります。

現時点では、App Runner は Bitbucket または GitHub リポジトリからソースコードを取得することも、 の Amazon Elastic Container Registry (Amazon ECR) からソースイメージを取得することもできます AWS アカウント。

次の図は、App Runner サービスアーキテクチャの概要を示しています。この図では、2 つのサンプルサービスがあります。1 つは GitHub からソースコードをデプロイし、もう 1 つは Amazon ECR からソースイメージをデプロイします。Bitbucket リポジトリにも同じフローが適用されます。



App Runner の概念

App Runner で実行されているウェブサービスに関連する主要な概念を次に示します。

- App Runner サービス – App Runner がソースコードリポジトリまたはコンテナイメージに基づいてアプリケーションをデプロイおよび管理するために使用する AWS リソース。App Runner サービスは、実行中のアプリケーションのバージョンです。サービスの作成の詳細については、「」を参照してくださいthe section called “作成”。
- ソースタイプ – App Runner サービスをデプロイするために指定するソースリポジトリのタイプ: ソースコードまたはソースイメージ。
- リポジトリプロバイダー – アプリケーションソースを含むリポジトリサービス (GitHub、Bitbucket、Amazon ECR など)。
- App Runner 接続 – App Runner がリポジトリプロバイダーアカウント (GitHub アカウントや組織など) にアクセスできるようにする AWS リソース。接続の詳細については、「the section called “Connections”」を参照してください。
- ランタイム – ソースコードリポジトリをデプロイするためのベースイメージ。App Runner は、さまざまなプログラミングプラットフォームとバージョンに対してさまざまなマネージドランタイムを提供します。詳細については、「コードベースのサービス」を参照してください。
- デプロイ — ソースリポジトリ (コードまたはイメージ) のバージョンを App Runner サービスに適用するアクション。サービスへの最初のデプロイは、サービスの作成の一環として行われます。後のデプロイは、次の 2 つの方法のいずれかで行うことができます。
  - 自動デプロイ — CI/CD 機能。App Runner サービスを設定して、リポジトリに表示されるアプリケーションの各バージョンを自動的にビルド (ソースコード用) してデプロイできます。これは、ソースコードリポジトリの新しいコミットでも、ソースイメージリポジトリの新しいイメージバージョンでもかまいません。
  - 手動デプロイ — 明示的に開始する App Runner サービスへのデプロイ。
- カスタムドメイン – App Runner サービスに関連付けるドメイン。ウェブアプリケーションのユーザーは、このドメインを使用して、デフォルトの App Runner サブドメインの代わりにウェブサービスにアクセスできます。詳細については、「the section called “カスタムドメイン名”」を参照してください。

> **Note**
>
> App Runner アプリケーションのセキュリティを強化するために、*.awsapprunner.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化

するために、App Runner アプリケーションのデフォルトのドメイン名に機密 Cookie を設定する必要がある場合は、__Host-プレフィックス付きの Cookie を使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

- メンテナンス – App Runner サービスを実行するインフラストラクチャで App Runner が時折実行するアクティビティ。メンテナンスが進行中の場合、サービスステータスは数分間一時的に OPERATION_IN_PROGRESS (コンソールで進行中のオペレーション) に変わります。この間、サービスに対するアクション (デプロイ、設定の更新、一時停止/再開、削除など) はブロックされます。数分後、サービスステータスが に戻ったら、アクションを再試行してくださいRUNNING。

  > **Note**
  >
  > アクションが失敗しても、App Runner サービスが停止しているわけではありません。アプリケーションはアクティブであり、リクエストの処理を継続します。サービスでダウンタイムが発生する可能性はほとんどありません。

  特に、App Runner は、サービスをホストしている基盤となるハードウェアの問題を検出すると、サービスを移行します。サービスのダウンタイムを防ぐために、App Runner は新しいインスタンスセットにサービスをデプロイし、トラフィックをそれらのインスタンスにシフトします (ブルー/グリーンデプロイ) 。料金が一時的にわずかに増加することがあります。

App Runner でサポートされている設定

App Runner サービスを設定するときは、サービスに割り当てる仮想 CPU とメモリの設定を指定します。選択したコンピューティング設定に基づいて料金が発生します。料金の詳細については、AWS Resource Groups 料金表を参照してください。

次の表は、App Runner がサポートする vCPU とメモリの設定に関する情報を示しています。

CPU	メモリ
0.25 vCPU	0.5 GB
0.25 vCPU	1 GB

CPU	メモリ
0.5 vCPU	1 GB
1 vCPU	2 GB
1 vCPU	3 GB
1 vCPU	4 GB
2 vCPU	4 GB
2 vCPU	6 GB
4 vCPU	8 GB
4 vCPU	10 GB
4 vCPU	12 GB

App Runner リソース

App Runner を使用する場合は、 でいくつかのタイプのリソースを作成および管理します AWS アカウント。これらのリソースは、コードにアクセスしてサービスを管理するために使用されます。

次の表に、これらのリソースの概要を示します。

リソース名	説明
Service	実行中のアプリケーションのバージョンを表します。このガイドの残りの部分では、サービスタイプ、管理、設定、モニタリングについて説明します。 ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`*` :service/`*`service-name`*` [/`*`service-id`*` ]`
Connection	App Runner サービスに、サードパーティープロバイダーに保存されているプライベートリポジトリへのアクセスを提供します。複数のサービス間で共有するための個別のリソースとして存在します。接続の詳

リソース名	説明
	細については、「the section called “Connections”」を参照してください。 ARN: arn:aws:apprunner: *region*:*account-id* :connection/ *connection-name* [/*connection-id*]
AutoScalingConfiguration	App Runner サービスに、アプリケーションの自動スケーリングを制御する設定を提供します。複数のサービス間で共有するための個別のリソースとして存在します。 Auto Scaling の詳細については、「the section called “Auto scaling”」を参照してください。 ARN: arn:aws:apprunner: *region*:*account-id* :autoscalingconfiguration/ *config-name* [/*config-revision* [/*config-id*]]
ObservabilityConfiguration	App Runner サービスの追加のアプリケーションオブザーバビリティ機能を設定します。複数のサービス間で共有するための個別のリソースとして存在します。オブザーバビリティ設定の詳細については、「」を参照してくださいthe section called “オブザーバビリティ設定”。 ARN: arn:aws:apprunner: *region*:*account-id* :observabilityconfiguration/ *config-name* [/*config-revision* [/*config-id*]]
VpcConnector	App Runner サービスの VPC 設定を構成します。複数のサービス間で共有するための個別のリソースとして存在します。VPC 機能の詳細については、「」を参照してくださいthe section called “送信トラフィック”。 ARN: arn:aws:apprunner: *region*:*account-id* :vpcconnector/ *connector-name* [/*connector-revision* [/*connector-id*]]

リソース名	説明
VpcIngressConnection	これは、受信トラフィックを設定するために使用される AWS App Runner リソースです。VPC インターフェイスエンドポイントと App Runner サービス間の接続を確立して、Amazon VPC 内からのみ App Runner サービスにアクセスできるようにします。VPCIngressConnection の機能の詳細については、「」を参照してくださいthe section called “プライベートエンドポイントを有効にする”。 ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`*` :vpcingressconnection/` *`vpc-ingress-connection-name`* ` [/`*`connector-id`*` ]]`

App Runner リソースクォータ

AWS は、各 で AWS リソースを使用するためのクォータ (制限とも呼ばれます) をアカウントに課します AWS リージョン。次の表に、App Runner リソースに関連するクォータを示します。クォータは、 のAWS App Runner エンドポイントとクォータにも一覧表示されますAWS 全般のリファレンス。

リソースクォータ		説明	デフォルト値:	調整可能か?
Services		アカウントで作成できるサービスの最大数 AWS リージョン。	30	✓ はい
Connections		各アカウントで作成できる接続の最大数 AWS リージョン。単一の接続を複数のサービスで使用できます。	10	✓ はい
Auto scaling configurations	名前	各アカウントで作成する Auto Scaling 設定で指定できる一意の名前の最大数 AWS リージョン。単一の Auto Scaling 設定を複数のサービスで使用きます。	10	✓ はい
	名前あたりの	AWS リージョン 一意の名前ごとにアカウントで作成できる Auto Scaling 設定リビジョンの最	5	Xいいえ

リソースクォータ		説明	デフォルト値:	調整可能か?
	リビジョン	大数。1 つの Auto Scaling 設定リビジョンを複数のサービスで使用できます。		
Observability configurations	名前	各アカウントで作成するオブザーバビリティ設定で保持できる一意の名前の最大数 AWS リージョン。単一の可観測性設定を複数のサービスで使用できます。	10	✓ はい
	名前あたりのリビジョン	AWS リージョン 一意の名前ごとにアカウントで作成できるオブザーバビリティ設定リビジョンの最大数。1 つのオブザーバビリティ設定リビジョンを複数のサービスで使用できます。	10	×いいえ
VPC connectors		アカウントでそれぞれ作成できる VPC コネクタの最大数 AWS リージョン。単一の VPC コネクタを複数のサービスで使用できます。	10	✓ はい
VPC Ingress Connection		アカウントで作成できる VPC イングレス接続の最大数 AWS リージョン。単一の VPC Ingress Connection を使用して、複数の App Runner サービスにアクセスできます。	1	×いいえ

ほとんどのクォータは調整可能で、クォータの引き上げをリクエストできます。詳細については、Service Quotas ユーザーガイドの「Requesting a quota increase (クォータの引き上げのリクエスト)」を参照してください。

ソースイメージに基づく App Runner サービス

を使用して AWS App Runner 、ソースコードとソースイメージの 2 つの根本的に異なるタイプのサービスソースに基づいてサービスを作成および管理できます。ソースタイプに関係なく、App Runner はサービスの起動、実行、スケーリング、ロードバランシングを処理します。App Runner の CI/CD 機能を使用して、ソースイメージまたはコードの変更を追跡できます。App Runner は変更を検出すると、自動的に (ソースコード用) をビルドし、新しいバージョンを App Runner サービスにデプロイします。

この章では、ソースイメージに基づくサービスについて説明します。ソースコードに基づくサービスの詳細については、「」を参照してくださいコードベースのサービス。

ソースイメージは、イメージリポジトリに保存されているパブリックまたはプライベートコンテナイメージです。App Runner をイメージにポイントすると、このイメージに基づいてコンテナを実行するサービスが開始されます。ビルドステージは必要ありません。むしろ、ready-to-deployできるイメージを提供します。

> **Note**
>
> コンテナイメージを提供する場合、これらのイメージを定期的に更新してパッチを適用する責任があります。App Runner がインフラストラクチャを管理する間は、提供されたコンテナイメージのセキュリティとup-to-dateステータスを確認する必要があります。詳細については、AWS App Runner ドキュメントを参照してください。

イメージリポジトリプロバイダー

App Runner は以下のイメージリポジトリプロバイダーをサポートしています。

- Amazon Elastic Container Registry (Amazon ECR) – プライベートなイメージを に保存します AWS アカウント。
- Amazon Elastic Container Registry Public (Amazon ECR Public) – パブリックに読み取り可能なイメージを保存します。

プロバイダーのユースケース

- AWS アカウントの Amazon ECR に保存されているイメージの使用

- 別の AWS アカウントの Amazon ECR に保存されているイメージの使用
- Amazon ECR Public に保存されているイメージの使用

AWS アカウントの Amazon ECR に保存されているイメージの使用

Amazon ECR はイメージをリポジトリに保存します。プライベートリポジトリとパブリックリポジトリがあります。プライベートリポジトリから App Runner サービスにイメージをデプロイするには、App Runner に Amazon ECR からイメージを読み取るアクセス許可が必要です。App Runner にアクセス許可を付与するには、App Runner にアクセスロールを提供する必要があります。これは、必要な Amazon ECR アクションのアクセス許可を持つ AWS Identity and Access Management (IAM) ロールです。App Runner コンソールを使用してサービスを作成する場合、アカウントで既存のロールを選択できます。または、IAM コンソールを使用して新しいカスタムロールを作成することもできます。または、App Runner コンソールで マネージドポリシーに基づいてロールを作成することもできます。

App Runner API または を使用する場合は AWS CLI、2 ステップのプロセスを完了します。まず、IAM コンソールを使用してアクセスロールを作成します。App Runner が提供する管理ポリシーを使用するか、独自のカスタムアクセス許可を入力できます。次に、CreateService API アクションを使用して、サービスの作成時にアクセスロールを指定します。

App Runner サービスの作成については、「」を参照してくださいthe section called “作成”。

別の AWS アカウントの Amazon ECR に保存されているイメージの使用

App Runner サービスを作成するときは、サービスが存在するアカウント以外の AWS アカウントに属する Amazon ECR リポジトリに保存されているイメージを使用できます。クロスアカウントイメージを使用する場合は、前のセクションで示したものに加えて、同じアカウントイメージに関する考慮事項がいくつかあります。

- クロスアカウントリポジトリには、ポリシーがアタッチされている必要があります。リポジトリポリシーは、アクセスロールにリポジトリ内のイメージを読み取るアクセス許可を付与します。この目的には、次のポリシーを使用します。`Principal` 要素のロールを、アクセスロールの Amazon リソースネーム (ARN) に置き換えます。

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::123456789012:role/MyApplicationRole"},
      "Action": [
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetDownloadUrlForLayer"
        ],
        "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/*"
    }
  ]
}
```

リポジトリポリシーを Amazon ECR リポジトリにアタッチする方法については、「Amazon Elastic Container Registry ユーザーガイド」の「リポジトリポリシーステートメントの設定」を参照してください。

- App Runner は、サービスが存在するアカウントとは異なるアカウントの Amazon ECR イメージの自動デプロイをサポートしていません。

Amazon ECR Public に保存されているイメージの使用

Amazon ECR Public は、パブリックに読み取り可能なイメージを保存します。App Runner サービスのコンテキストで注意すべき Amazon ECR と Amazon ECR Public の主な違いは次のとおりです。

- Amazon ECR Public イメージはパブリックに読み取り可能です。Amazon ECR Public イメージに基づいてサービスを作成するときに、アクセスロールを提供する必要はありません。リポジトリにアタッチされたポリシーは必要ありません。
- App Runner は、Amazon ECR Public イメージの自動 (継続的な) デプロイをサポートしていません。

Amazon ECR Public から直接サービスを起動する

App Runner で実行されているウェブサービスとして、Amazon ECR Public Gallery でホストされている互換性のあるウェブアプリケーションのコンテナイメージを直接起動できます。ギャラリーを参照するときは、ギャラリーページで Launch with App Runner を探してイメージを探します。このオプションを使用するイメージは App Runner と互換性があります。ギャラリーの詳細については、Amazon ECR Public ユーザーガイドの「Amazon ECR Public Gallery の使用」を参照してください。

App Runner サービスとしてギャラリーイメージを起動するには

1. イメージのギャラリーページで、App Runner で起動を選択します。

   結果: App Runner コンソールが新しいブラウザタブで開きます。コンソールにサービスの作成ウィザードが表示され、必要な新しいサービスの詳細のほとんどが事前に入力されています。
2. コンソールに表示されるリージョン以外の AWS リージョンでサービスを作成する場合は、コンソールヘッダーに表示されるリージョンを選択します。次に、別のリージョンを選択します。
3. ポートには、イメージアプリケーションがリッスンするポート番号を入力します。通常、イメージのギャラリーページで確認できます。
4. 必要に応じて、その他の設定の詳細を変更します。
5. 次へを選択し、設定を確認してから、作成とデプロイを選択します。

イメージの例

App Runner チームは、hello-app-runner サンプルイメージを Amazon ECR Public Gallery に保持します。この例を使用して、イメージベースの App Runner サービスの作成を開始できます。詳細については、「hello-app-runner」を参照してください。

ソースコードに基づく App Runner サービス

を使用して AWS App Runner 、ソースコードとソースイメージの 2 つの異なるタイプのサービスソースに基づいてサービスを作成および管理できます。ソースタイプに関係なく、App Runner はサービスの起動、実行、スケーリング、ロードバランシングを処理します。App Runner の CI/CD 機能を使用して、ソースイメージまたはコードの変更を追跡できます。App Runner は変更を検出すると、自動的に (ソースコード用) をビルドし、新しいバージョンを App Runner サービスにデプロイします。

この章では、ソースコードに基づく のサービスについて説明します。ソースイメージに基づくサービスの詳細については、「」を参照してくださいイメージベースのサービス。

ソースコードは、App Runner が構築およびデプロイするアプリケーションコードです。App Runner をコードリポジトリのソースディレクトリにポイントし、プログラミングプラットフォームのバージョンに対応する適切なランタイムを選択します。App Runner は、ランタイムのベースイメージとアプリケーションコードに基づいてイメージを構築します。次に、このイメージに基づいてコンテナを実行するサービスを開始します。

App Runner は、プラットフォーム固有の便利なマネージドランタイムを提供します。これらのランタイムはそれぞれ、ソースコードからコンテナイメージを構築し、言語ランタイムの依存関係をイメージに追加します。Dockerfile などのコンテナ設定やビルド手順を指定する必要はありません。

この章のサブトピックでは、App Runner がサポートするさまざまなプラットフォームについて説明します。これは、さまざまなプログラミング環境とバージョンにマネージドランタイムを提供するマネージドプラットフォームです。

トピック

- ソースコードリポジトリプロバイダー
- ソースディレクトリ
- App Runner マネージドプラットフォーム
- マネージドランタイムバージョンのサポート終了
- マネージドランタイムバージョンと App Runner ビルド
- Python プラットフォームを使用する
- Node.js プラットフォームを使用する
- Java プラットフォームの使用

- .NET プラットフォームを使用する
- PHP プラットフォームを使用する
- Ruby プラットフォームを使用する
- Go プラットフォームを使用する

ソースコードリポジトリプロバイダー

App Runner は、ソースコードリポジトリから読み取ってソースコードをデプロイします。App Runner はGitHub と Bitbucket の 2 つのソースコードリポジトリプロバイダーをサポートしています。

ソースコードリポジトリプロバイダーからのデプロイ

ソースコードリポジトリから App Runner サービスにソースコードをデプロイするために、App Runner はそれへの接続を確立します。App Runner コンソールを使用してサービスを作成するときは、App Runner がソースコードをデプロイするための接続の詳細とソースディレクトリを指定します。

Connections

サービス作成手順の一部として接続の詳細を指定します。App Runner API または を使用する場合 AWS CLI、接続は別のリソースです。まず、CreateConnection API アクションを使用して接続を作成します。次に、CreateService API アクションを使用して、サービスの作成時に接続の ARN を指定します。

ソースディレクトリ

サービスを作成するときは、ソースディレクトリも指定します。デフォルトでは、App Runner はリポジトリのルートディレクトリをソースディレクトリとして使用します。ソースディレクトリは、アプリケーションのソースコードと設定ファイルを保存するソースコードリポジトリ内の場所です。ビルドコマンドとスタートコマンドは、ソースディレクトリからも実行されます。App Runner API または を使用してサービス AWS CLI を作成または更新する場合は、CreateService および UpdateService API アクションでソースディレクトリを指定します。詳細については、次の「ソースディレクトリ」セクションを参照してください。

App Runner サービスの作成の詳細については、「」を参照してくださいthe section called “作成”。App Runner 接続の詳細については、「」を参照してくださいthe section called “Connections”。

ソースディレクトリ

App Runner サービスを作成するときは、リポジトリとブランチとともにソースディレクトリを指定できます。Source ディレクトリフィールドの値を、アプリケーションのソースコードと設定ファイルを保存するリポジトリディレクトリパスに設定します。App Runner は、指定したソースディレクトリパスからビルドコマンドとスタートコマンドを実行します。

ルートリポジトリディレクトリからソースディレクトリパスの値を絶対として入力します。値を指定しない場合、デフォルトでリポジトリの最上位ディレクトリになります。これはリポジトリルートディレクトリとも呼ばれます。

また、最上位リポジトリディレクトリの他に異なるソースディレクトリパスを指定することもできます。これにより、モノレポリポジトリアーキテクチャがサポートされます。つまり、複数のアプリケーションのソースコードは 1 つのリポジトリに保存されます。単一のモノレポから複数の App Runner サービスを作成してサポートするには、各サービスを作成するときに異なるソースディレクトリを指定します。

> **Note**
>
> 複数の App Runner サービスに同じソースディレクトリを指定すると、両方のサービスが個別にデプロイおよび動作します。

`apprunner.yaml` 設定ファイルを使用してサービスパラメータを定義する場合は、リポジトリのソースディレクトリフォルダに配置します。

デプロイトリガーオプションが自動に設定されている場合、ソースディレクトリでコミットした変更は自動デプロイをトリガーします。ソースディレクトリパスの変更のみが自動デプロイをトリガーします。ソースディレクトリの場所が自動デプロイの範囲にどのように影響するかを理解することが重要です。詳細については、「」の「自動デプロイ」を参照してくださいデプロイ方法。

> **Note**
>
> App Runner サービスが PHP マネージドランタイムを使用していて、デフォルトのルートリポジトリ以外のソースディレクトリを指定する場合は、正しい PHP ランタイムバージョンを使用することが重要です。詳細については、「PHP プラットフォームを使用する」を参照してください。

App Runner マネージドプラットフォーム

App Runner マネージドプラットフォームは、さまざまなプログラミング環境にマネージドランタイムを提供します。各マネージドランタイムを使用すると、プログラミング言語またはランタイム環境のバージョンに基づいてコンテナを簡単に構築して実行できます。マネージドランタイムを使用すると、App Runner はマネージドランタイムイメージで始まります。このイメージは Amazon Linux Docker イメージに基づいており、言語ランタイムパッケージ、いくつかのツール、一般的な依存関係パッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで `runtime`キーワードを使用します。マネージドランタイムの命名規則は *`<language-name><major-version>`* です。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションで特定のバージョンのマネージドランタイムが必要な場合は、App Runner 設定ファイルの `runtime-version`キーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

マネージドランタイムバージョンのサポート終了

マネージド言語ランタイムの公式プロバイダーまたはコミュニティがバージョンをサポート終了 (EOL) と公式に宣言すると、App Runner はその後、バージョンステータスをサポート終了と宣言します。サポート終了に達したマネージド言語ランタイムバージョンでサービスが実行されている場合、以下のポリシーと推奨事項が適用されます。

言語ランタイムバージョンのサポート終了:

- 既存のサービスは、サポート終了に達したランタイムを使用しても、引き続きトラフィックを実行し、処理します。ただし、更新、セキュリティパッチ、またはテクニカルサポートを受け取らなくなったサポートされていないランタイムで実行されます。
- サポート終了ランタイムを使用する既存のサービスの更新は引き続き許可されますが、サービスのサポート終了ランタイムを引き続き使用することはお勧めしません。
- サポート終了日に達したランタイムを使用して新しいサービスを作成することはできません。

サポート終了ステータスの言語ランタイムバージョンに必要なアクション:

- サービスがソースイメージに基づいている場合、そのサービスのユーザー側でそれ以上のアクションは必要ありません。
- サービスがソースコードに基づいている場合は、サポートされているランタイムバージョンを使用するようにサービス設定を更新します。これを行うには、App Runner コンソールでサポートされているランタイムバージョンを選択するか、apprunner.yaml 設定ファイルの `runtime`フィールドを更新するか、CreateService/UpdateService API オペレーションまたは IaC ツールを使用して `runtime`パラメータを設定します。サポートされているランタイムのリストについては、この章の「特定のランタイムのリリース情報」ページを参照してください。
- または、App Runner のコンテナイメージソースオプションに切り替えることもできます。詳細については、イメージベースのサービスを参照してください。

> **Note**
>
> Node.js 12、14、または 16 から Node.js 22、または Python 3.7 または 3.8 から Python 3.11 に移行する場合は、Node.js 22 および Python 3.11 は、より高速で効率的なビルドを提供する改訂された App Runner ビルドプロセスを使用することに注意してください。アップグレード前に互換性を確保するために、次のセクションのビルドプロセスガイダンスを確認することをお勧めします。

次の表に、サポート終了日が指定されている App Runner マネージドランタイムバージョンを示します。

ランタイムバージョン	App Runner サポート終了日
Python 3.8 でサポートされているランタイム	2025 年 12 月 1 日
Python 3.7 でサポートされているランタイム	2025 年 12 月 1 日
Node.js 18 でサポートされているランタイム	2025 年 12 月 1 日
Node.js 16 でサポートされているランタイム	2025 年 12 月 1 日
Node.js 14 でサポートされているランタイム	2025 年 12 月 1 日
Node.js 12 でサポートされているランタイム	2025 年 12 月 1 日

ランタイムバージョン	App Runner サポート終了日
.NET 6 *	2025 年 12 月 1 日
PHP 8.1 *	2025 年 12 月 31 日
Ruby 3.1 *	2025 年 12 月 1 日
Go 1 *	2025 年 12 月 1 日

* App Runner は、アスタリスク (*) が付いたランタイムの新しい言語バージョンをリリースしません。これらのランタイムは、.NET、PHP、Ruby、Go です。これらのランタイム用にコードベースのサービスが設定されている場合は、次のいずれかのアクションをお勧めします。

- 該当する場合は、サービス設定をサポートされている別のマネージドランタイムに切り替えます。
- または、任意のランタイムバージョンでカスタムコンテナイメージを構築し、App Runner イメージベースのサービスのオプションを使用してデプロイします。Amazon ECR でイメージをホストできます。

マネージドランタイムバージョンと App Runner ビルド

App Runner は、最新のメジャーバージョンランタイムで実行されるアプリケーション用に更新されたビルドプロセスを提供します。この改訂されたビルドプロセスは、より迅速で効率的です。また、アプリケーションの実行に必要なソースコード、ビルドアーティファクト、ランタイムのみを含む、フットプリントが小さい最終イメージを作成します。

新しいビルドプロセスを改訂された App Runner ビルドと呼び、元のビルドプロセスを元の App Runner ビルドと呼びます。以前のバージョンのランタイムプラットフォームへの重大な変更を避けるために、App Runner は改訂されたビルドを特定のランタイムバージョン、通常は新しくリリースされたメジャーリリースにのみ適用します。

`apprunner.yaml` 設定ファイルに新しいコンポーネントが導入されました。これは、改訂されたビルドを特定のユースケースに対して下位互換性を持たせ、アプリケーションのビルドをより柔軟に設定できるようにするためです。これはオプションの`pre-run`パラメータです。このパラメータをいつ使用するか、およびビルドに関するその他の有用な情報を以下のセクションで説明します。

次の表は、特定のマネージドランタイムバージョンに適用される App Runner ビルドのバージョンを示しています。このドキュメントは引き続き更新され、現在のランタイムについてお知らせします。

プラットフォーム	ランタイムバージョン	ビルドプロセス	App Runner サポート終了日
Python – リリース情報	Python 3.11 (!)	改訂	
	Python 3.8	元	2025 年 12 月 1 日
	Python 3.7	元	2025 年 12 月 1 日
Node.js – リリース情報	Node.js 22	改訂	
	Node.js 18	改訂	2025 年 12 月 1 日
	Node.js 16	元	2025 年 12 月 1 日
	Node.js 14	元	2025 年 12 月 1 日
	Node.js 12	元	2025 年 12 月 1 日
Corretto – リリース情報	Corretto 11	元	
	Corretto 8	元	
.NET – リリース情報	.NET 6 *	元	2025 年 12 月 1 日
PHP – リリース情報	PHP 8.1 *	元	2025 年 12 月 31 日
Ruby – リリース情報	Ruby 3.1 *	元	2025 年 12 月 1 日
Go – リリース情報	Go 1 *	元	2025 年 12 月 1 日

> **Note**
>
> リストされているランタイムの一部には、サポート終了日が含まれています。詳細については、「the section called “マネージドランタイムバージョンのサポート終了”」を参照してください。

> **Important**
>
> Python 3.11 – Python 3.11 マネージドランタイムを使用するサービスのビルド設定に関する特定の推奨事項があります。詳細については、Python プラットフォームトピック特定のランタイムバージョンのコールアウトの「」を参照してください。

App Runner のビルドと移行の詳細

改訂されたビルドを使用する新しいランタイムにアプリケーションを移行する場合、ビルド設定を少し変更する必要がある場合があります。

移行に関する考慮事項のコンテキストを提供するために、最初に元の App Runner ビルドと改訂されたビルドの両方の高レベルプロセスについて説明します。設定の更新が必要になる可能性のあるサービスに関する特定の属性を記述するセクションに続きます。

元の App Runner ビルド

元の App Runner アプリケーションビルドプロセスは、 AWS CodeBuild サービスを活用します。初期ステップは、CodeBuild サービスによってキュレートされたイメージに基づいています。Docker ビルドプロセスは、該当する App Runner マネージドランタイムイメージをベースイメージとして使用するプロセスに従います。

一般的なステップは次のとおりです。

1. CodeBuild でキュレートされたイメージで`pre-build`コマンドを実行します。

   `pre-build` コマンドはオプションです。`apprunner.yaml` これらは設定ファイルでのみ指定できます。
2. 前のステップと同じイメージで CodeBuild を使用して`build`コマンドを実行します。

   `build` コマンドは必須です。これらは、App Runner コンソール、App Runner API、または`apprunner.yaml`設定ファイルで指定できます。
3. Docker ビルドを実行して、特定のプラットフォームとランタイムバージョンの App Runner マネージドランタイムイメージに基づいてイメージを生成します。
4. ステップ 2 で生成したイメージから /app ディレクトリをコピーします。送信先は、ステップ 3 で生成した App Runner マネージドランタイムイメージに基づくイメージです。
5. 生成された App Runner マネージドランタイムイメージで`build`コマンドを再度実行します。ビルドコマンドを再度実行して、ステップ 4 でコピーした`/app`ディレクトリのソースコードから

ビルドアーティファクトを生成します。このイメージは、後で App Runner によってデプロイされ、コンテナでウェブサービスを実行します。

`build` コマンドは必須です。これらは、App Runner コンソール、App Runner API、または`apprunner.yaml`設定ファイルで指定できます。

6. ステップ 2 の CodeBuild イメージで`post-build`コマンドを実行します。

   `post-build` コマンドはオプションです。`apprunner.yaml` これらは設定ファイルでのみ指定できます。

ビルドが完了すると、App Runner はステップ 5 で生成された App Runner マネージドランタイムイメージをデプロイして、コンテナでウェブサービスを実行します。

改訂された App Runner ビルド

改訂されたビルドプロセスは、前のセクションで説明した元のビルドプロセスよりも高速で効率的です。これにより、以前のバージョンビルドで発生するビルドコマンドの重複がなくなります。また、アプリケーションの実行に必要なソースコード、ビルドアーティファクト、ランタイムのみを含む、フットプリントが小さい最終イメージを作成します。

このビルドプロセスでは、Docker マルチステージビルドを使用します。一般的なプロセスステップは次のとおりです。

1. ビルドステージ — App Runner ビルドイメージ上で `pre-build`および `build` コマンドを実行する docker ビルドプロセスを開始します。

   a. アプリケーションのソースコードを `/app` ディレクトリにコピーします。

   > **Note**
   >
   > この`/app`ディレクトリは、Docker ビルドのすべてのステージで作業ディレクトリとして指定されます。

   b. `pre-build` コマンドを実行します。

      コマンドは`pre-build`オプションです。`apprunner.yaml` これらは設定ファイルでのみ指定できます。

   c. `build` コマンドを実行します。

build コマンドは必須です。これらは、App Runner コンソール、App Runner API、またはapprunner.yaml設定ファイルで指定できます。

2. パッケージングステージ — App Runner 実行イメージにも基づいている、最終的な顧客コンテナイメージを生成します。
   a. ディレクトリを以前のビルドステージ/appから新しい実行イメージにコピーします。これには、アプリケーションのソースコードと前のステージのビルドアーティファクトが含まれます。
   b. pre-run コマンドを実行します。build コマンドを使用して/appディレクトリの外部でランタイムイメージを変更する必要がある場合は、apprunner.yaml設定ファイルのこのセグメントに同じコマンドまたは必要なコマンドを追加します。

      これは、改訂された App Runner ビルドをサポートするために導入された新しいパラメータです。

      pre-run コマンドはオプションです。apprunner.yaml これらは設定ファイルでのみ指定できます。

      > 注意事項
      > - pre-run コマンドは、改訂されたビルドでのみサポートされます。サービスが元のビルドを使用するランタイムバージョンを使用している場合は、設定ファイルに追加しないでください。
      > - build コマンドを使用して/appディレクトリの外部を変更する必要がない場合は、pre-runコマンドを指定する必要はありません。

3. ビルド後ステージ — このステージはビルドステージから再開し、post-buildコマンドを実行します。
   a. /app ディレクトリ内でpost-buildコマンドを実行します。

      post-build コマンドはオプションです。apprunner.yaml これらは設定ファイルでのみ指定できます。

ビルドが完了すると、App Runner は Run イメージをデプロイして、コンテナでウェブサービスを実行します。

> **Note**
>
> ビルドプロセスを設定するapprunner.yamlときに、 の実行セクションのenvエントリに誤解しないでください。ステップ 2 (b) で参照される pre-run コマンドパラメータは Run セクションにありますが、Run セクションの envパラメータを使用してビルドを設定しないでください。pre-run コマンドは、設定ファイルのビルドセクションで定義されているenv変数のみを参照します。詳細については、App Runner 設定ファイルの章実行セクションの「」を参照してください。

移行に関する考慮事項のサービス要件

アプリケーション環境にこれら 2 つの要件のいずれかがある場合は、 pre-run コマンドを追加してビルド設定を修正する必要があります。

- build コマンドを使用して/appディレクトリの外部を変更する必要がある場合。
- コマンドを 2 build回実行して必要な環境を作成する必要がある場合。これは非常にまれな要件です。ほとんどのビルドはこれを行いません。

/app ディレクトリ外の変更

- 改訂された App Runner ビルドでは、アプリケーションに /app ディレクトリ外の依存関係がないことを前提としています。
- apprunner.yaml ファイル、App Runner API、または App Runner コンソールで指定するコマンドは、 /app ディレクトリにビルドアーティファクトを生成する必要があります。
- pre-build、、および post-build コマンドを変更してbuild、すべてのビルドアーティファクトが /app ディレクトリにあることを確認できます。
- アプリケーションのビルドで、 サービスの生成されたイメージを /app ディレクトリの外部でさらに変更する必要がある場合は、 で新しいpre-runコマンドを使用できますapprunner.yaml。詳細については、「設定ファイルを使用した App Runner サービスオプションの設定」を参照してください。

build コマンドを 2 回実行する

- 元の App Runner ビルドは、最初にステップ 2、次にステップ 5 でbuildコマンドを 2 回実行します。 改訂された App Runner ビルドは、この冗長性を修復し、buildコマンドを 1 回だけ実行し

ます。アプリケーションにbuildコマンドを 2 回実行するための異常な要件がある場合、改訂された App Runner ビルドには、 pre-runパラメータを使用して同じコマンドを再度指定して実行するオプションが用意されています。これにより、同じダブルビルド動作が保持されます。

Python プラットフォームを使用する

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Python 3.7 および Python 3.8 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

Python AWS App Runner プラットフォームはマネージドランタイムを提供します。各ランタイムにより、Python バージョンに基づいてウェブアプリケーションを使用してコンテナを簡単に構築および実行できます。Python ランタイムを使用すると、App Runner はマネージド Python ランタイムイメージから開始します。このイメージは Amazon Linux Docker イメージに基づいており、Python のバージョン用のランタイムパッケージと、いくつかのツールと一般的な依存関係パッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで runtimeキーワードを使用します。マネージドランタイムの命名規則は *<language-name><major-version>* です。

有効な Python ランタイム名とバージョンについては、「」を参照してくださいthe section called “リリース情報”。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションで特定のバージョンのマネージドランタイムが必要な場合は、App Runner 設定ファイルの runtime-versionキーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

Python ランタイムのバージョン構文: *major*[.*minor*[.*patch*]]

例: `3.8.5`

次の例は、バージョンロックを示しています。

- `3.8` – メジャーバージョンとマイナーバージョンをロックします。App Runner はパッチバージョンのみを更新します。
- `3.8.5` – 特定のパッチバージョンにロックします。App Runner はランタイムバージョンを更新しません。

トピック

Python ランタイム設定

マネージドランタイムを選択するときは、少なくとも コマンドを構築して実行することも設定する必要があります。App Runner サービスを作成または更新するときに設定します。これは、次のいずれかの方法を使用して実行できます。

- App Runner コンソールの使用 – 作成プロセスまたは設定タブのビルドの設定セクションでコマンドを指定します。
- App Runner API の使用 – CreateService または UpdateService API オペレーションを呼び出します。CodeConfigurationValues データ型の `BuildCommand`および `StartCommand`メンバーを使用してコマンドを指定します。
- 設定ファイルの使用 – 最大 3 つのビルドフェーズで 1 つ以上のビルドコマンドと、アプリケーションを起動する 1 つの実行コマンドを指定します。追加のオプションの設定があります。

設定ファイルの提供はオプションです。コンソールまたは API を使用して App Runner サービスを作成する場合、App Runner が作成時に設定を直接取得するか、設定ファイルから取得するかを指定します。

特定のランタイムバージョンのコールアウト

> **Note**
>
> App Runner は、Python 3.11、Node.js 22、Node.js 18 のランタイムバージョンに基づいて、アプリケーションの更新されたビルドプロセスを実行するようになりました。アプリケーションがこれらのランタイムバージョンのいずれかで実行されている場合、改訂されたビルドプロセスマネージドランタイムバージョンと App Runner ビルドの詳細については、「」を参照してください。他のすべてのランタイムバージョンを使用するアプリケーションは影響を受けず、元のビルドプロセスを引き続き使用します。

Python 3.11 (App Runner ビルドの改訂)

マネージド Python 3.11 ランタイムには、apprunner.yaml で次の設定を使用します。

- トップセクションの `runtime` キーを に設定します。 `python311`

  Example

  ```
  runtime: python311
  ```

- 依存関係をインストールするpipには、 pip3の代わりに を使用します。
- の代わりに python3 インタープリタを使用しますpython。
- pip3 インストーラをpre-runコマンドとして実行します。Python は/app、 ディレクトリの外部に依存関係をインストールします。App Runner は Python 3.11 用に改訂された App Runner ビルドを実行するため、apprunner.yamlファイルのビルドセクションのコマンドを使用して/appディレクトリの外部にインストールされたすべてのものは失われます。詳細については、「改訂された App Runner ビルド」を参照してください。

  Example

  ```
  run:
    runtime-version: 3.11
    pre-run:
      - pip3 install pipenv
      - pipenv install
      - python3 copy-global-files.py
    command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  ```

詳細については、このトピックの後半にある Python 3.11 用の拡張設定ファイルの例も参照してください。

Python ランタイムの例

次の例は、Python サービスを構築および実行するための App Runner 設定ファイルを示しています。最後の例は、Python ランタイムサービスにデプロイできる完全な Python アプリケーションのソースコードです。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *3.7.7* および *3.11* です。使用するバージョンに置き換えることができます。サポートされている最新の Python ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

最小 Python 設定ファイル

この例では、Python マネージドランタイムで使用できる最小限の設定ファイルを示しています。App Runner が最小限の設定ファイルで行う前提については、「」を参照してくださいthe section called “設定ファイルの例”。

Python 3.11 では、 コマンドpip3と python3 コマンドを使用します。詳細については、このトピックの後半にある Python 3.11 用の拡張設定ファイルの例を参照してください。

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install pipenv
      - pipenv install
run:
  command: python app.py
```

拡張 Python 設定ファイル

この例では、Python マネージドランタイムですべての設定キーを使用する方法を示します。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *3.7.7* です。使用するバージョンに置き換えることができます。サポートされている最新の Python ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。
> Python 3.11 では、 コマンドpip3と python3 コマンドを使用します。詳細については、このトピックの後半にある Python 3.11 用の拡張設定ファイルの例を参照してください。

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip install pipenv
      - pipenv install
    post-build:
      - python manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.7.7
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
```

```
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

拡張 Python 設定ファイル — Python 3.11 (改訂されたビルドを使用)

この例では、 で Python 3.11 マネージドランタイムを使用するすべての設定キーの使用を示していますapprunner.yaml。この例では、このバージョンの Python は改訂された App Runner ビルドを使用するため、 pre-runセクションが含まれています。

pre-run パラメータは、改訂された App Runner ビルドでのみサポートされます。アプリケーションが元の App Runner ビルドでサポートされているランタイムバージョンを使用している場合は、このパラメータを設定ファイルに挿入しないでください。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *3.11* です。使用するバージョンに置き換えることができます。サポートされている最新の Python ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip3 install pipenv
      - pipenv install
    post-build:
      - python3 manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
```

```
run:
  runtime-version: 3.11
  pre-run:
    - pip3 install pipenv
    - pipenv install
    - python3 copy-global-files.py
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

完全な Python アプリケーションソース

この例では、Python ランタイムサービスにデプロイできる完全な Python アプリケーションのソースコードを示します。

Example requirements.txt

```
pyramid==2.0
```

Example server.py

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
import os

def hello_world(request):
    name = os.environ.get('NAME')
    if name == None or len(name) == 0:
        name = "world"
    message = "Hello, " + name + "!\n"
```

```
    return Response(message)

if __name__ == '__main__':
    port = int(os.environ.get("PORT"))
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', port, app)
    server.serve_forever()
```

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install -r requirements.txt
run:
  command: python server.py
```

Python ランタイムリリース情報

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Python 3.7 および Python 3.8 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

このトピックでは、App Runner がサポートする Python ランタイムバージョンの詳細を一覧表示します。

サポートされているランタイムバージョン — App Runner ビルドの改訂

ランタイム名	マイナーバージョン	含まれるパッケージ
Python 3.11 (Python311)	3.11.14	SQLite 3.50.2
	3.11.13	SQLite 3.50.2

ランタイム名	マイナーバージョン	含まれるパッケージ
	3.11.13	SQLite 3.50.1
	3.11.12	SQLite 3.50.0
	3.11.11	SQLite 3.49.1
	3.11.10	SQLite 3.46.1
	3.11.9	SQLite 3.46.1
	3.11.8	SQLite 3.45.2
	3.11.7	SQLite 3.44.2

注意事項

- Python 3.11 – Python 3.11 マネージドランタイムを使用するサービスのビルド設定に関する特定の推奨事項があります。詳細については、Python プラットフォームトピック特定のランタイムバージョンのコールアウトの「」を参照してください。
- App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションに、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

サポートされているランタイムバージョン — 元の App Runner ビルド

ランタイム名	マイナーバージョン	含まれるパッケージ
Python 3 (python3)	3.8.20	SQLite 3.50.2
	3.8.20	SQLite 3.50.1
	3.8.20	SQLite 3.50.0
	3.8.16	SQLite 3.46.1

ランタイム名	マイナーバージョン	含まれるパッケージ
	3.8.15	SQLite 3.40.0
	3.7.16	SQLite 3.50.2
	3.7.16	SQLite 3.50.0
	3.7.15	SQLite 3.40.0
	3.7.10	SQLite 3.40.0

> **Note**
>
> App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションに、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

Node.js プラットフォームを使用する

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Node.js 12、Node.js 14、Node.js 16、Node.js 18 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

AWS App Runner Node.js プラットフォームはマネージドランタイムを提供します。各ランタイムにより、Node.js バージョンに基づいてウェブアプリケーションでコンテナを簡単に構築して実行できます。Node.js ランタイムを使用すると、App Runner はマネージド Node.js ランタイムイメージで始まります。このイメージは Amazon Linux Docker イメージに基づいており、Node.js のバージョンと一部のツールのランタイムパッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで `runtime`キーワードを使用します。マネージドランタイムの命名規則は *`<language-name><major-version>`* です。

有効な Node.js ランタイム名とバージョンについては、「」を参照してくださいthe section called "リリース情報"。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションで特定のバージョンのマネージドランタイムが必要な場合は、App Runner 設定ファイルの `runtime-version`キーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

Node.js ランタイムのバージョン構文: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例: `22.14.0`

次の例は、バージョンロックを示しています。

- `22.14` – メジャーバージョンとマイナーバージョンをロックします。App Runner はパッチバージョンのみを更新します。
- `22.14.0` – 特定のパッチバージョンにロックします。App Runner はランタイムバージョンを更新しません。

トピック

- Node.js ランタイム設定
- 特定のランタイムバージョンのコールアウト
- Node.js ランタイムの例
- Node.js ランタイムリリース情報

Node.js ランタイム設定

マネージドランタイムを選択する場合は、少なくとも コマンドを構築して実行するように設定する必要があります。App Runner サービスを作成または更新するときに設定します。これを行うには、次のいずれかの方法を使用します。

- App Runner コンソールの使用 – 作成プロセスまたは設定タブのビルドの設定セクションでコマンドを指定します。
- App Runner API の使用 – CreateService または UpdateService API オペレーションを呼び出します。CodeConfigurationValues データ型の `BuildCommand`および `StartCommand`メンバーを使用してコマンドを指定します。
- 設定ファイルの使用 – 最大 3 つのビルドフェーズで 1 つ以上のビルドコマンドと、アプリケーションを起動する 1 つの実行コマンドを指定します。追加のオプションの設定があります。

設定ファイルの提供はオプションです。コンソールまたは API を使用して App Runner サービスを作成する場合、App Runner が作成時に設定を直接取得するか、設定ファイルから取得するかを指定します。

Node.js ランタイムでは、ソースリポジトリのルート`package.json`にある という名前の JSON ファイルを使用してビルドとランタイムを設定することもできます。このファイルを使用して、Node.js エンジンのバージョン、依存関係パッケージ、およびさまざまなコマンド (コマンドラインアプリケーション) を設定できます。npm や yarn などのパッケージマネージャーは、このファイルをコマンドの入力として解釈します。

例えば、次のようになります。

- npm install は、 の `dependencies`および `devDependencies`ノードで定義されたパッケージをインストールします`package.json`。
- npm start または は、 の`scripts/start`ノードで定義されたコマンドnpm run startを実行しますpackage.json。

次は、`package.json` ファイルの例です。

package.json

```
{
  "name": "node-js-getting-started",
  "version": "0.3.0",
  "description": "A sample Node.js app using Express 4",
  "engines": {
    "node": "22.14.0"
  },
  "scripts": {
    "start": "node index.js",
```

```
    "test": "node test.js"
  },
  "dependencies": {
    "cool-ascii-faces": "^1.3.4",
    "ejs": "^2.5.6",
    "express": "^4.15.2"
  },
  "devDependencies": {
    "got": "^11.3.0",
    "tape": "^4.7.0"
  }
}
```

の詳細についてはpackage.json、npm Docs ウェブサイトの package.json ファイルの作成を参照してください。

ヒント

- package.json ファイルがstartコマンドを定義する場合は、次の例に示すように、App Runner 設定ファイルのrunコマンドとして使用できます。

  Example

  package.json

  ```
  {
    "scripts": {
      "start": "node index.js"
    }
  }
  ```

  apprunner.yaml

  ```
  run:
    command: npm start
  ```

- npm install 開発環境で を実行すると、npm はファイル を作成しますpackage-lock.json。このファイルには、インストールしたばかりのパッケージバージョンのスナップショットが含まれています。その後、npm が依存関係をインストールすると、これらの正確なバージョンが使用されます。yarn をインストールすると、 yarn.lock ファイルが作成されます。これらのファイルをソースコードリポジトリにコミットして、アプリ

ケーションが開発してテストした依存関係のバージョンでインストールされていることを確認します。

- App Runner 設定ファイルを使用して Node.js バージョンを設定し、コマンドを開始することもできます。これを行うと、これらの定義は の定義よりも優先されますpackage.json。App Runner 設定ファイルの nodeバージョンpackage.jsonとruntime-version値の間に競合があると、App Runner ビルドフェーズが失敗します。

特定のランタイムバージョンのコールアウト

Node.js 22 および Node.js 18 (App Runner ビルドの改訂)

App Runner は、Python 3.11、Node.js 22、Node.js 18 のランタイムバージョンに基づいて、アプリケーションの更新されたビルドプロセスを実行するようになりました。アプリケーションがこれらのランタイムバージョンのいずれかで実行されている場合、改訂されたビルドプロセスマネージドランタイムバージョンと App Runner ビルドの詳細については、「」を参照してください。他のすべてのランタイムバージョンを使用するアプリケーションは影響を受けず、元のビルドプロセスを引き続き使用します。

Node.js ランタイムの例

次の例は、Node.js サービスを構築および実行するための App Runner 設定ファイルを示しています。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *22.14.0* です。使用するバージョンに置き換えることができます。サポートされている最新の Node.js ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

最小 Node.js 設定ファイル

この例では、Node.js マネージドランタイムで使用できる最小限の設定ファイルを示しています。App Runner が最小限の設定ファイルで行う前提については、「」を参照してくださいthe section called “設定ファイルの例”。

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
build:
  commands:
    build:
      - npm install --production
run:
  command: node app.js
```

拡張 Node.js 設定ファイル

この例では、Node.js マネージドランタイムですべての設定キーを使用する方法を示します。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *22.14.0* です。使用するバージョンに置き換えることができます。サポートされている最新の Node.js ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
build:
  commands:
    pre-build:
      - npm install --only=dev
      - node test.js
    build:
      - npm install --production
    post-build:
      - node node_modules/ejs/postinstall.js
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 22.14.0
  command: node app.js
  network:
```

```
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

拡張 Node.js 設定ファイル – Node.js 22 (改訂されたビルドを使用)

この例では、 で Node.js マネージドランタイムを使用するすべての設定キーの使用を示していますapprunner.yaml。この例では、このバージョンの Node.js は改訂された App Runner ビルドを使用するため、 pre-runセクションが含まれています。

pre-run パラメータは、改訂された App Runner ビルドでのみサポートされます。アプリケーションが元の App Runner ビルドでサポートされているランタイムバージョンを使用している場合は、このパラメータを設定ファイルに挿入しないでください。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *22.14.0* です。使用するバージョンに置き換えることができます。サポートされている最新の Node.js ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
build:
  commands:
    pre-build:
      - npm install --only=dev
      - node test.js
    build:
      - npm install --production
    post-build:
      - node node_modules/ejs/postinstall.js
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 22.14.0
```

```
  pre-run:
    - node copy-global-files.js
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Grunt を使用した Node.js アプリ

この例では、Grunt で開発された Node.js アプリケーションを設定する方法を示します。Grunt はコマンドライン JavaScript タスクランナーです。反復タスクを実行し、プロセスの自動化を管理して、人為的ミスを減らします。Grunt プラグインと Grunt プラグインは、npm を使用してインストールおよび管理されます。Grunt を設定するには、ソースリポジトリのルートに `Gruntfile.js` ファイルを含めます。

Example package.json

```
{
  "scripts": {
    "build": "grunt uglify",
    "start": "node app.js"
  },
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  },
  "dependencies": {
    "express": "^4.15.2"
  },
}
```

Example Gruntfile.js

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
```

```
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
  grunt.registerTask('default', ['uglify']);

};
```

Example apprunner.yaml

> **Note**
>
> これらの例で使用されるランタイムバージョンは *22.14.0* です。使用するバージョンに置き換えることができます。サポートされている最新の Node.js ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

```
version: 1.0
runtime: nodejs22
build:
  commands:
    pre-build:
      - npm install grunt grunt-cli
      - npm install --only=dev
      - npm run build
    build:
      - npm install --production
run:
  runtime-version: 22.14.0
  command: node app.js
```

```
network:
  port: 8000
  env: APP_PORT
```

Node.js ランタイムリリース情報

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Node.js 12、Node.js 14、Node.js 16、Node.js 18 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

> **Note**
>
> App Runner の標準的な非推奨ポリシーは、ランタイムの主要コンポーネントがコミュニティ長期サポート (LTS) が終了し、セキュリティ更新プログラムが利用できなくなったときに、ランタイムを非推奨にすることです。場合によっては、App Runner は、ランタイムでサポートされている言語バージョンのend-of-supportを超えて、限られた期間ランタイムの非推奨化を遅らせることがあります。このようなケースの例としては、ランタイムのサポートを拡張して、お客様が移行の時間を確保できることが挙げられます。

このトピックでは、App Runner がサポートする Node.js ランタイムバージョンの詳細を一覧表示します。

サポートされているランタイムバージョン — App Runner ビルドの改訂

ランタイム名	マイナーバージョン	含まれるパッケージ
Node.js 22 (nodejs22)	22.21.1	npm 10.9.4、ヤーン 1.22.22
	22.20.0	npm 10.9.3、ヤーン 1.22.22
	22.17.0	npm 10.9.2、ヤーン 1.22.22
	22.16.0	npm 10.9.2、ヤーン 1.22.22
	22.14.0	npm 10.9.2、ヤーン 1.22.22

> **Note**
>
> App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションに、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

サポートされているランタイムバージョン — App Runner ビルドの改訂

ランタイム名	マイナーバージョン	含まれるパッケージ
Node.js 18 (nodejs18)	18.20.8	npm 10.8.2、ヤーン 1.22.22
	18.20.7	npm 10.8.2、ヤーン 1.22.22
	18.20.6	npm 10.8.2、ヤーン 1.22.22
	18.20.5	npm 10.8.2、ヤーン 1.22.22
	18.20.4	npm 10.7.0、ヤーン 1.22.22
	18.20.3	npm 10.7.0、ヤーン 1.22.22
	18.20.2	npm 10、ヤーン *
	18.19.1	npm 10、ヤーン *
	18.19.0	npm 10、ヤーン *

サポートされているランタイムバージョン — 元の App Runner ビルド

ランタイム名	マイナーバージョン	含まれるパッケージ
Node.js 16 (nodejs16)	16.20.2	npm 8.19.4、ヤーン 1.22.22
	16.20.1	npm 8.19.4、ヤーン *
	16.20.0	npm 8.19.4、ヤーン *

ランタイム名	マイナーバージョン	含まれるパッケージ
	16.19.1	npm 8.19.4、ヤーン *
	16.19.0	npm 8.19.4、ヤーン *
	16.18.1	npm 8.19.4、ヤーン *
	16.17.1	npm 8.19.4、ヤーン *
	16.17.0	npm 8.19.4、ヤーン *
Node.js 14 (nodejs14)	14.21.3	npm 6.14.18、ヤーン 1.22.22
	14.21.2	npm 6.14.18、 yarn *
	14.21.1	npm 6.14.18、 yarn *
	14.20.1	npm 6.14.18、 yarn *
	14.19.0	npm 6.14.18、 yarn *
Node.js 12 (nodejs12)	12.22.12	npm 6.14.16、ヤーン 1.22.22
	12.21.0	npm 6.14.16、 yarn *

Java プラットフォームの使用

Java AWS App Runner プラットフォームはマネージドランタイムを提供します。各ランタイムにより、Java バージョンに基づいてウェブアプリケーションでコンテナを簡単に構築して実行できます。Java ランタイムを使用すると、App Runner はマネージド Java ランタイムイメージから開始します。このイメージは Amazon Linux Docker イメージに基づいており、Java のバージョンと一部のツールのランタイムパッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで `runtime`キーワー

ドを使用します。マネージドランタイムの命名規則は *<language-name><major-version>* です。

現時点では、サポートされているすべての Java ランタイムは Amazon Corretto に基づいています。有効な Java ランタイム名とバージョンについては、「」を参照してくださいthe section called “リリース情報”。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションで特定のバージョンのマネージドランタイムが必要な場合は、App Runner 設定ファイルの `runtime-version`キーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

Amazon Corretto ランタイムのバージョン構文:

ランタイム	[Syntax] (構文)	例
corretto11	`11.0[.`*`openjdk-update`* `[.`*`openjdk-build`* `[.`*`corretto-specific-revision`* `]]]`	`11.0.13.08.1`
corretto8	`8[.`*`openjdk-update`* `[.`*`openjdk-build`* `[.`*`corretto-specific-revision`* `]]]`	`8.312.07.1`

次の例は、バージョンロックを示しています。

- `11.0.13` – Open JDK 更新バージョンをロックします。App Runner は Open JDK と Amazon Corretto の下位レベルのビルドのみを更新します。
- `11.0.13.08.1` – 特定のバージョンにロックします。App Runner はランタイムバージョンを更新しません。

トピック

- Java ランタイム設定
- Java ランタイムの例
- Java ランタイムリリース情報

Java ランタイム設定

マネージドランタイムを選択するときは、少なくとも コマンドを構築して実行することも設定する必要があります。App Runner サービスを作成または更新するときに設定します。これは、次のいずれかの方法を使用して実行できます。

- App Runner コンソールの使用 – 作成プロセスまたは設定タブのビルドの設定セクションでコマンドを指定します。
- App Runner API の使用 – CreateService または UpdateService API オペレーションを呼び出します。CodeConfigurationValues データ型の `BuildCommand`および `StartCommand`メンバーを使用してコマンドを指定します。
- 設定ファイルの使用 – 最大 3 つのビルドフェーズで 1 つ以上のビルドコマンドと、アプリケーションを起動する 1 つの実行コマンドを指定します。追加のオプションの設定があります。

設定ファイルの提供はオプションです。コンソールまたは API を使用して App Runner サービスを作成する場合、App Runner が作成時に設定を直接取得するか、設定ファイルから取得するかを指定します。

Java ランタイムの例

次の例は、Java サービスを構築および実行するための App Runner 設定ファイルを示しています。最後の例は、Corretto 11 ランタイムサービスにデプロイできる完全な Java アプリケーションのソースコードです。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *`11.0.13.08.1`* です。使用するバージョンに置き換えることができます。サポートされている最新の Java ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

最小 Corretto 11 設定ファイル

この例では、Corretto 11 マネージドランタイムで使用できる最小限の設定ファイルを示しています。App Runner が最小限の設定ファイルで行う前提については、「」を参照してください。

Example apprunner.yaml

```
version: 1.0
```

```
runtime: corretto11
build:
  commands:
    build:
      - mvn clean package
run:
  command: java -Xms256m -jar target/MyApp-1.0-SNAPSHOT.jar .
```

拡張 Corretto 11 設定ファイル

この例では、Corretto 11 マネージドランタイムですべての設定キーを使用する方法を示します。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *11.0.13.08.1* です。使用するバージョンに置き換えることができます。サポートされている最新の Java ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: corretto11
build:
  commands:
    pre-build:
      - yum install some-package
      - scripts/prebuild.sh
    build:
      - mvn clean package
    post-build:
      - mvn clean test
  env:
    - name: M2
      value: "/usr/local/apache-maven/bin"
    - name: M2_HOME
      value: "/usr/local/apache-maven/bin"
run:
  runtime-version: 11.0.13.08.1
  command: java -Xms256m -jar target/MyApp-1.0-SNAPSHOT.jar .
  network:
    port: 8000
```

```
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Corretto 11 アプリケーションソースを完了する

この例では、Corretto 11 ランタイムサービスにデプロイできる完全な Java アプリケーションのソースコードを示します。

Example src/main/java/com/HelloWorld/HelloWorld.java

```
package com.HelloWorld;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorld {

    @RequestMapping("/")
    public String index(){
        String s = "Hello World";
        return s;
    }
}
```

Example src/main/java/com/HelloWorld/Main.java

```
package com.HelloWorld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {

    public static void main(String[] args) {

        SpringApplication.run(Main.class, args);
    }
}
```

Example apprunner.yaml

```
version: 1.0
runtime: corretto11
build:
  commands:
    build:
      - mvn clean package
run:
  command: java -Xms256m -jar target/HelloWorldJavaApp-1.0-SNAPSHOT.jar .
  network:
    port: 8080
```

Example pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
    <relativePath/>
  </parent>
  <groupId>com.HelloWorld</groupId>
  <artifactId>HelloWorldJavaApp</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <java.version>11</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
```

```
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <release>11</release>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Java ランタイムリリース情報

このトピックでは、App Runner がサポートする Java ランタイムバージョンの詳細を一覧表示します。

サポートされているランタイムバージョン — 元の App Runner ビルド

ランタイム名	マイナーバージョン	含まれるパッケージ
Corretto 11 (corretto11)	11.0.28.6.1	Maven 3.9.11、Gradle 6.9.4
	11.0.27.6.1	Maven 3.9.10、Gradle 6.9.4
	11.0.27.6.1	Maven 3.9.9、Gradle 6.9.4

ランタイム名	マイナーバージョン	含まれるパッケージ
	11.0.26.4.1	Maven 3.9.9、Gradle 6.9.4
	11.0.25.9.1	Maven 3.9.9、Gradle 6.9.4
	11.0.24.8.1	Maven 3.9.9、Gradle 6.9.4
	11.0.23.9.1	Maven 3.9.8、Gradle 6.9.4
	11.0.22.7.1	Maven 3.9.6、Gradle 6.9.4
	11.0.21.9.1	Maven 3.9.6、Gradle 6.9.4
	11.0.21.9.1	Maven 3.9.5、Gradle 6.9.4
	11.0.20.8.1	Maven 3.9.3、Gradle 6.9.4
	11.0.19.7.1	Maven 3.9.3、Gradle 6.9.4
	11.0.18.10.1	Maven 3.9.1、Gradle 6.9.4
	11.0.17.8.1	Maven 3.8.6、Gradle 6.9.3
	11.0.16.9.1	Maven 3.8.6、Gradle 6.9.2
	11.0.13.08.1	Maven 3.6.3、Gradle 6.5
Corretto 8 (corretto8)	8.472.08.1	Maven 3.9.11、Gradle 6.9.4
	8.462.08.1	Maven 3.9.11、Gradle 6.9.4
	8.452.09.2	Maven 3.9.10、Gradle 6.9.4
	8.452.09.2	Maven 3.9.9、Gradle 6.9.4
	8.452.09.1	Maven 3.9.9、Gradle 6.9.4
	8.442.06.1	Maven 3.9.9、Gradle 6.9.4
	8.432.06.1	Maven 3.9.9、Gradle 6.9.4

ランタイム名	マイナーバージョン	含まれるパッケージ
	8.422.05.1	Maven 3.9.9、Gradle 6.9.4
	8.412.08.1	Maven 3.9.8、Gradle 6.9.4
	8.402.08.1	Maven 3.9.6、Gradle 6.9.4
	8.392.08.1	Maven 3.9.6、Gradle 6.9.4
	8.382.05.1	Maven 3.9.4、Gradle 6.9.4
	8.372.07.1	Maven 3.9.3、Gradle 6.9.4
	8.362.08.1	Maven 3.9.1、Gradle 6.9.4
	8.352.08.1	Maven 3.8.6、Gradle 6.9.3
	8.342.07.4	Maven 3.8.6、Gradle 6.9.2
	8.312.07.1	Maven 3.6.3、Gradle 6.5

Note

App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションには、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

.NET プラットフォームを使用する

Important

App Runner は .NET 6 のサポートを 2025 年 12 月 1 日に終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

AWS App Runner .NET プラットフォームはマネージドランタイムを提供します。各ランタイムにより、.NET バージョンに基づいてウェブアプリケーションを使用してコンテナを簡単に構築および実行できます。.NET ランタイムを使用すると、App Runner はマネージド .NET ランタイムイメージで始まります。このイメージは Amazon Linux Docker イメージに基づいており、.NET のバージョン用のランタイムパッケージと、いくつかのツールと一般的な依存関係パッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで `runtime`キーワードを使用します。マネージドランタイムの命名規則は *`<language-name><major-version>`* です。

有効な .NET ランタイム名とバージョンについては、「」を参照してくださいthe section called “リリース情報”。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションで特定のバージョンのマネージドランタイムが必要な場合は、App Runner 設定ファイルの `runtime-version`キーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

.NET ランタイムのバージョン構文: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例: `6.0.9`

次の例は、バージョンロックを示しています。

- `6.0` – メジャーバージョンとマイナーバージョンをロックします。App Runner はパッチバージョンのみを更新します。
- `6.0.9` – 特定のパッチバージョンにロックします。App Runner はランタイムバージョンを更新しません。

トピック

- .NET ランタイム設定
- .NET ランタイムの例

- .NET ランタイムリリース情報

.NET ランタイム設定

マネージドランタイムを選択する場合は、少なくとも コマンドを構築して実行するように設定する必要があります。App Runner サービスを作成または更新するときに設定します。これを行うには、次のいずれかの方法を使用します。

- App Runner コンソールの使用 – 作成プロセスまたは設定タブのビルドの設定セクションでコマンドを指定します。
- App Runner API の使用 – CreateService または UpdateService API オペレーションを呼び出します。CodeConfigurationValues データ型の `BuildCommand`および `StartCommand`メンバーを使用してコマンドを指定します。
- 設定ファイルの使用 – 最大 3 つのビルドフェーズで 1 つ以上のビルドコマンドと、アプリケーションを起動する 1 つの実行コマンドを指定します。追加のオプションの設定があります。

設定ファイルの提供はオプションです。コンソールまたは API を使用して App Runner サービスを作成する場合、App Runner が作成時に設定を直接取得するか、設定ファイルから取得するかを指定します。

.NET ランタイムの例

次の例は、.NET サービスを構築および実行するための App Runner 設定ファイルを示しています。最後の例は、.NET ランタイムサービスにデプロイできる完全な .NET アプリケーションのソースコードです。

> **Note**
>
> これらの例で使用されるランタイムバージョンは `6.0.9` です。使用するバージョンに置き換えることができます。サポートされている最新の .NET ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

最小 .NET 設定ファイル

この例では、.NET マネージドランタイムで使用できる最小限の設定ファイルを示しています。App Runner が最小限の設定ファイルで行う前提については、「」を参照してくださいthe section called “設定ファイルの例”。

Example apprunner.yaml

```
version: 1.0
runtime: dotnet6
build:
  commands:
    build:
      - dotnet publish -c Release -o out
run:
  command: dotnet out/HelloWorldDotNetApp.dll
```

拡張 .NET 設定ファイル

この例では、.NET マネージドランタイムですべての設定キーを使用する方法を示します。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *6.0.9* です。使用するバージョンに置き換えることができます。サポートされている最新の .NET ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: dotnet6
build:
  commands:
    pre-build:
      - scripts/prebuild.sh
    build:
      - dotnet publish -c Release -o out
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 6.0.9
  command: dotnet out/HelloWorldDotNetApp.dll
  network:
```

```
    port: 5000
    env: APP_PORT
  env:
    - name: ASPNETCORE_URLS
      value: "http://*:5000"
```

完全な .NET アプリケーションソース

この例では、.NET ランタイムサービスにデプロイできる完全な .NET アプリケーションのソースコードを示します。

> **Note**
>
> - 次のコマンドを実行して、シンプルな .NET 6 ウェブアプリケーションを作成します。`dotnet new web --name HelloWorldDotNetApp -f net6.0`
> - 作成した .NET 6 ウェブアプリ`apprunner.yaml`に を追加します。

Example HelloWorldDotNetApp

```
version: 1.0
runtime: dotnet6
build:
  commands:
    build:
      - dotnet publish -c Release -o out
run:
  command: dotnet out/HelloWorldDotNetApp.dll
  network:
    port: 5000
    env: APP_PORT
  env:
    - name: ASPNETCORE_URLS
      value: "http://*:5000"
```

.NET ランタイムリリース情報

> **Important**
>
> App Runner は .NET 6 のサポートを 2025 年 12 月 1 日に終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

このトピックでは、App Runner がサポートする .NET ランタイムバージョンの詳細を一覧表示します。

サポートされているランタイムバージョン — 元の App Runner ビルド

ランタイム名	マイナーバージョン	含まれるパッケージ
.NET 6 (dotnet6)	6.0.36	.NET SDK 6.0.428
	6.0.33	.NET SDK 6.0.425
	6.0.32	.NET SDK 6.0.424
	6.0.31	.NET SDK 6.0.423
	6.0.30	.NET SDK 6.0.422
	6.0.29	.NET SDK 6.0.421
	6.0.28	.NET SDK 6.0.420
	6.0.26	.NET SDK 6.0.418
	6.0.25	.NET SDK 6.0.417
	6.0.24	.NET SDK 6.0.416
	6.0.22	.NET SDK 6.0.414
	6.0.21	.NET SDK 6.0.413
	6.0.20	.NET SDK 6.0.412

ランタイム名	マイナーバージョン	含まれるパッケージ
	6.0.19	.NET SDK 6.0.411
	6.0.16	.NET SDK 6.0.408
	6.0.15	.NET SDK 6.0.407
	6.0.14	.NET SDK 6.0.406
	6.0.13	.NET SDK 6.0.405
	6.0.12	.NET SDK 6.0.404
	6.0.11	.NET SDK 6.0.403
	6.0.10	.NET SDK 6.0.402
	6.0.9	.NET SDK 6.0.401

Note

App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションには、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

PHP プラットフォームを使用する

Important

App Runner は、2025 年 12 月 31 日に PHP 8.1 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

AWS App Runner PHP プラットフォームはマネージドランタイムを提供します。各ランタイムを使用して、PHP バージョンに基づいてウェブアプリケーションでコンテナを構築および実行できます。PHP ランタイムを使用すると、App Runner はマネージド PHP ランタイムイメージで始まります。このイメージは Amazon Linux Docker イメージに基づいており、PHP のバージョンと一部のツールのランタイムパッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで `runtime`キーワードを使用します。マネージドランタイムの命名規則は *`<language-name><major-version>`* です。

有効な PHP ランタイム名とバージョンについては、「」を参照してくださいthe section called “リリース情報”。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションで特定のバージョンのマネージドランタイムが必要な場合は、App Runner 設定ファイルの `runtime-version`キーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

PHP ランタイムのバージョン構文: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例: `8.1.10`

バージョンロックの例を次に示します。

- `8.1` – メジャーバージョンとマイナーバージョンをロックします。App Runner はパッチバージョンのみを更新します。
- `8.1.10` – 特定のパッチバージョンにロックします。App Runner はランタイムバージョンを更新しません。

> **Important**
>
> デフォルトのリポジトリルートディレクトリ以外の場所で App Runner サービスのコードリポジトリソースディレクトリを指定する場合は、PHP マネージドランタイムバージョンが

PHP 8.1.22以降である必要があります。より前の PHP ランタイムバージョン8.1.22では、デフォルトのルートソースディレクトリのみを使用できます。

トピック

PHP ランタイム設定

マネージドランタイムを選択する場合は、少なくとも コマンドを構築して実行するように設定する必要があります。App Runner サービスを作成または更新するときに設定します。これを行うには、次のいずれかの方法を使用します。

- App Runner コンソールの使用 – 作成プロセスまたは設定タブのビルドの設定セクションでコマンドを指定します。
- App Runner API の使用 – CreateService または UpdateService API オペレーションを呼び出します。CodeConfigurationValues データ型の `BuildCommand`および `StartCommand`メンバーを使用してコマンドを指定します。
- 設定ファイルの使用 – 最大 3 つのビルドフェーズで 1 つ以上のビルドコマンドと、アプリケーションを起動する 1 つの実行コマンドを指定します。追加のオプションの設定があります。

設定ファイルの提供はオプションです。コンソールまたは API を使用して App Runner サービスを作成する場合、App Runner が作成時に設定を直接取得するか、設定ファイルから取得するかを指定します。

互換性

次のいずれかのウェブサーバーを使用して、PHP プラットフォームで App Runner サービスを実行できます。

- Apache HTTP Server
- NGINX

Apache HTTP Server および NGINXは PHP-FPM と互換性があります。Apache HTTP Server とは、次のいずれかNGINXを使用して開始できます。

- スーパーバイザー - の実行の詳細については、「スーパーバイザーの実行」を参照してください。supervisord
- 起動スクリプト

Apache HTTP Server または NGINX を使用して PHP プラットフォームで App Runner サービスを設定する方法の例については、「」を参照してくださいthe section called “PHP アプリケーションソースを完了する”。

ファイル構造

は、ウェブサーバーの `root` ディレクトリの `public`フォルダにインストール`index.php`する必要があります。

> **Note**
>
> `startup.sh` または `supervisord.conf`ファイルは、ウェブサーバーのルートディレクトリに保存することをお勧めします。`start` コマンドが `startup.sh`または `supervisord.conf`ファイルが保存されている場所を指していることを確認します。

を使用している場合のファイル構造の例を次に示しますsupervisord。

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

スタートアップスクリプトを使用している場合のファイル構造の例を次に示します。

```
/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

これらのファイル構造は、App Runner サービス用に指定されたコードリポジトリのソースディレクトリに保存することをお勧めします。

```
/<sourceDirectory>/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

> **Important**
>
> デフォルトのリポジトリルートディレクトリ以外の場所で App Runner サービスのコードリポジトリソースディレクトリを指定する場合、PHP マネージドランタイムバージョンは PHP 8.1.22以降である必要があります。より前の PHP ランタイムバージョン8.1.22では、デフォルトのルートソースディレクトリのみを使用できます。
> App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。App Runner 設定ファイルの `runtime-version`キーワードを使用してバージョンロックを指定しない限り、サービスはデフォルトで最新のランタイムを使用します。

PHP ランタイムの例

PHP サービスの構築と実行に使用される App Runner 設定ファイルの例を次に示します。

最小 PHP 設定ファイル

次の例は、PHP マネージドランタイムで使用できる最小限の設定ファイルです。最小設定ファイルの詳細については、「」を参照してくださいthe section called “設定ファイルの例”。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
```

拡張 PHP 設定ファイル

次の例では、PHP マネージドランタイムですべての設定キーを使用します。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *8.1.10* です。使用するバージョンに置き換えることができます。サポートされている最新の PHP ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - echo example build command for PHP
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 8.1.10
  command: ./startup.sh
  network:
    port: 5000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

PHP アプリケーションソースを完了する

次の例は、 Apache HTTP Serverまたは を使用して PHP ランタイムサービスにデプロイするために使用できる PHP アプリケーションのソースコードですNGINX。これらの例では、デフォルトのファイル構造を使用することを前提としています。

Apache HTTP Server を使用して で PHP プラットフォームを実行する supervisord

Exampleファイル構造

> **Note**
>
> - supervisord.conf ファイルはリポジトリのどこにでも保存できます。start コマンドがsupervisord.confファイルの保存先を指していることを確認します。
> - は、 root ディレクトリの publicフォルダにインストールindex.phpする必要があります。

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

Example supervisord.conf

```
[supervisord]
nodaemon=true

[program:httpd]
command=httpd -DFOREGROUND
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[program:php-fpm]
command=php-fpm -F
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - PYTHON=python2 amazon-linux-extras install epel
      - yum -y install supervisor
run:
  command: supervisord
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

Apache HTTP Server を使用して で PHP プラットフォームを実行する startup script

Exampleファイル構造

> **Note**
>
> - startup.sh ファイルはリポジトリのどこにでも保存できます。start コマンドがstartup.shファイルの保存先を指していることを確認します。
> - は、 root ディレクトリの publicフォルダにインストールindex.phpする必要があります。

```
/
```

```
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

Example startup.sh

```
#!/bin/bash

set -o monitor

trap exit SIGCHLD

# Start apache
httpd -DFOREGROUND &

# Start php-fpm
php-fpm -F &

wait
```

Note

- Git リポジトリにコミットする前に、startup.shファイルを実行可能ファイルとして保存してください。を使用してchmod +x startup.sh、 startup.sh ファイルに実行アクセス許可を設定します。
- startup.sh ファイルを実行可能ファイルとして保存しない場合は、apprunner.yamlファイルに build コマンドchmod +x startup.shとして と入力します。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
```

```
  command: ./startup.sh
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

NGINX を使用して で PHP プラットフォームを実行する supervisord

Exampleファイル構造

> **Note**
>
> - supervisord.conf ファイルはリポジトリのどこにでも保存できます。start コマンドがsupervisord.confファイルの保存先を指していることを確認します。
> - は、 root ディレクトリの publicフォルダにインストールindex.phpする必要があります。

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

Example supervisord.conf

```
[supervisord]
nodaemon=true
```

```
[program:nginx]
command=nginx -g "daemon off;"
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[program:php-fpm]
command=php-fpm -F
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - PYTHON=python2 amazon-linux-extras install epel
      - yum -y install supervisor
run:
  command: supervisord
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
```

```
</body>
</html>
```

NGINX を使用した PHP プラットフォームの実行 startup script

Exampleファイル構造

> **Note**
>
> - `startup.sh` ファイルはリポジトリのどこにでも保存できます。`start` コマンドが`startup.sh`ファイルの保存先を指していることを確認します。
> - は、 `root` ディレクトリの `public`フォルダにインストール`index.php`する必要があります。

```
/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

Example startup.sh

```
#!/bin/bash

set -o monitor

trap exit SIGCHLD

# Start nginx
nginx -g 'daemon off;' &

# Start php-fpm
php-fpm -F &

wait
```

> **Note**
>
> - Git リポジトリにコミットする前に、startup.shファイルを実行可能ファイルとして保存してください。を使用してchmod +x startup.sh、 startup.sh ファイルに実行アクセス許可を設定します。
> - startup.sh ファイルを実行可能ファイルとして保存しない場合は、apprunner.yamlファイルに build コマンドchmod +x startup.shとして と入力します。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

PHP ランタイムリリース情報

> **Important**
>
> App Runner は、2025 年 12 月 31 日に PHP 8.1 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

このトピックでは、App Runner がサポートする PHP ランタイムバージョンの詳細を一覧表示します。

サポートされているランタイムバージョン — 元の App Runner ビルド

ランタイム名	マイナーバージョン	含まれるパッケージ
PHP 8.1 (php81)	8.1.33	
	8.1.32	
	8.1.31	
	8.1.29	
	8.1.28	
	8.1.27	
	8.1.26	
	8.1.24	
	8.1.22	
	8.1.21	
	8.1.20	
	8.1.19	
	8.1.17	

ランタイム名	マイナーバージョン	含まれるパッケージ
	8.1.16	
	8.1.14	
	8.1.13	
	8.1.12	
	8.1.10	

> **Note**
>
> App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションには、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

Ruby プラットフォームを使用する

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Ruby 3.1 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

AWS App Runner Ruby プラットフォームはマネージドランタイムを提供します。各ランタイムにより、Ruby バージョンに基づいてウェブアプリケーションを使用してコンテナを簡単に構築および実行できます。Ruby ランタイムを使用すると、App Runner はマネージド Ruby ランタイムイメージで始まります。このイメージは Amazon Linux Docker イメージに基づいており、Ruby のバージョンと一部のツールのランタイムパッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで `runtime`キーワードを使用します。マネージドランタイムの命名規則は *`<language-name><major-version>`* です。

有効な Ruby ランタイム名とバージョンについては、「」を参照してくださいthe section called “リリース情報”。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションで特定のバージョンのマネージドランタイムが必要な場合は、App Runner 設定ファイルの `runtime-version`キーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

Ruby ランタイムのバージョン構文: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例: `3.1.2`

次の例は、バージョンロックを示しています。

- `3.1` – メジャーバージョンとマイナーバージョンをロックします。App Runner はパッチバージョンのみを更新します。
- `3.1.2` – 特定のパッチバージョンにロックします。App Runner はランタイムバージョンを更新しません。

トピック

- Ruby ランタイム設定
- Ruby ランタイムの例
- Ruby ランタイムリリース情報

Ruby ランタイム設定

マネージドランタイムを選択する場合は、少なくとも コマンドを構築して実行するように設定する必要があります。App Runner サービスを作成または更新するときに設定します。これは、次のいずれかの方法を使用して実行できます。

- App Runner コンソールの使用 – 作成プロセスまたは設定タブのビルドの設定セクションでコマンドを指定します。
- App Runner API の使用 – CreateService または UpdateService API オペレーションを呼び出します。CodeConfigurationValues データ型の `BuildCommand`および `StartCommand`メンバーを使用してコマンドを指定します。
- 設定ファイルの使用 – 最大 3 つのビルドフェーズで 1 つ以上のビルドコマンドと、アプリケーションを起動する 1 つの実行コマンドを指定します。追加のオプションの設定があります。

設定ファイルの提供はオプションです。コンソールまたは API を使用して App Runner サービスを作成する場合、App Runner が作成時に設定を直接取得するか、設定ファイルから取得するかを指定します。

Ruby ランタイムの例

次の例は、Ruby サービスを構築および実行するための App Runner 設定ファイルを示しています。

最小 Ruby 設定ファイル

この例では、Ruby マネージドランタイムで使用できる最小限の設定ファイルを示しています。App Runner が最小限の設定ファイルで行う前提については、「」を参照してくださいthe section called “設定ファイルの例”。

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
    build:
      - bundle install
run:
  command: bundle exec rackup --host 0.0.0.0 -p 8080
```

拡張 Ruby 設定ファイル

この例では、Ruby マネージドランタイムですべての設定キーを使用する方法を示します。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *3.1.2* です。使用するバージョンに置き換えることができます。サポートされている最新の Ruby ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - bundle install
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.1.2
  command: bundle exec rackup --host 0.0.0.0 -p 4567
  network:
    port: 4567
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Ruby アプリケーションソースを完了する

これらの例は、Ruby ランタイムサービスにデプロイできる完全な Ruby アプリケーションのソースコードを示しています。

Example server.rb

```
# server.rb
require 'sinatra'
```

```
get '/' do
  'Hello World!'
end
```

Example config.ru

```
# config.ru

require './server'

run Sinatra::Application
```

Example Gemfile

```
# Gemfile
source 'https://rubygems.org (https://rubygems.org/)'

gem 'sinatra'
gem 'puma'
```

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
    build:
      - bundle install
run:
  command: bundle exec rackup --host 0.0.0.0 -p 4567
  network:
    port: 4567
    env: APP_PORT
```

Ruby ランタイムリリース情報

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Ruby 3.1 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

このトピックでは、App Runner がサポートする Ruby ランタイムバージョンの詳細を一覧表示します。

サポートされているランタイムバージョン — 元の App Runner ビルド

ランタイム名	マイナーバージョン	含まれるパッケージ
Ruby 3.1 (ruby31)	3.1.7	SQLite 3.50.2
	3.1.7	SQLite 3.50.1
	3.1.7	SQLite 3.50.0
	3.1.6	SQLite 3.49.1
	3.1.4	SQLite 3.46.0
	3.1.3	SQLite 3.41.0
	3.1.2	SQLite 3.39.4

> **Note**
>
> App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションには、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

Go プラットフォームを使用する

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Go 1.18 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

AWS App Runner Go プラットフォームはマネージドランタイムを提供します。各ランタイムにより、Go バージョンに基づいてウェブアプリケーションを使用してコンテナを簡単に構築および実行できます。Go ランタイムを使用すると、App Runner はマネージド Go ランタイムイメージで始まります。このイメージは Amazon Linux Docker イメージに基づいており、Go のバージョンと一部のツールのランタイムパッケージが含まれています。App Runner はこのマネージドランタイムイメージをベースイメージとして使用し、アプリケーションコードを追加して Docker イメージを構築します。次に、このイメージをデプロイして、コンテナでウェブサービスを実行します。

App Runner コンソールまたは CreateService API オペレーションを使用してサービスを作成するときに、App Runner サービスのランタイムを指定します。ソースコードの一部としてランタイムを指定することもできます。コードリポジトリに含める App Runner 設定ファイルで `runtime`キーワードを使用します。マネージドランタイムの命名規則は *`<language-name><major-version>`* です。

有効な Go ランタイム名とバージョンについては、「」を参照してくださいthe section called “リリース情報”。

App Runner は、デプロイまたはサービスの更新ごとに、サービスのランタイムを最新バージョンに更新します。アプリケーションでマネージドランタイムの特定のバージョンが必要な場合は、App Runner 設定ファイルの `runtime-version`キーワードを使用して指定できます。メジャーバージョンやマイナーバージョンなど、任意のレベルのバージョンにロックできます。App Runner は、サービスのランタイムに対してのみ低レベルの更新を行います。

Go ランタイムのバージョン構文: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例: `1.18.7`

次の例は、バージョンロックを示しています。

- `1.18` – メジャーバージョンとマイナーバージョンをロックします。App Runner はパッチバージョンのみを更新します。

- `1.18.7` – 特定のパッチバージョンにロックします。App Runner はランタイムバージョンを更新しません。

トピック

- Go ランタイム設定
- Go ランタイムの例
- Go ランタイムリリース情報

Go ランタイム設定

マネージドランタイムを選択する場合は、少なくとも コマンドを構築して実行するように設定する必要があります。App Runner サービスを作成または更新するときに設定します。これを行うには、次のいずれかの方法を使用します。

- App Runner コンソールの使用 – 作成プロセスまたは設定タブのビルドの設定セクションでコマンドを指定します。
- App Runner API の使用 – CreateService または UpdateService API オペレーションを呼び出します。CodeConfigurationValues データ型の `BuildCommand`および `StartCommand`メンバーを使用してコマンドを指定します。
- 設定ファイルの使用 – 最大 3 つのビルドフェーズで 1 つ以上のビルドコマンドと、アプリケーションを起動する 1 つの実行コマンドを指定します。追加のオプションの設定があります。

設定ファイルの提供はオプションです。コンソールまたは API を使用して App Runner サービスを作成する場合、App Runner が作成時に設定を直接取得するか、設定ファイルから取得するかを指定します。

Go ランタイムの例

次の例は、Go サービスを構築および実行するための App Runner 設定ファイルを示しています。

最小 Go 設定ファイル

この例では、Go マネージドランタイムで使用できる最小限の設定ファイルを示しています。App Runner が最小限の設定ファイルで行う前提については、「」を参照してくださいthe section called “設定ファイルの例”。

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
    build:
      - go build main.go
run:
  command: ./main
```

拡張 Go 設定ファイル

この例では、Go マネージドランタイムですべての設定キーを使用する方法を示します。

> **Note**
>
> これらの例で使用されるランタイムバージョンは *1.18.7* です。使用するバージョンに置き換えることができます。サポートされている最新の Go ランタイムバージョンについては、「」を参照してくださいthe section called “リリース情報”。

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - go build main.go
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 1.18.7
  command: ./main
  network:
```

```
    port: 3000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Go アプリケーションソースを完了する

これらの例は、Go ランタイムサービスにデプロイできる完全な Go アプリケーションのソースコードを示しています。

Example main.go

```
package main
import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprint(w, "<h1>Welcome to App Runner</h1>")
    })
    fmt.Println("Starting the server on :3000...")
    http.ListenAndServe(":3000", nil)
}
```

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
    build:
      - go build main.go
run:
  command: ./main
  network:
    port: 3000
    env: APP_PORT
```

Go ランタイムリリース情報

> **Important**
>
> App Runner は、2025 年 12 月 1 日に Go 1.18 のサポートを終了します。推奨事項と詳細については、「」を参照してくださいthe section called “マネージドランタイムバージョンのサポート終了”。

このトピックでは、App Runner がサポートする Go ランタイムバージョンの詳細を一覧表示します。

サポートされているランタイムバージョン — 元の App Runner ビルド

ランタイム名	マイナーバージョン	含まれるパッケージ
Go 1 (go1)	1.18.10	
	1.18.9	
	1.18.8	
	1.18.7	

> **Note**
>
> App Runner は、最近リリースされた特定のメジャーランタイムのビルドプロセスを改訂しました。このため、このドキュメントの特定のセクションに、改訂された App Runner ビルドと元の App Runner ビルドへの参照が表示されます。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

App Runner のアプリケーションコードの開発

この章では、デプロイ先のアプリケーションコードを開発または移行する際に考慮すべきランタイム情報と開発ガイドラインについて説明します AWS App Runner。

ランタイム情報

コンテナイメージを提供するか App Runner が構築するかにかかわらず、App Runner はコンテナインスタンスでアプリケーションコードを実行します。コンテナインスタンスランタイム環境の主な側面をいくつか紹介します。

- フレームワークのサポート – App Runner は、ウェブアプリケーションを実装するすべてのイメージをサポートします。選択したプログラミング言語と、使用するウェブアプリケーションサーバーまたはフレームワークには関係ありません。便宜上、さまざまなプログラミングプラットフォーム用のプラットフォーム固有のマネージドランタイムを提供し、アプリケーションビルドプロセスと抽象イメージの作成を合理化します。
- ウェブリクエスト – App Runner は、コンテナインスタンスに HTTP 1.0 と HTTP 1.1 のサポートを提供します。サービスの設定の詳細については、「」を参照してくださいthe section called “設定”。HTTPS セキュアトラフィックの処理を実装する必要はありません。App Runner は、すべての受信 HTTP リクエストを対応する HTTPS エンドポイントにリダイレクトします。HTTP ウェブリクエストのリダイレクトを有効にするように設定する必要はありません。App Runner は、アプリケーションコンテナインスタンスにリクエストを渡す前に TLS を終了します。

> **Note**
>
> - HTTP リクエストには合計 120 秒のリクエストタイムアウト制限があります。120 秒には、本文を含むリクエストをアプリケーションが読み取り、HTTP レスポンスの書き込みを完了するのにかかる時間が含まれます。
> - リクエストの読み取りとレスポンスのタイムアウトの制限は、使用するアプリケーションによって異なります。これらのアプリケーションには、Python 用の HTTP サーバーである Gunicorn など、独自の内部タイムアウトがあり、デフォルトのタイムアウト制限は 30 秒です。このような場合、アプリケーションのタイムアウト制限は App Runner の 120 秒のタイムアウト制限よりも優先されます。
> - App Runner はフルマネージドサービスであり、TLS 終了を管理するため、TLS 暗号スイートやその他のパラメータを設定する必要はありません。

- ステートレスアプリ – 現在、App Runner はステートフルアプリをサポートしていません。したがって、App Runner は、単一の受信ウェブリクエストを処理する期間を超えて状態が永続化することを保証しません。
- ストレージ – App Runner は、受信トラフィック量に応じて App Runner アプリケーションのインスタンスを自動的にスケールアップまたはスケールダウンします。App Runner アプリケーションの Auto Scaling オプションを設定できます。ウェブリクエストを処理する現在アクティブなインスタンスの数は受信トラフィック量に基づいているため、App Runner はファイルが 1 つのリクエストの処理を超えて保持できることを保証できません。したがって、App Runner はコンテナインスタンスにファイルシステムをエフェメラルストレージとして実装します。そのため、ファイルは一時的なものです。たとえば、App Runner サービスを一時停止して再開しても、ファイルは保持されません。

  App Runner は 3 GB のエフェメラルストレージを提供し、3 GB のエフェメラルストレージの一部をインスタンスのプル、圧縮、および非圧縮コンテナイメージに使用します。残りのエフェメラルストレージは App Runner サービスで使用できます。ただし、ステートレスであるため、これは永続的なストレージではありません。

  > **Note**
  >
  > ストレージファイルがリクエスト間で保持される場合があります。たとえば、次のリクエストが同じインスタンス上にある場合、ストレージファイルは保持されます。リクエスト間のストレージファイルの永続化は、特定の状況で役立ちます。たとえば、リクエストを処理するときに、今後のリクエストで必要になる可能性がある場合は、アプリケーションがダウンロードしたファイルをキャッシュできます。これにより、将来のリクエスト処理が高速化される可能性がありますが、速度の向上を保証することはできません。コードでは、前のリクエストでダウンロードされたファイルがまだ存在すると想定しないでください。
  >
  > 高スループット、低レイテンシーのインメモリデータストアを使用したキャッシュが保証されるようにするには、Amazon ElastiCache などのサービスを使用します。

- 環境変数 – デフォルトでは、App Runner はコンテナインスタンスで`PORT`環境変数を利用できるようにします。ポート情報を使用して変数値を設定し、カスタム環境変数と値を追加できます。AWS Secrets Manager または AWS Systems Manager Parameter Store に保存されている機密データを環境変数として参照することもできます。環境変数の作成の詳細については、「」を参照してくださいリファレンス環境変数。
- インスタンスロール – アプリケーションコードが AWS サービス APIs またはいずれかの AWS SDKs を使用して任意のサービスを呼び出す場合は、 AWS Identity and Access Management

(IAM) を使用してインスタンスロールを作成します。次に、作成時に App Runner サービスにアタッチします。コードが必要とするすべての AWS サービスアクションのアクセス許可をインスタンスロールに含めます。詳細については、「the section called “インスタンスロール”」を参照してください。

コード開発ガイドライン

App Runner ウェブアプリケーションのコードを開発するときは、以下のガイドラインを考慮してください。

- コンテナイメージのパッチ適用 – コンテナイメージを提供する場合、これらのイメージを定期的に更新してパッチを適用する責任があります。App Runner がインフラストラクチャを管理する間は、提供されたコンテナイメージのセキュリティとup-to-dateステータスを確認する必要があります。詳細については、AWS App Runner ドキュメントを参照してください。
- ステートレスコードを設計する – App Runner サービスにデプロイするウェブアプリケーションをステートレスに設計します。コードは、単一の受信ウェブリクエストを処理する期間を超えて状態が維持されないことを前提とする必要があります。
- 一時ファイルの削除 – ファイルを作成すると、ファイルはファイルシステムに保存され、サービスのストレージ割り当ての一部になります。out-of-storageエラーを避けるため、一時ファイルを長期間保持しないでください。ファイルキャッシュの決定を行うときは、ストレージサイズとリクエスト処理速度のバランスを取ります。
- インスタンスの起動 – App Runner はインスタンスの起動時間を 5 分で提供します。インスタンスは、設定されたリッスンポートでリクエストをリッスンし、起動後 5 分以内に正常である必要があります。起動時に、App Runner インスタンスには vCPU 設定に基づいて仮想 CPU (vCPU) が割り当てられます。使用可能な vCPU 設定の詳細については、「」を参照してくださいthe section called “App Runner でサポートされている設定”。

  インスタンスが正常に起動すると、インスタンスはアイドル状態になり、リクエストを待機します。料金は、インスタンスの起動時間に基づき、インスタンスの開始ごとに 1 分の最小料金がかかります。料金については、「AWS App Runner の料金」を参照してください。

App Runner コンソールの使用

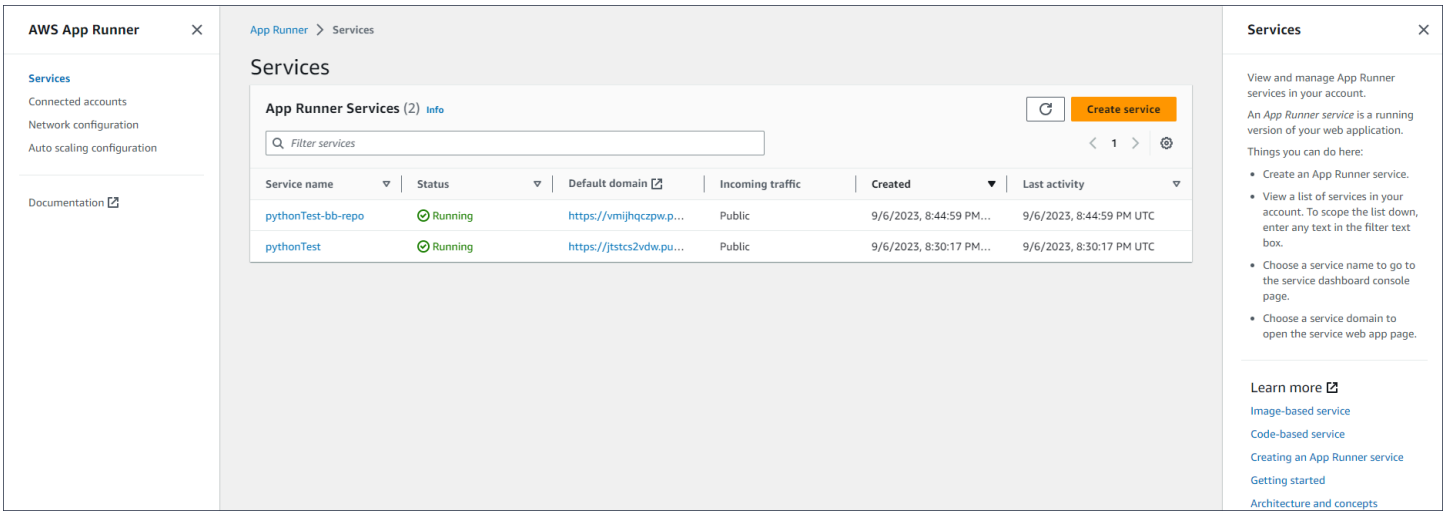
AWS App Runner コンソールを使用して、接続されたアカウントなどの App Runner サービスおよび関連リソースを作成、管理、モニタリングします。既存のサービスの表示、新しいサービスの作成、サービスの設定を行うことができます。App Runner サービスのステータスの表示、ログの表示、アクティビティのモニタリング、メトリクスの追跡を行うことができます。サービスのウェブサイトまたはソースリポジトリに移動することもできます。

以下のセクションでは、コンソールのレイアウトと機能について説明し、関連情報を示します。

コンソール全体のレイアウト

App Runner コンソールには 3 つの領域があります。左から右へ:

- ナビゲーションペイン – 折りたたんだり展開したりできるサイドペイン。これを使用して、使用する最上位コンソールページを選択します。
- コンテンツペイン – コンソールページのメイン部分。これを使用して情報を表示し、タスクを実行します。
- ヘルプペイン – 詳細については、サイドペインを参照してください。展開して、ページに関するヘルプを表示します。または、コンソールページで任意の情報リンクを選択して、コンテキストヘルプを取得します。



サービスページ

サービスページには、アカウントの App Runner サービスが一覧表示されます。フィルターテキストボックスを使用して、リストの範囲を絞り込むことができます。

サービスページに移動するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで [Services (サービス)] を選択します。

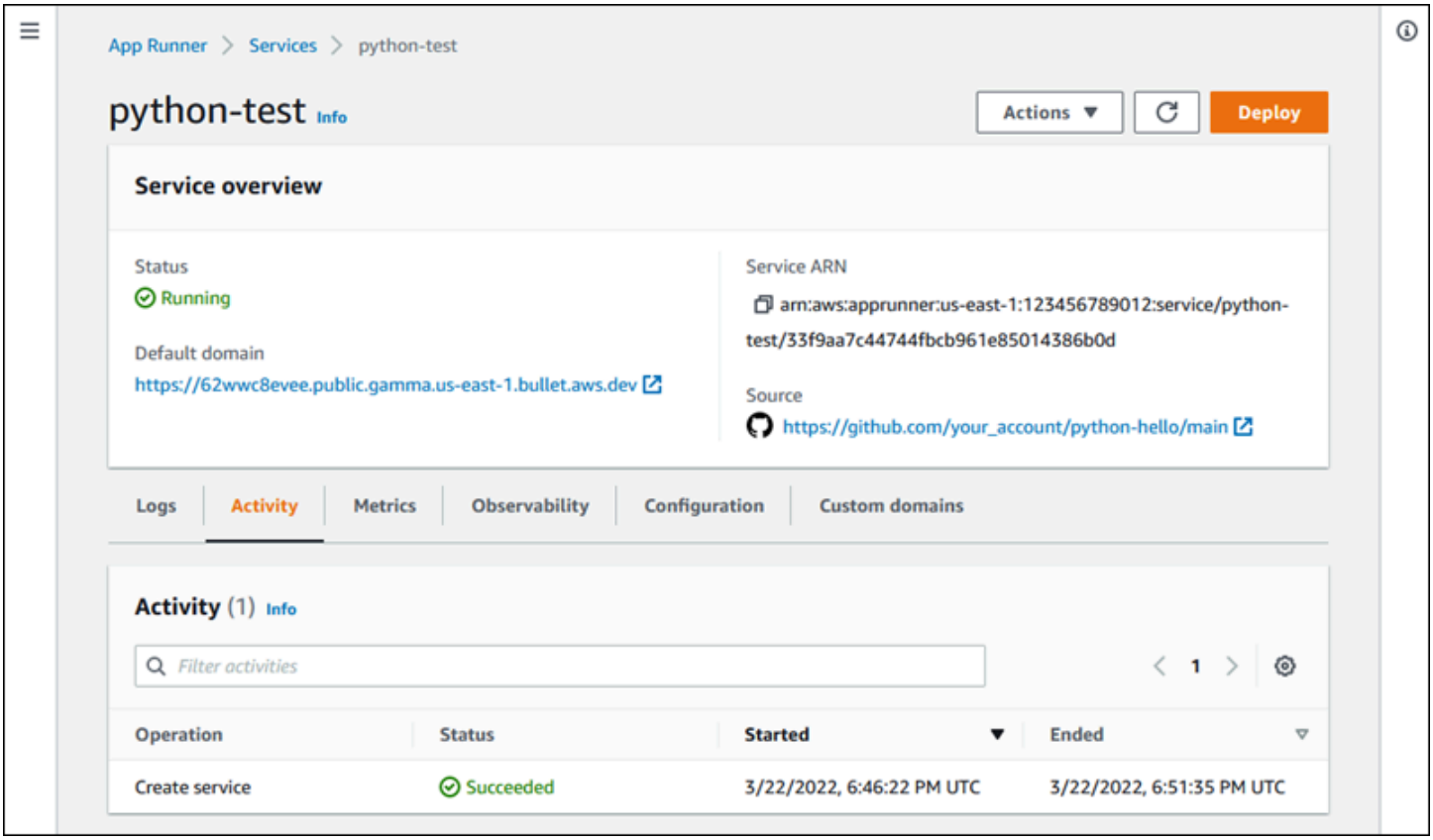
ここでできること:

- App Runner サービスを作成します。詳細については、「the section called “作成”」を参照してください。
- サービス名を選択して、サービスダッシュボードコンソールページに移動します。
- サービスドメインを選択して、サービスウェブページを開きます。

サービスダッシュボードページ

App Runner サービスに関する情報を表示し、サービスダッシュボードページから管理できます。ページの上部にサービス名が表示されます。

サービスダッシュボードにアクセスするには、サービスページに移動し (前のセクションを参照) 、App Runner サービスを選択します。



サービスの概要セクションには、App Runner サービスとアプリケーションに関する基本的な詳細が表示されます。ここでできること:

- ステータス、ヘルス、ARN などのサービスの詳細を表示します。
- デフォルトドメインに移動します。デフォルトドメインは、サービスで実行されているウェブアプリケーション用に App Runner が提供するドメインです。これは App Runner が所有する`awsapprunner.com`ドメインのサブドメインです。
- サービスにデプロイされたソースリポジトリに移動します。
- サービスへのソースリポジトリのデプロイを開始します。
- サービスを一時停止、再開、削除します。

サービスの概要の下のタブは、サービス管理とオブザーバビリティ用です。

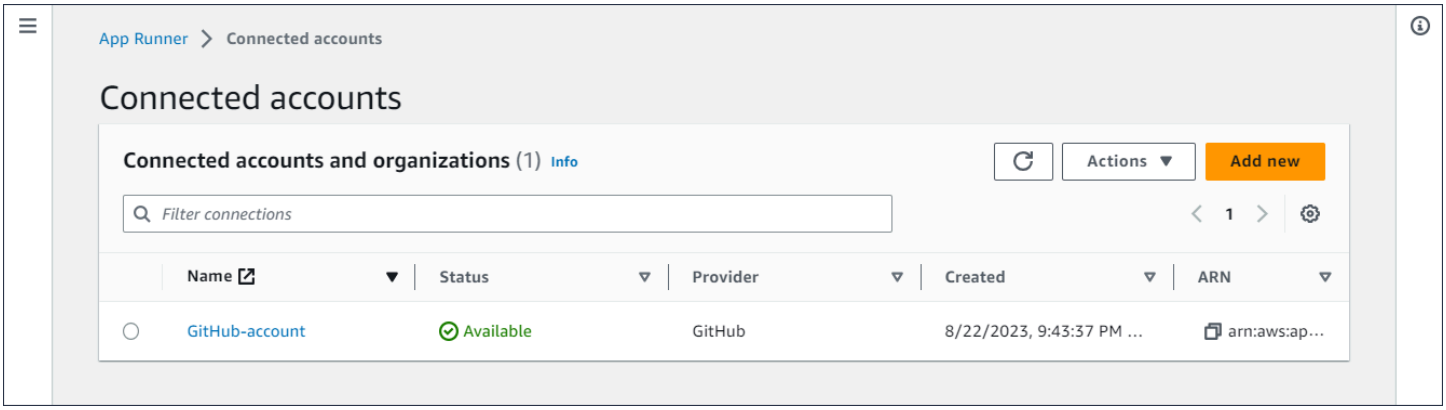
接続されたアカウントページ

接続されたアカウントページには、アカウントのソースコードリポジトリプロバイダーへの App Runner 接続が一覧表示されます。フィルターテキストボックスを使用して、リストの範囲を絞り込

むことができます。接続されたアカウントの詳細については、「」を参照してくださいthe section called “Connections”。

接続されたアカウントページに移動するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、接続されたアカウントを選択します。



ここでできること:

- アカウントのリポジトリプロバイダー接続のリストを表示します。リストの範囲を絞り込むには、フィルターテキストボックスに任意のテキストを入力します。
- 接続名を選択して、関連するプロバイダーアカウントまたは組織に移動します。
- 接続を選択して、(サービスの作成の一環として) 確立した接続のハンドシェイクを完了するか、接続を削除します。

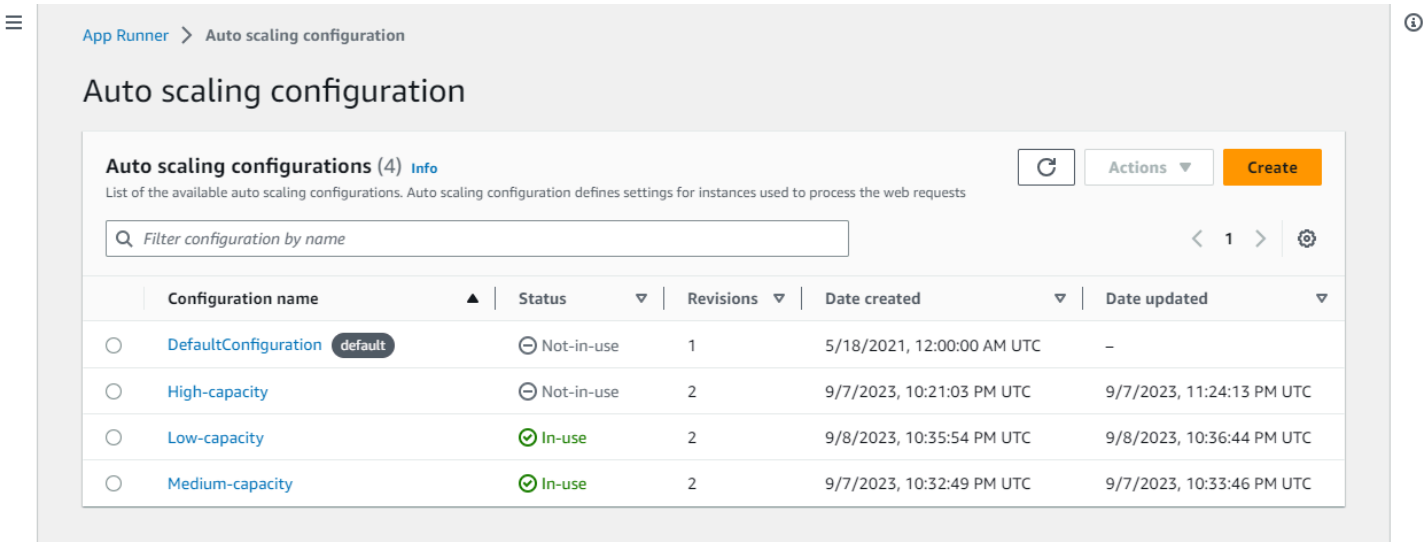
Auto Scaling 設定ページ

Auto Scaling 設定ページには、アカウントで設定した Auto Scaling 設定が一覧表示されます。自動スケーリングの動作を調整し、後で 1 つ以上の App Runner サービスに割り当てることができるさまざまな設定に保存するように、いくつかのパラメータを設定できます。フィルターテキストボックスを使用して、リストの範囲を絞り込むことができます。自動スケーリング設定の詳細については、「」を参照してくださいサービスの自動スケーリングを管理する。

Auto Scaling 設定ページに移動するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。

2. ナビゲーションペインで、Auto Scaling 設定を選択します。



ここでできること:

- アカウント内の既存の Auto Scaling 設定のリストを表示します。
- 新しい自動スケーリング設定または既存の設定のリビジョンを作成します。
- 自動スケーリング設定を、作成する新しいサービスのデフォルトとして設定します。
- 設定を削除します。
- 設定の名前を選択して Auto Scaling リビジョンパネルに移動し、リビジョンを管理します。

App Runner サービスの管理

この章では、 AWS App Runner サービスを管理する方法について説明します。この章では、サービスの作成、設定、削除、新しいアプリケーションバージョンのサービスへのデプロイ、サービスの一時停止と再開によるウェブサービスの可用性の制御など、サービスのライフサイクルを管理する方法について説明します。また、接続や自動スケーリングなど、サービスの他の側面を管理する方法についても説明します。

トピック

App Runner サービスの作成

AWS App Runner は、コンテナイメージまたはソースコードリポジトリから、自動的にスケールする実行中のウェブサービスへの移行を自動化します。App Runner をソースイメージまたはコードにポイントし、少数の必要な設定のみを指定します。App Runner は、必要に応じてアプリケーションを構築し、コンピューティングリソースをプロビジョニングし、アプリケーションを実行できるようにデプロイします。

サービスを作成すると、App Runner はサービスリソースを作成します。場合によっては、接続リソースを提供する必要があります。App Runner コンソールを使用する場合、コンソールは接続リソースを暗黙的に作成します。App Runner リソースタイプの詳細については、「」を参照してくださいthe section called “App Runner リソース”。これらのリソースタイプには、各 のアカウントに関連付けられたクォータがあります AWS リージョン。詳細については、「the section called “App Runner リソースクォータ”」を参照してください。

ソースタイプとプロバイダーに応じて、サービスを作成する手順に微妙な違いがあります。このトピックでは、状況に適したものに従うことができるように、これらのソースタイプを作成するさまざまな手順について説明します。コード例を使用して基本的な手順を開始するには、「」を参照してください開始方法。

前提条件

App Runner サービスを作成する前に、必ず次のアクションを実行してください。

- のセットアップ手順を完了します設定。
- アプリケーションソースの準備が整っていることを確認します。GitHub のコードリポジトリ、Bitbucket、または Amazon Elastic Container Registry (Amazon ECR) のコンテナイメージのいずれかを使用して、App Runner サービスを作成できます。

サービスを作成する

このセクションでは、ソースコードに基づく とコンテナイメージに基づく 2 つの App Runner サービスタイプの作成プロセスについて説明します。

> **Note**
>
> サービスのアウトバウンドトラフィック VPC コネクタを作成すると、次のサービス起動プロセスで 1 回限りのレイテンシーが発生します。この設定は、新しいサービスの作成時または作成後にサービスの更新で設定できます。詳細については、このガイド1 回限りのレイテンシーの「Networking with App Runner」の章の「」を参照してください。

コードリポジトリからサービスを作成する

以下のセクションでは、ソースが GitHub または Bitbucket のコードリポジトリである場合に App Runner サービスを作成する方法を示します。コードリポジトリを使用する場合、App Runner はプロバイダーの組織またはアカウントに接続する必要があります。したがって、この接続の確立を支援する必要があります。App Runner 接続の詳細については、「」を参照してくださいthe section called “Connections”。

サービスを作成すると、App Runner はアプリケーションコードと依存関係を含む Docker イメージを構築します。次に、このイメージのコンテナインスタンスを実行するサービスを起動します。

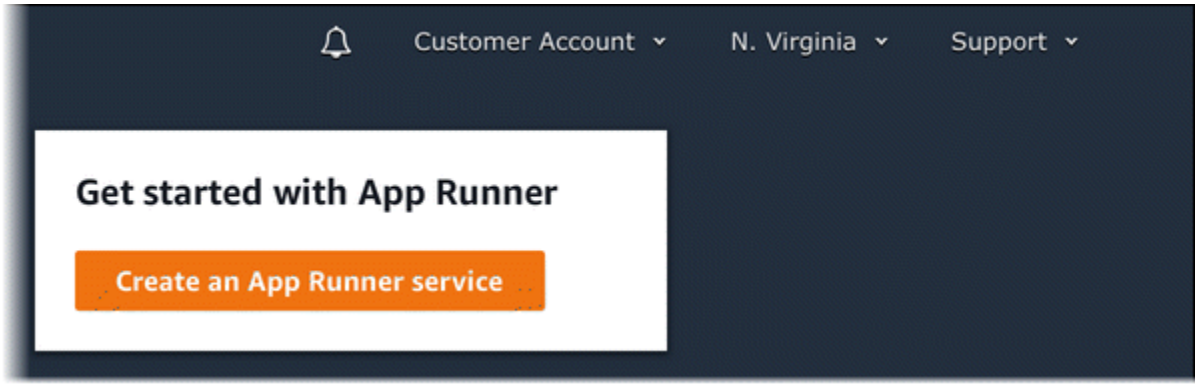
App Runner コンソールを使用してコードからサービスを作成する

コンソールを使用して App Runner サービスを作成するには

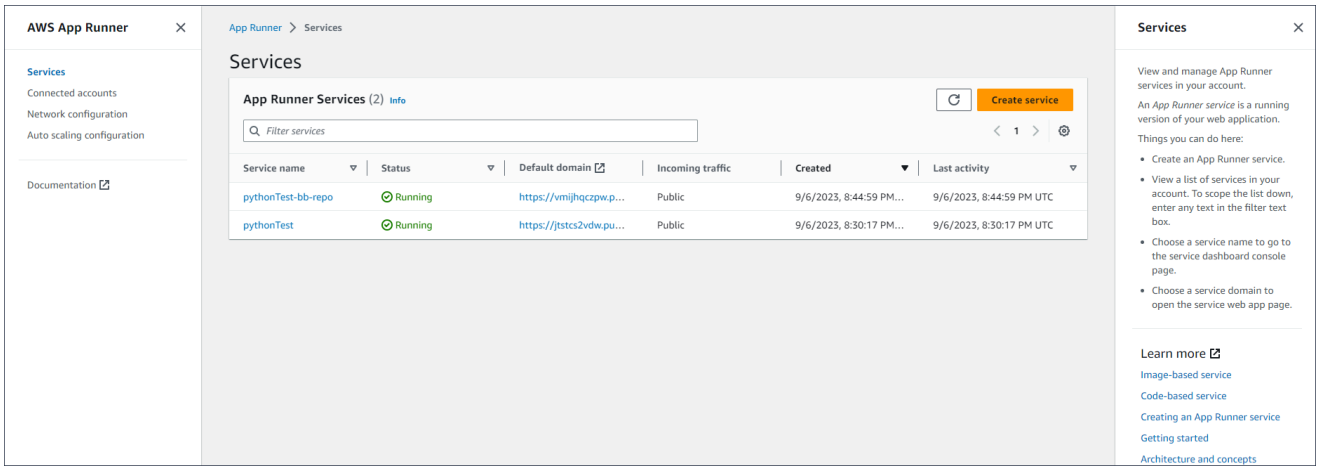
1. ソースコードを設定します。

   a. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。

   b. AWS アカウント に App Runner サービスがまだない場合は、コンソールのホームページが表示されます。App Runner サービスの作成を選択します。



   に既存のサービス AWS アカウント がある場合、サービスのリストを含むサービスページが表示されます。[Create service (サービスの作成)] を選択します。



   c. 「ソースとデプロイ」ページの「ソース」セクションの「リポジトリタイプ」で、「ソースコードリポジトリ」を選択します。

   d. プロバイダータイプを選択します。GitHub または Bitbucket を選択します。

   e. 次に、以前に使用したプロバイダーのアカウントまたは組織を選択するか、新規追加を選択します。次に、コードリポジトリの認証情報を提供し、接続するアカウントまたは組織を選択するプロセスを実行します。

f. リポジトリで、アプリケーションコードを含むリポジトリを選択します。

g. Branch で、デプロイするブランチを選択します。

h. Source ディレクトリには、アプリケーションコードと設定ファイルを保存するソースリポジトリに ディレクトリを入力します。

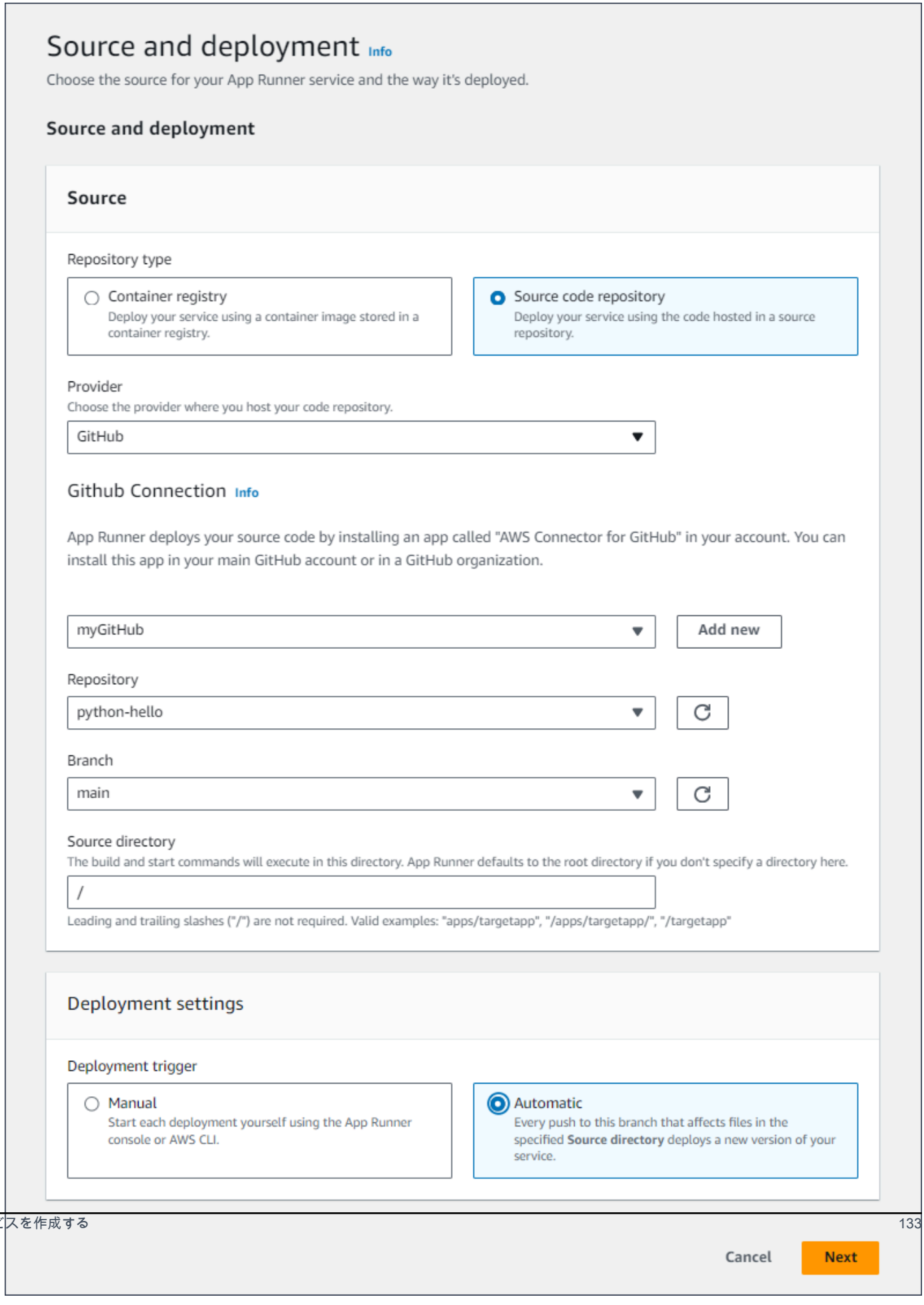
> **Note**
>
> ビルドコマンドとスタートコマンドは、指定したソースディレクトリから実行されます。App Runner はルートからの絶対パスとして処理します。ここで値を指定しない場合、ディレクトリはデフォルトでリポジトリルートになります。

2. デプロイを設定します。

a. デプロイ設定セクションで、手動または自動を選択します。

デプロイ方法の詳細については、「」を参照してくださいthe section called “デプロイ方法”。

b. [次へ] を選択します。


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source and deployment
Source
Repository type
Container registry
Deploy your service using a container image stored in a container registry.
Source code repository
Deploy your service using the code hosted in a source repository.
Provider
Choose the provider where you host your code repository.
GitHub
Github Connection Info
App Runner deploys your source code by installing an app called "AWS Connector for GitHub" in your account. You can install this app in your main GitHub account or in a GitHub organization.
myGitHub
Add new
Repository
python-hello
Branch
main
Source directory
The build and start commands will execute in this directory. App Runner defaults to the root directory if you don't specify a directory here.
/
Leading and trailing slashes ("/") are not required. Valid examples: "apps/targetapp", "/apps/targetapp/", "/targetapp"
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
Every push to this branch that affects files in the specified Source directory deploys a new version of your service.
Cancel
Next

3. アプリケーションビルドを設定します。

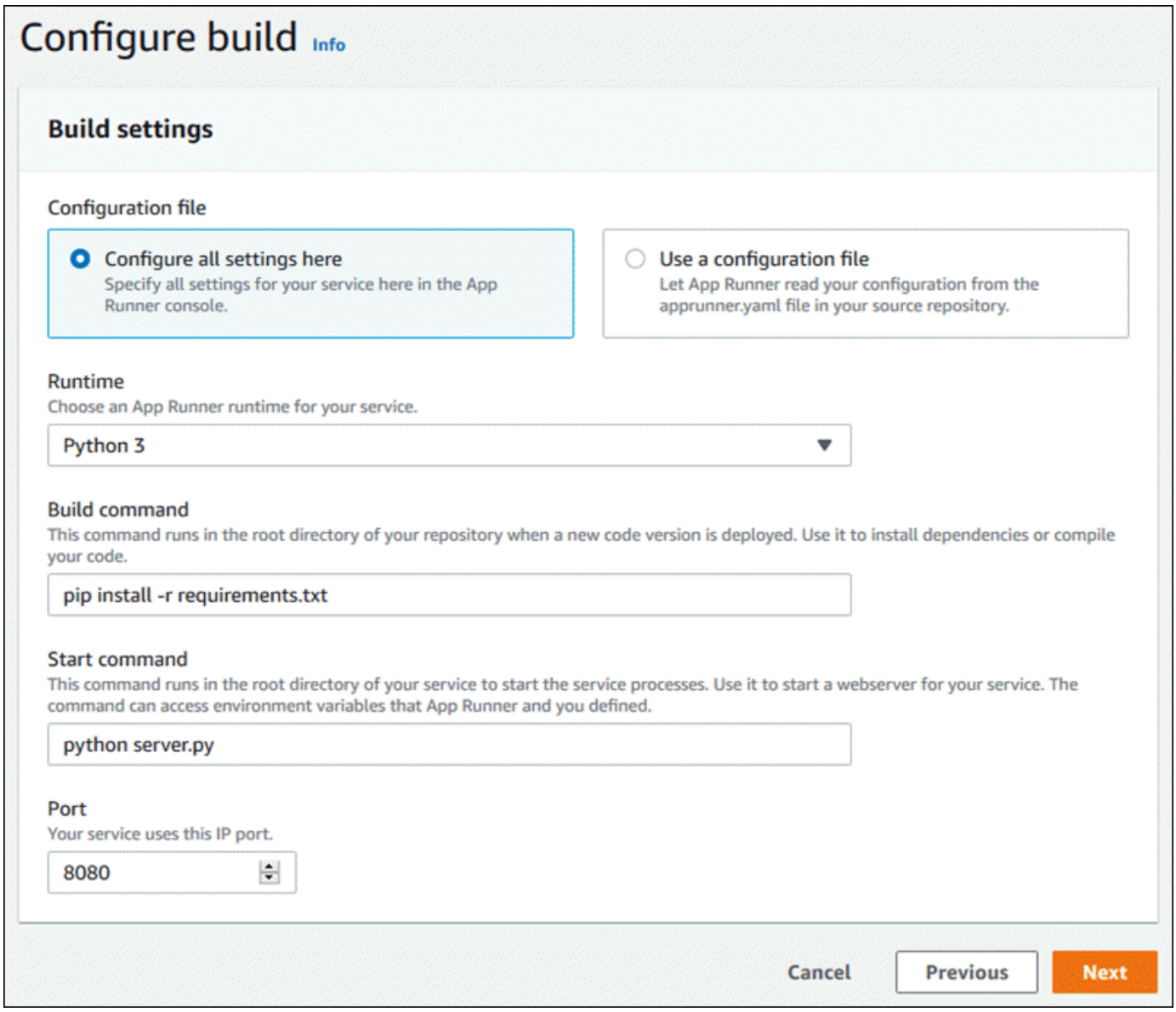
   a. 「ビルドの設定」ページの「設定ファイル」で、リポジトリに App Runner 設定ファイルが含まれていない場合は、ここですべての設定を設定する」を選択するか、含まれている場合は設定ファイルを使用するを選択します。

   > **Note**
   >
   > App Runner 設定ファイルは、アプリケーションソースの一部としてビルド設定を維持する方法です。指定すると、App Runner はファイルからいくつかの値を読み取り、コンソールで設定することはできません。

   b. 次のビルド設定を指定します。

   - ランタイム – アプリケーションの特定のマネージドランタイムを選択します。
   - ビルドコマンド – ソースコードからアプリケーションを構築するコマンドを入力します。これは、言語固有のツールでも、コードに付属のスクリプトでもかまいません。
   - Start command – ウェブサービスを開始するコマンドを入力します。
   - ポート – ウェブサービスがリッスンする IP ポートを入力します。

   c. [次へ] を選択します。



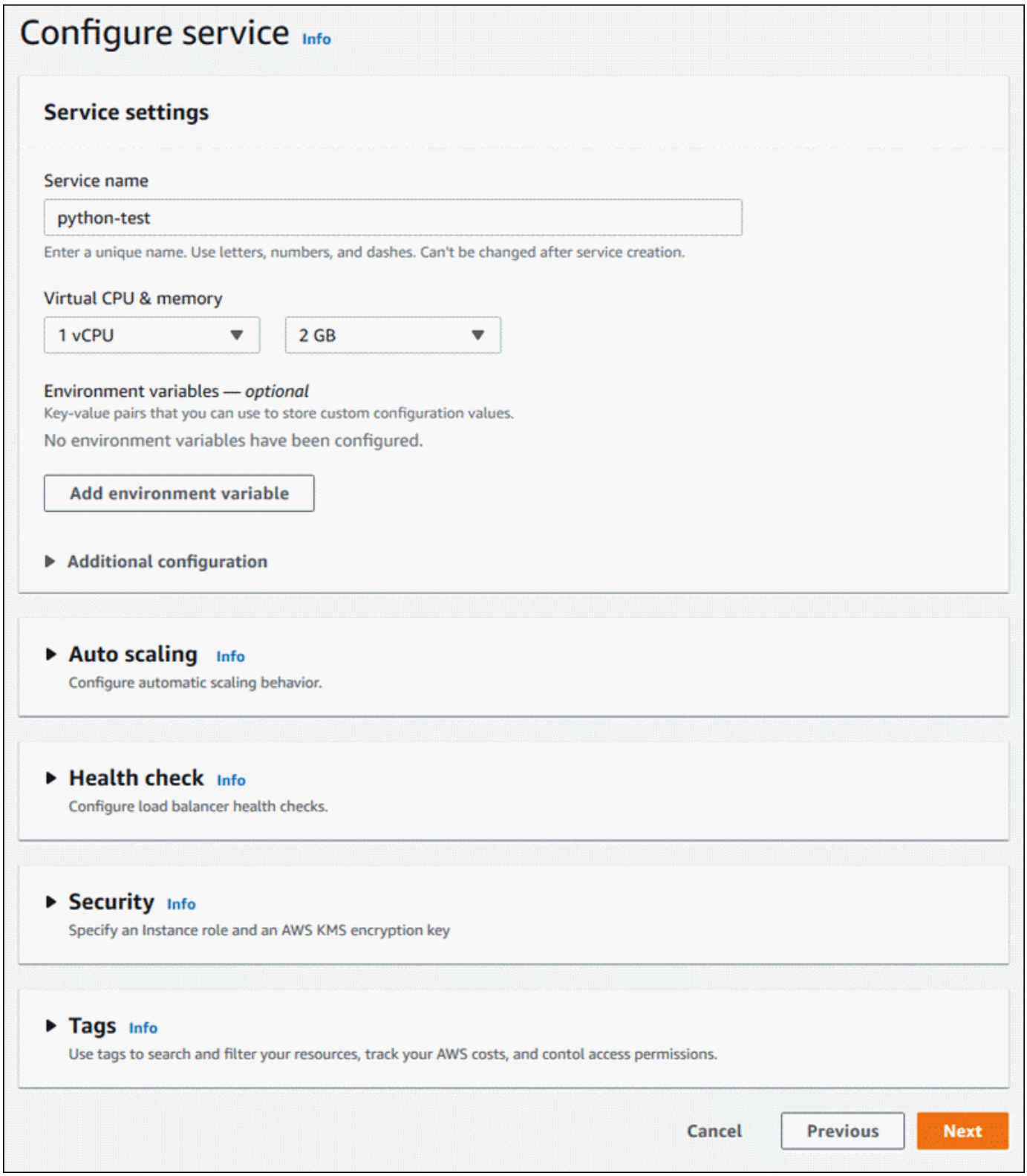
4. サービスを設定します。

   a. 「サービスの設定」ページの「サービス設定」セクションに、サービス名を入力します。

   > **Note**
   >
   > 他のすべてのサービス設定はオプションであるか、コンソールが提供するデフォルトです。

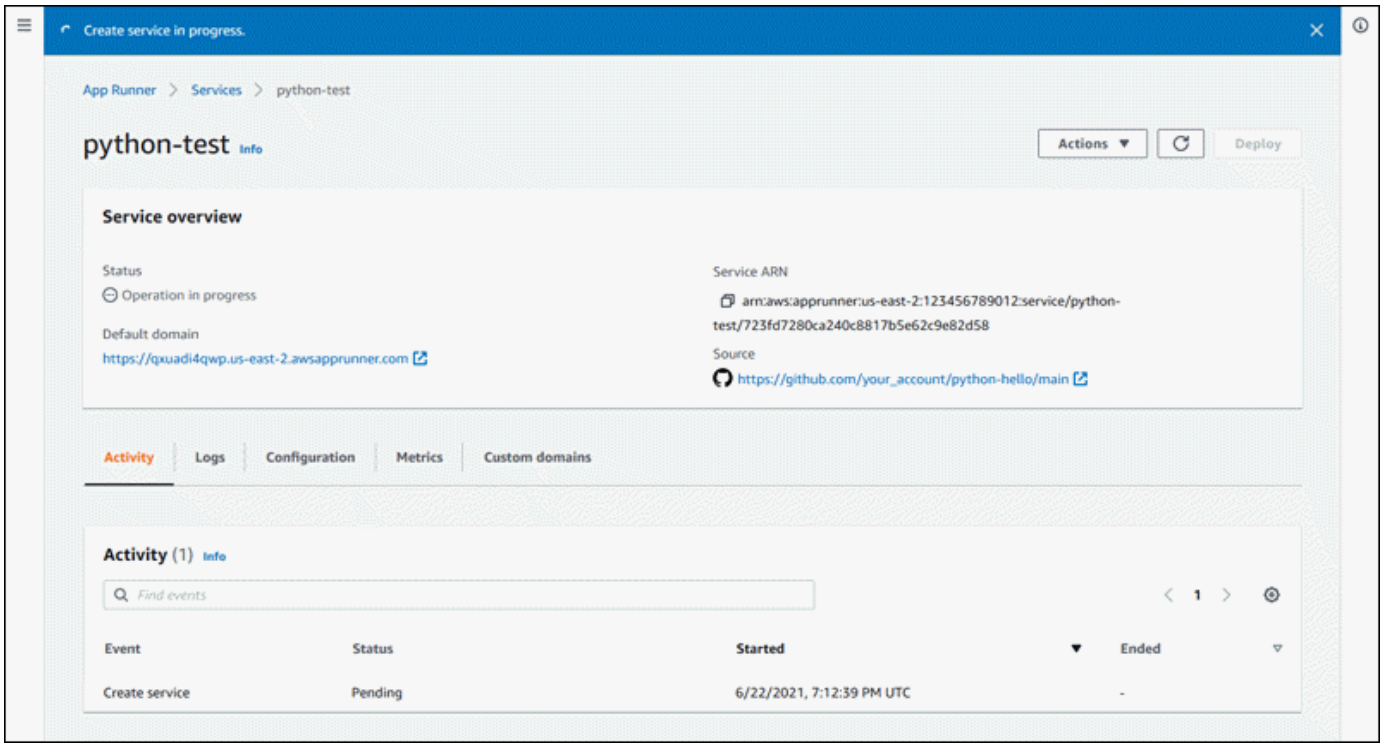
   b. 必要に応じて、アプリケーション要件を満たすように他の設定を変更または追加します。

c. [次へ] を選択します。


Configure service Info
Service settings
Service name
python-test
Enter a unique name. Use letters, numbers, and dashes. Can't be changed after service creation.
Virtual CPU & memory
1 vCPU
2 GB
Environment variables — optional
Key-value pairs that you can use to store custom configuration values.
No environment variables have been configured.
Add environment variable
Additional configuration
Auto scaling Info
Configure automatic scaling behavior.
Health check Info
Configure load balancer health checks.
Security Info
Specify an Instance role and an AWS KMS encryption key
Tags Info
Use tags to search and filter your resources, track your AWS costs, and contol access permissions.
Cancel
Previous
Next

5. 確認と作成ページで、入力したすべての詳細を確認し、作成とデプロイを選択します。

   結果: サービスが正常に作成されると、コンソールにサービスダッシュボードに新しいサービスの概要が表示されます。



6. サービスが実行されていることを確認します。

   a. サービスダッシュボードページで、サービスのステータスが実行中になるまで待ちます。

   b. デフォルトのドメイン値を選択します。これは、サービスのウェブサイトへの URL です。

   c. ウェブサイトを使用して、正しく実行されていることを確認します。

App Runner API または を使用してコードからサービスを作成する AWS CLI

App Runner API または を使用してサービスを作成するには AWS CLI、`CreateService` API アクションを呼び出します。詳細と例については、「CreateService」を参照してください。ソースコードリポジトリ (GitHub または Bitbucket) の特定の組織またはアカウントを使用してサービスを作成するのが初めての場合は、まず CreateConnection を呼び出します。これにより、App Runner とリポジトリプロバイダーの組織またはアカウント間の接続が確立されます。App Runner 接続の詳細については、「」を参照してくださいthe section called “Connections”。

呼び出しが を示す Service オブジェクトで成功したレスポンスを返すと`"Status": "CREATING"`、サービスは作成を開始します。

呼び出しの例については、 AWS App Runner API リファレンスの「ソースコードリポジトリサービスの作成」を参照してください。

Amazon ECR イメージからサービスを作成する

以下のセクションでは、ソースが Amazon ECR に保存されているコンテナイメージである場合に App Runner サービスを作成する方法を示します。Amazon ECR は です AWS のサービス。したがって、Amazon ECR イメージに基づいてサービスを作成するには、必要な Amazon ECR アクションのアクセス許可を含むアクセスロールを App Runner に提供します。

> **Note**
>
> Amazon ECR Public に保存されているイメージは公開されています。したがって、イメージが Amazon ECR Public に保存されている場合、アクセスロールは必要ありません。

サービスが作成されると、App Runner は、指定したイメージのコンテナインスタンスを実行するサービスを起動します。この場合、ビルドフェーズはありません。

詳細については、「イメージベースのサービス」を参照してください。

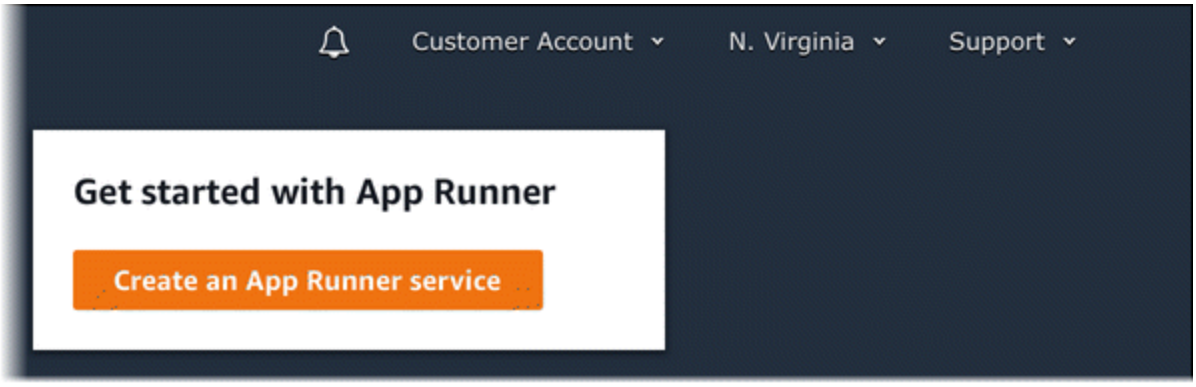
App Runner コンソールを使用してイメージからサービスを作成する

コンソールを使用して App Runner サービスを作成するには

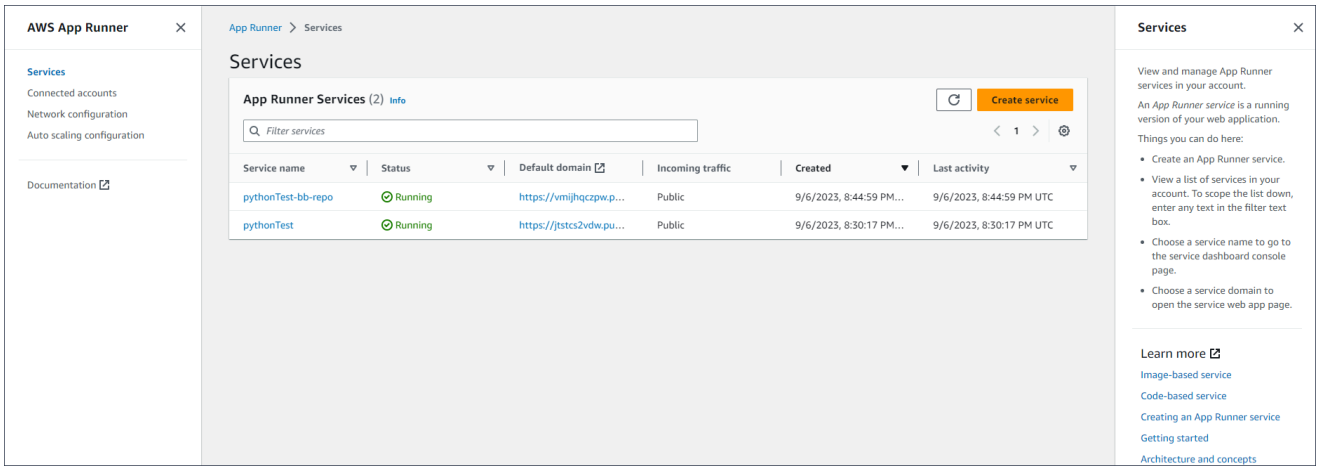
1. ソースコードを設定します。

   a. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。

   b. AWS アカウント に App Runner サービスがまだない場合は、コンソールのホームページが表示されます。App Runner サービスの作成を選択します。



に既存のサービス AWS アカウント がある場合、サービスのリストを含むサービスページが表示されます。[Create service (サービスの作成)] を選択します。



c. 「ソースとデプロイ」ページの「ソース」セクションの「リポジトリタイプ」で、「コンテナレジストリ」を選択します。

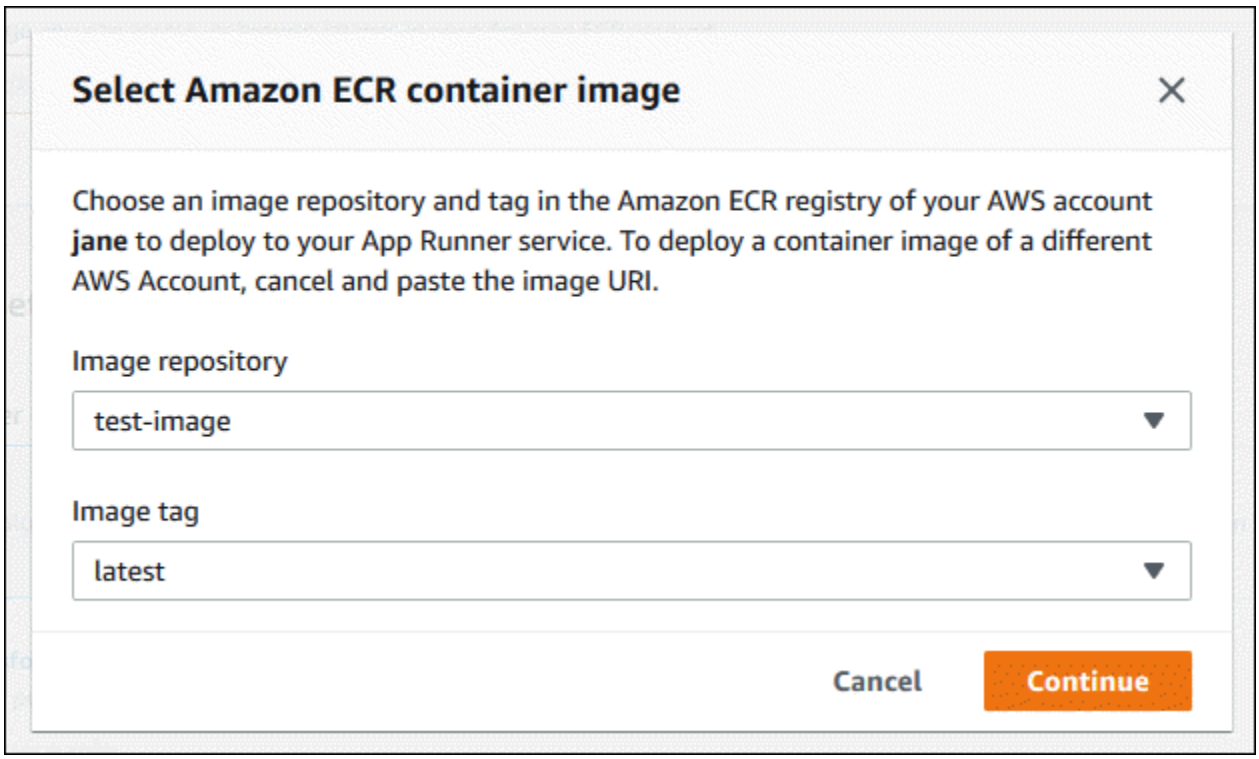
d. Provider で、イメージが保存されているプロバイダーを選択します。

- Amazon ECR – Amazon ECR に保存されているプライベートイメージ。
- Amazon ECR Public – Amazon ECR Public に保存されているパブリックに読み取り可能なイメージ。

e. コンテナイメージ URI で、参照 を選択します。

f. Amazon ECR コンテナイメージの選択ダイアログボックスのイメージリポジトリで、イメージを含むリポジトリを選択します。

g. イメージタグで、デプロイする特定のイメージタグ (最新など) を選択し、続行を選択します。



2. デプロイを設定します。

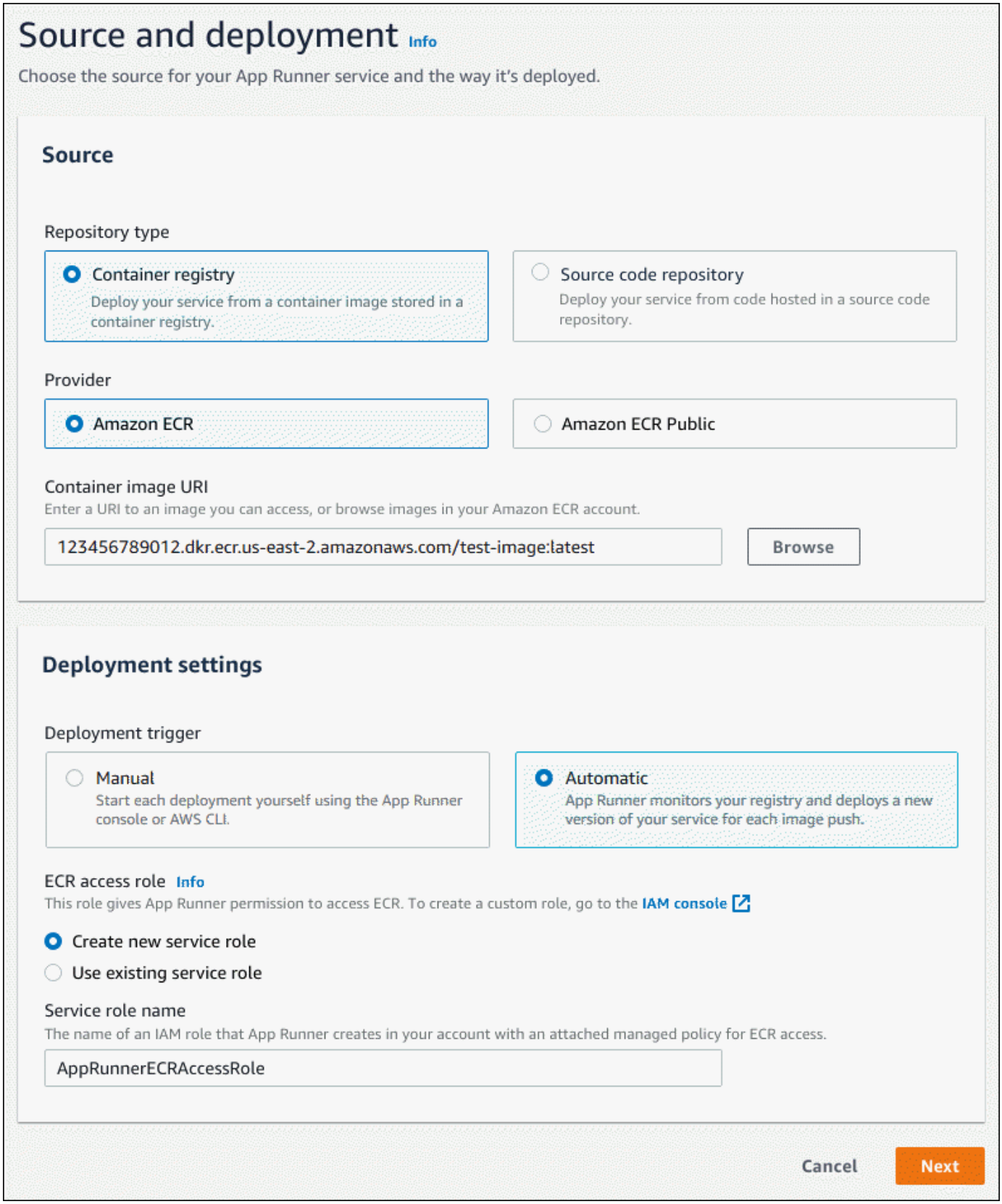
   a. デプロイ設定セクションで、手動または自動を選択します。

   > **Note**
   >
   > App Runner は、Amazon ECR Public イメージ、およびサービスが存在するアカウントとは異なる AWS アカウントに属する Amazon ECR リポジトリ内のイメージの自動デプロイをサポートしていません。

   デプロイ方法の詳細については、「」を参照してくださいthe section called “デプロイ方法”。

   b. [Amazon ECR プロバイダー] ECR アクセスロールの場合は、アカウント内の既存のサービスロールを選択するか、新しいロールの作成を選択します。手動デプロイを使用している場合は、デプロイ時に IAM ユーザーロールを使用することを選択することもできます。

   c. [次へ] を選択します。


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source
Repository type
Container registry
Deploy your service from a container image stored in a container registry.
Source code repository
Deploy your service from code hosted in a source code repository.
Provider
Amazon ECR
Amazon ECR Public
Container image URI
Enter a URI to an image you can access, or browse images in your Amazon ECR account.
123456789012.dkr.ecr.us-east-2.amazonaws.com/test-image:latest
Browse
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
App Runner monitors your registry and deploys a new version of your service for each image push.
ECR access role Info
This role gives App Runner permission to access ECR. To create a custom role, go to the IAM console
Create new service role
Use existing service role
Service role name
The name of an IAM role that App Runner creates in your account with an attached managed policy for ECR access.
AppRunnerECRAccessRole
Cancel
Next

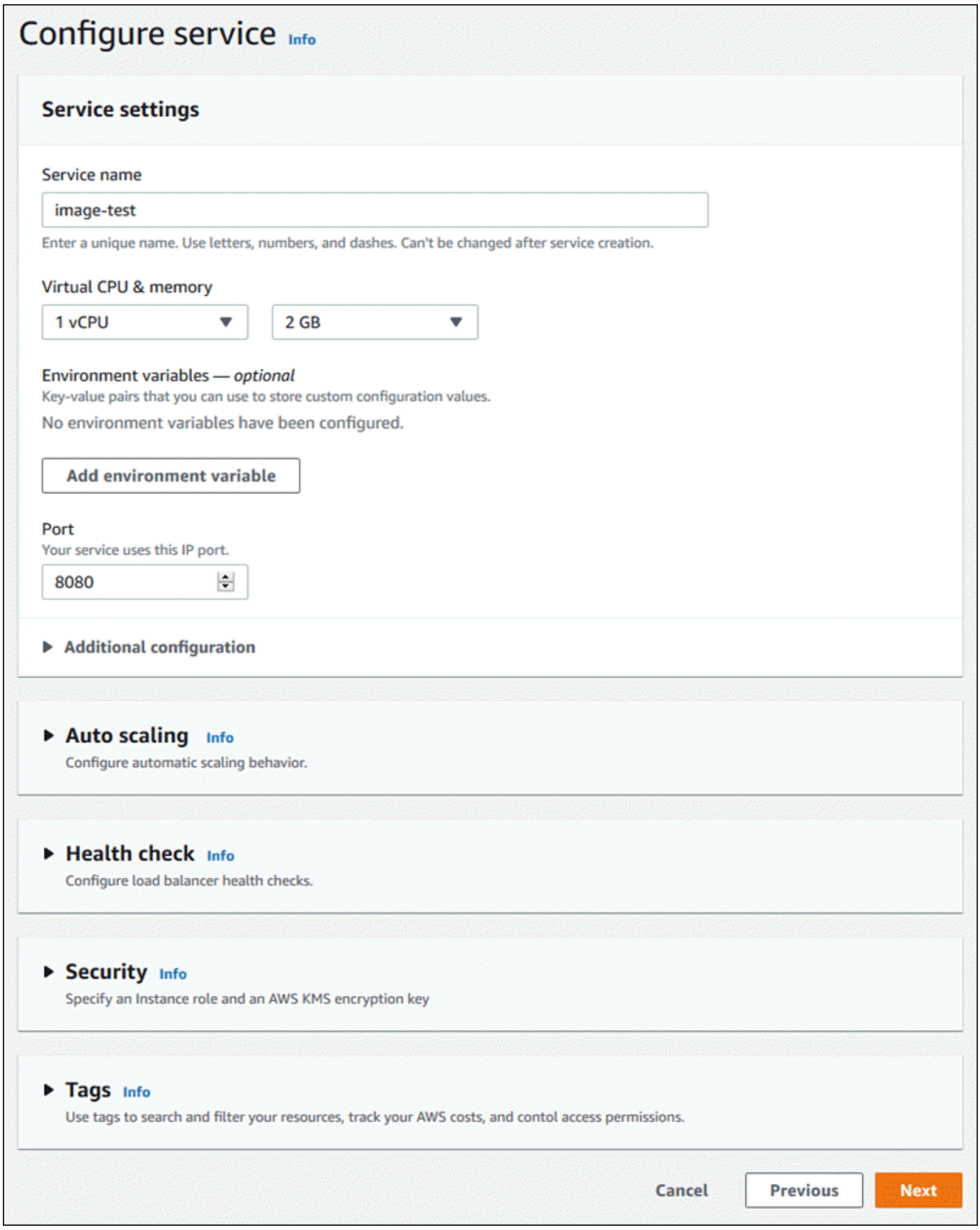
3. サービスを設定します。

   a. 「サービスの設定」ページの「サービス設定」セクションに、サービス名とサービスウェブサイトがリッスンする IP ポートを入力します。

      > **Note**
      >
      > 他のすべてのサービス設定はオプションであるか、コンソールが提供するデフォルトです。

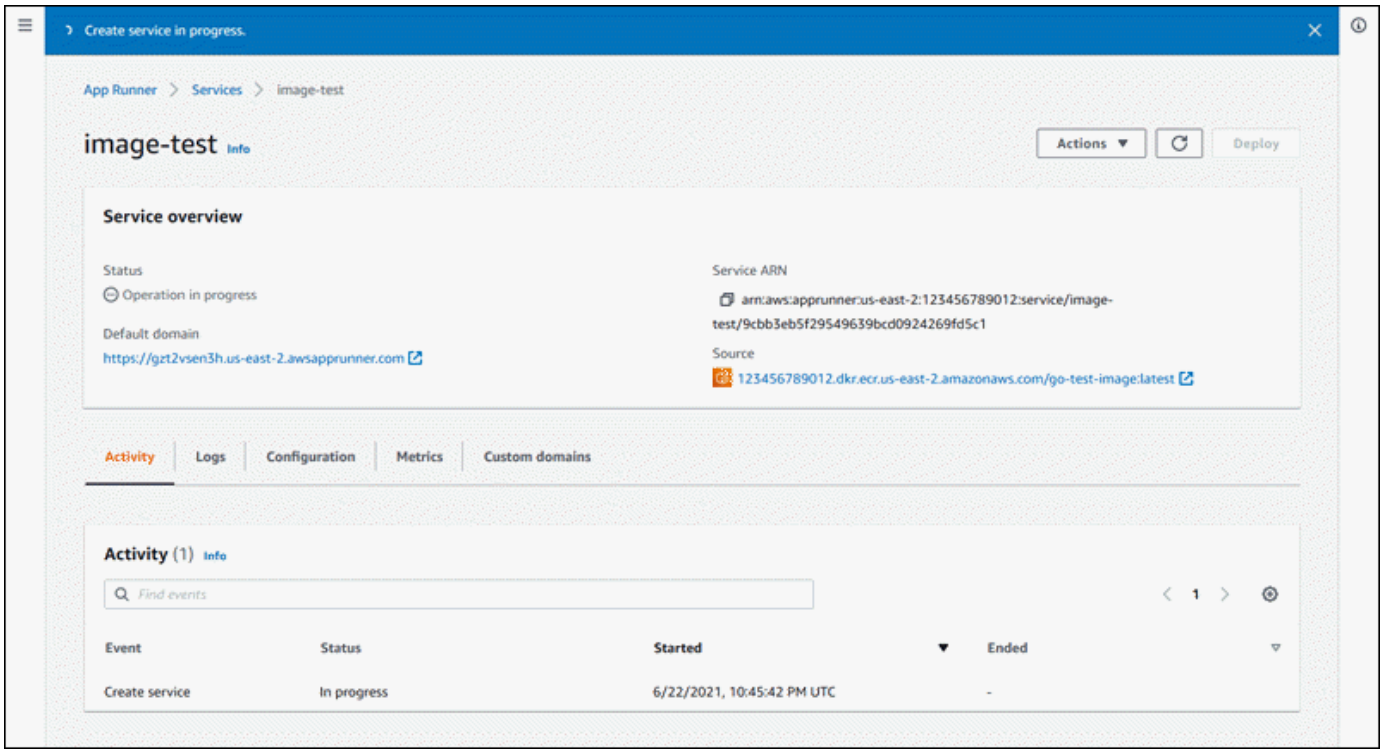
   b. (オプション) アプリケーションのニーズに応じて、他の設定を変更または追加します。

   c. [次へ] を選択します。


Configure service Info
Service settings
Service name
image-test
Enter a unique name. Use letters, numbers, and dashes. Can't be changed after service creation.
Virtual CPU & memory
1 vCPU
2 GB
Environment variables — optional
Key-value pairs that you can use to store custom configuration values.
No environment variables have been configured.
Add environment variable
Port
Your service uses this IP port.
8080
Additional configuration
Auto scaling Info
Configure automatic scaling behavior.
Health check Info
Configure load balancer health checks.
Security Info
Specify an Instance role and an AWS KMS encryption key
Tags Info
Use tags to search and filter your resources, track your AWS costs, and contol access permissions.
Cancel
Previous
Next

4. 確認と作成ページで、入力したすべての詳細を確認し、作成とデプロイを選択します。

   結果: サービスが正常に作成されると、コンソールにサービスダッシュボードが表示され、新しいサービスの概要が表示されます。



5. サービスが実行されていることを確認します。

   a. サービスダッシュボードページで、サービスのステータスが実行中になるまで待ちます。

   b. デフォルトのドメイン値を選択します。これは、サービスのウェブサイトへの URL です。

   c. ウェブサイトを使用して、正しく実行されていることを確認します。

App Runner API または を使用してイメージからサービスを作成する AWS CLI

App Runner API または を使用してサービスを作成するには AWS CLI、CreateService API アクションを呼び出します。

サービスの作成は、呼び出しが を示すサービスオブジェクトで成功したレスポンスを返したときに開始されます`"Status": "CREATING"`。

呼び出しの例については、 AWS App Runner API リファレンスの「ソースイメージリポジトリサービスの作成」を参照してください。

失敗した App Runner サービスの再構築

App Runner サービスの作成時に作成失敗エラーが表示された場合は、次のいずれかを実行できます。

- の手順に従って、エラーの原因the section called “サービスの作成に失敗しました”を特定します。
- ソースまたは設定でエラーが見つかった場合は、必要な変更を加えてからサービスを再構築します。
- App Runner の一時的な問題によりサービスが失敗した場合、ソースまたは設定を変更せずに、失敗したサービスを再構築します。

失敗したサービスは、App Runner コンソールまたは App Runner API または AWS CLI を使用して再構築できます。

App Runner コンソールを使用した失敗した App Runner サービスの再構築

Rebuild with updates

サービスの作成は、さまざまな理由で失敗する可能性があります。この場合、サービスを再構築する前に、問題の根本原因を特定して修正することが重要です。詳細については、「the section called “サービスの作成に失敗しました”」を参照してください。

更新で失敗したサービスを再構築するには

1. サービスページの 設定 タブに移動し、編集 を選択します。

   このページで概要パネルが開き、すべての更新のリストが表示されます。
2. 必要な変更を行い、概要パネルで確認します。
3. 保存して再構築を選択します。

   サービスページのログタブで進行状況をモニタリングできます。

Rebuild without updates

一時的な問題によりサービスの作成が失敗した場合、ソースまたは設定を変更せずにサービスを再構築できます。

更新なしで失敗したサービスを再構築するには

- サービスページの右上隅にある再構築を選択します。

  サービスページのログタブで進行状況をモニタリングできます。
- サービスを再度作成できない場合は、「」のトラブルシューティング手順に従ってくださいthe section called “サービスの作成に失敗しました”。必要な変更を行い、サービスを再構築します。

App Runner API または を使用した失敗した App Runner サービスの再構築 AWS CLI

Rebuild with updates

失敗したサービスを再構築するには:

1. the section called “サービスの作成に失敗しました”「」の手順に従って、エラーの原因を見つけます。
2. ブランチ、ソースリポジトリのイメージ、またはエラーの原因となった設定に必要な変更を加えます。
3. 新しいソースコードリポジトリまたはソースイメージリポジトリパラメータを使用してUpdateService API アクションを呼び出して再構築します。App Runner は、ソースコードリポジトリから最新のコミットを取得します。

Example更新による再構築

次の例では、イメージベースのサービスのソース設定が更新されています。の値は に変更Portされます80。

イメージベースの App Runner サービスの `input.json` ファイルの更新

```
{
  "ServiceArn": "arn:aws:apprunner:us-east-1:123456789012:service/python-
app/8fe1e10304f84fd2b0df550fe98a71fa",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageConfiguration": {
        "Port": "80"
      }
```

```
    }
  }
}
```

`UpdateService` API アクションを呼び出します。

```
aws apprunner update-service
--cli-input-json file://input.json
```

Rebuild without updates

App Runner API または を使用して失敗したサービスを再構築するには AWS CLI、サービスのソースまたは設定を変更せずに UpdateService API アクションを呼び出します。App Runner の一時的な問題によりサービスの作成が失敗した場合にのみ、更新せずに再構築することを選択します。

App Runner への新しいアプリケーションバージョンのデプロイ

でサービスを作成するときは AWS App Runner、コンテナイメージまたはソースリポジトリなどのアプリケーションソースを設定します。App Runner は、サービスを実行するためのリソースをプロビジョニングし、そのリソースにアプリケーションをデプロイします。

このトピックでは、新しいバージョンが利用可能になったときにアプリケーションソースを App Runner サービスに再デプロイする方法について説明します。これは、イメージリポジトリの新しいイメージバージョンでも、コードリポジトリの新しいコミットでもかまいません。App Runner には、自動と手動の 2 つのデプロイ方法があります。

デプロイ方法

App Runner には、アプリケーションのデプロイの開始方法を制御するための以下の方法が用意されています。

自動デプロイ

サービスの継続的インテグレーションとデプロイ (CI/CD) 動作が必要な場合は、自動デプロイを使用します。App Runner は、イメージまたはコードリポジトリの変更をモニタリングします。

イメージリポジトリ – 新しいイメージバージョンをイメージリポジトリにプッシュするか、コードリポジトリに新しいコミットをプッシュするたびに、App Runner はユーザー側でそれ以上のアクションを行わずに、自動的にそれをサービスにデプロイします。

コードリポジトリ – ソースディレクトリに変更を加えるコードリポジトリに新しいコミットをプッシュするたびに、App Runner はリポジトリ全体をデプロイします。ソースディレクトリの変更のみが自動デプロイをトリガーするため、ソースディレクトリの場所が自動デプロイの範囲にどのように影響するかを理解することが重要です。

- 最上位ディレクトリ (リポジトリルート) – これは、サービスの作成時にソースディレクトリに設定されたデフォルト値です。ソースディレクトリがこの値に設定されている場合、リポジトリ全体がソースディレクトリ内にあることを意味します。したがって、ソースリポジトリにプッシュするすべてのコミットは、この場合はデプロイをトリガーします。
- リポジトリルートではないディレクトリパス (デフォルト以外) – ソースディレクトリ内でプッシュされた変更のみが自動デプロイをトリガーするため、ソースディレクトリにないリポジトリにプッシュされた変更は自動デプロイをトリガーしません。したがって、手動デプロイを使用して、ソースディレクトリの外部にプッシュする変更をデプロイする必要があります。

> **Note**
>
> App Runner は、Amazon ECR Public イメージ、およびサービスが存在するアカウントとは異なる AWS アカウントに属する Amazon ECR リポジトリ内のイメージの自動デプロイをサポートしていません。

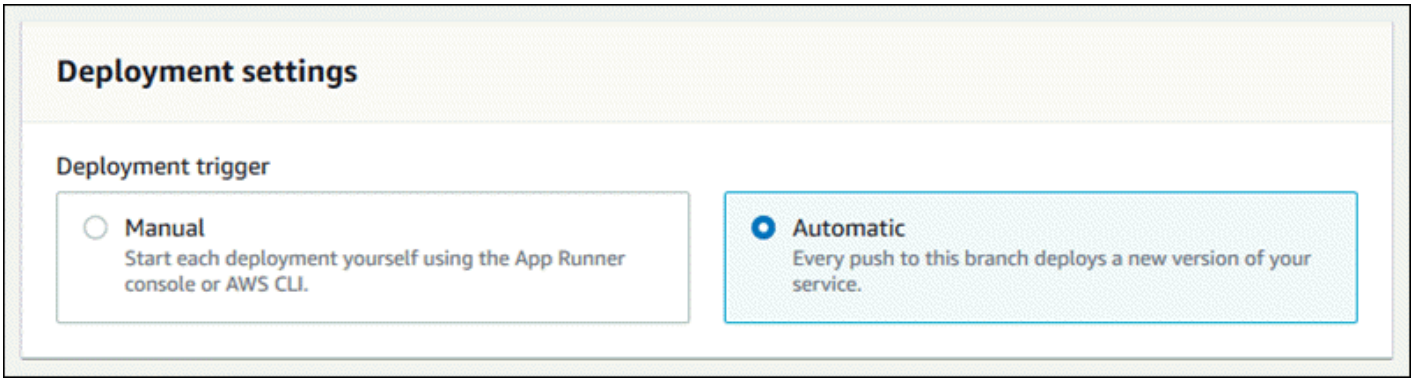
手動デプロイ

サービスへの各デプロイを明示的に開始する場合は、手動デプロイを使用します。サービス用に設定したリポジトリにデプロイする新しいバージョンがある場合、デプロイを開始します。詳細については、「the section called “手動デプロイ”」を参照してください。

> **Note**
>
> 手動デプロイを実行すると、App Runner は完全なリポジトリからソースをデプロイします。

サービスのデプロイ方法は、次の方法で設定できます。

- コンソール – 作成する新しいサービスまたは既存のサービスについては、ソースとデプロイ設定ページのデプロイ設定セクションで、手動または自動を選択します。



- API または AWS CLI – CreateService または UpdateService アクションの呼び出しで、手動デプロイFalseの場合は SourceConfiguration パラメータAutoDeploymentsEnabledのメンバーを に設定し、自動デプロイTrueの場合は に設定します。

> **自動デプロイと手動デプロイの比較**
>
> 自動デプロイと手動デプロイの両方が同じ結果になります。どちらの方法でもリポジトリ全体がデプロイされます。
> 2 つの方法の違いは、トリガーメカニズムです。
>
> - 手動デプロイは、コンソールからのデプロイ、 の呼び出し AWS CLI、または App Runner API の呼び出しによってトリガーされます。以下の手動デプロイセクションでは、これらの手順について説明します。
> - 自動デプロイは、ソースディレクトリの内容内の変更によってトリガーされます。

手動デプロイ

手動デプロイでは、サービスへの各デプロイを明示的に開始する必要があります。新しいバージョンのアプリケーションイメージまたはコードをデプロイする準備ができたら、以下のセクションを参照して、 コンソールと API を使用してデプロイを実行する方法を確認できます。

> **Note**
>
> 手動デプロイを実行すると、App Runner は完全なリポジトリからソースをデプロイします。

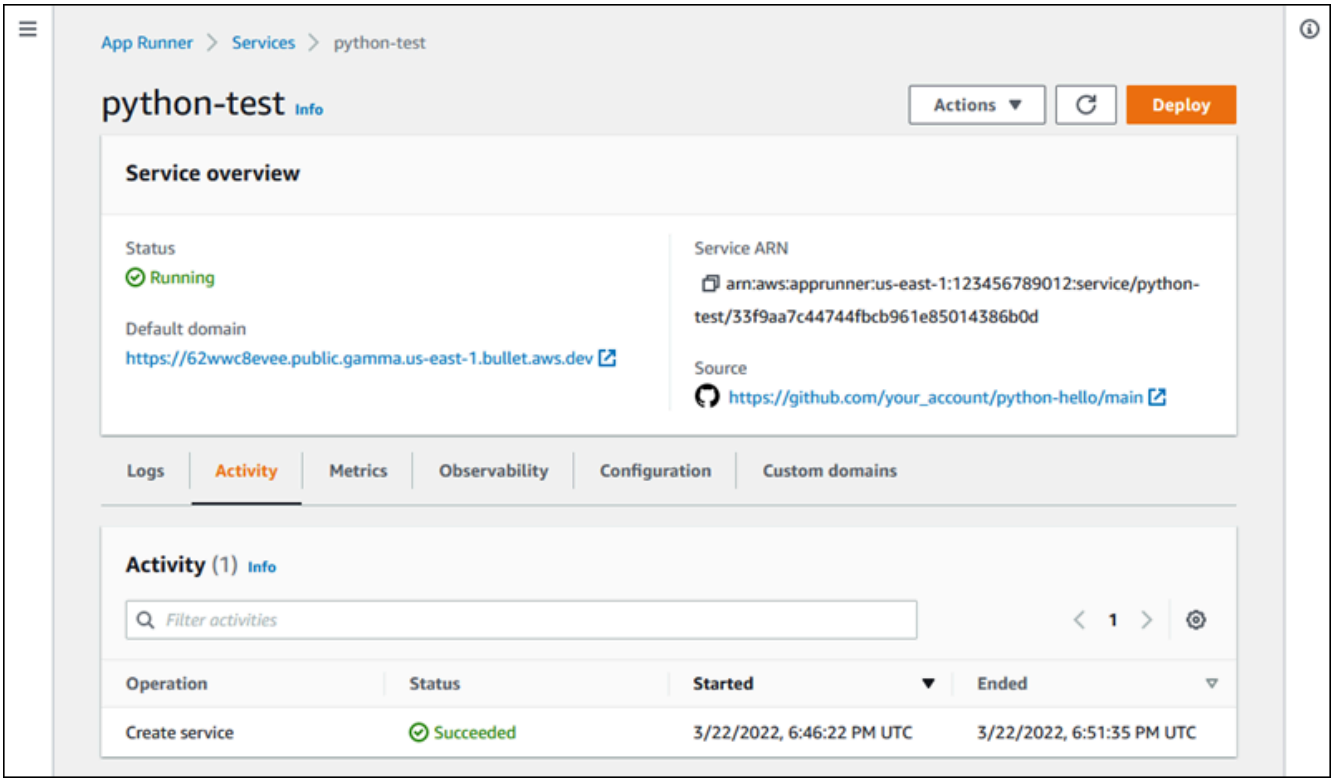
次のいずれかの方法を使用して、アプリケーションのバージョンをデプロイします。

App Runner console

App Runner コンソールを使用してデプロイするには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. [デプロイ] をクリックします。

   結果: 新しいバージョンのデプロイが開始されます。サービスダッシュボードページで、サービスのステータスが進行中のオペレーションに変わります。
4. デプロイが終了するのを待ちます。サービスダッシュボードページで、サービスのステータスが実行中に戻ります。
5. デプロイが成功したことを確認するには、サービスダッシュボードページで、デフォルトのドメイン値を選択します。これはサービスのウェブサイトへの URL です。ウェブアプリケーションを検査または操作し、バージョンの変更を確認します。

> **Note**
>
> App Runner アプリケーションのセキュリティを強化するために、*.awsapprunner.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化するために、App Runner アプリケーションのデフォルトのドメイン名で機密 Cookie を設定する必要がある場合は、__Host-プレフィックス付きの Cookie を使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

App Runner API or AWS CLI

App Runner API または を使用してデプロイするには AWS CLI、StartDeployment API アクションを呼び出します。渡す唯一のパラメータは、サービス ARN です。サービスの作成時にアプリケーションソースの場所をすでに設定していると、App Runner は新しいバージョンを見つけることができます。呼び出しが成功したレスポンスを返すと、デプロイが開始されます。

App Runner サービスの設定

AWS App Runner サービスを作成するときは、さまざまな設定値を設定します。これらの設定の一部は、サービスの作成後に変更できます。その他の設定は、サービスの作成中にのみ適用でき、それ以降は変更できません。このトピックでは、App Runner API、App Runner コンソール、および App Runner 設定ファイルを使用したサービスの設定について説明します。

トピック

- App Runner API または を使用してサービスを設定する AWS CLI
- App Runner コンソールを使用してサービスを設定する
- App Runner 設定ファイルを使用してサービスを設定する
- サービスのオブザーバビリティの設定
- 共有可能なリソースを使用したサービス設定の構成
- サービスのヘルスチェックの設定

App Runner API または を使用してサービスを設定する AWS CLI

API は、サービスの作成後に変更できる設定を定義します。次のリストでは、関連するアクション、タイプ、制限について説明します。

- UpdateService アクション – 作成後に呼び出して、一部の設定を更新できます。
  - 更新可能 – SourceConfiguration、、InstanceConfigurationおよび HealthCheckConfigurationパラメータの設定を更新できます。ただし、 でSourceConfigurationはなく、ソースタイプをコードからイメージ、またはその逆に切り替えることはできません。サービスの作成時に指定したのと同じリポジトリパラメータを指定する必要があります。これは CodeRepositoryまたは のいずれかですImageRepository。

    サービスに関連付けられた個別の設定リソースの次の ARNs を更新することもできます。

    - AutoScalingConfigurationArn
    - VpcConnectorArn
  - 更新不可 – CreateService アクションで使用できる ServiceNameおよび EncryptionConfigurationパラメータを変更することはできません。作成後に変更することはできません。UpdateService アクションには、これらのパラメータは含まれません。
  - API と ファイル – CodeConfiguration タイプの ConfigurationSourceパラメータ (の一部としてソースコードリポジトリに使用SourceConfiguration) を に設定できますRepository。この場合、App Runner は の設定を無視しCodeConfigurationValues、リポジトリの設定ファイルからこれらの設定を読み取ります。ConfigurationSource を に設定するとAPI、App Runner は API コールからすべての設定を取得し、設定ファイルが存在する場合でも無視します。
- TagResource アクション – サービスの作成後に呼び出して、サービスにタグを追加したり、既存のタグの値を更新したりできます。
- UntagResource アクション – サービスの作成後に呼び出して、サービスからタグを削除できます。

> **Note**
>
> サービスのアウトバウンドトラフィック VPC コネクタを作成すると、次のサービス起動プロセスで 1 回限りのレイテンシーが発生します。この設定は、新しいサービスの作成時または作成後にサービスの更新で設定できます。詳細については、このガイド1 回限りのレイテンシーの「Networking with App Runner」の章の「」を参照してください。

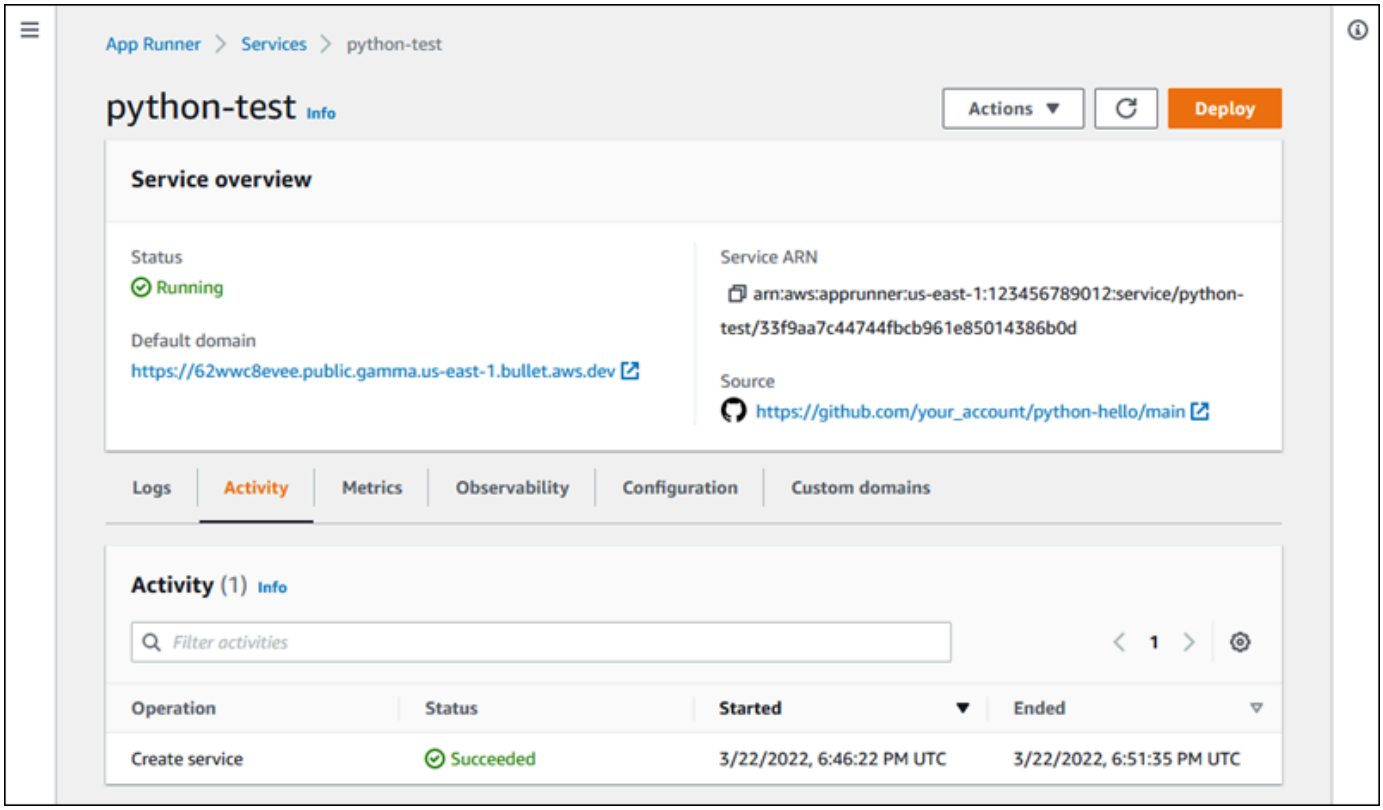
App Runner コンソールを使用してサービスを設定する

コンソールは App Runner API を使用して設定の更新を適用します。前のセクションで定義したように、API が課す更新ルールによって、 コンソールを使用して設定できる内容が決まります。サービスの作成時に使用可能な一部の設定は、後で変更することはできません。さらに、設定ファイルを使用する場合、追加の設定はコンソールで非表示になり、App Runner はファイルからそれらを読み取ります。

サービスを設定するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. サービスダッシュボードページで、設定タブを選択します。

   結果: コンソールには、ソースとデプロイ、ビルドの設定、サービスの設定の各セクションにサービスの現在の設定が表示されます。
4. 任意のカテゴリの設定を更新するには、編集 を選択します。
5. 設定の編集ページで、必要な変更を加え、変更の保存を選択します。

> **Note**
>
> サービスのアウトバウンドトラフィック VPC コネクタを作成すると、次のサービス起動プロセスで 1 回限りのレイテンシーが発生します。この設定は、新しいサービスの作成時または作成後にサービスの更新で設定できます。詳細については、このガイド1 回限りのレイテンシーの「Networking with App Runner」の章の「」を参照してください。

App Runner 設定ファイルを使用してサービスを設定する

App Runner サービスを作成または更新するときに、ソースリポジトリの一部として指定した設定ファイルから一部の設定を読み取るように App Runner に指示できます。これにより、ソース管理下にあるソースコードに関連する設定を、コード自体とともに管理できます。設定ファイルには、コンソールまたは API を使用して設定できない特定の詳細設定もあります。詳細については、「App Runner 設定ファイル」を参照してください。

> **Note**
>
> サービスのアウトバウンドトラフィック VPC コネクタを作成すると、次のサービス起動プロセスで 1 回限りのレイテンシーが発生します。この設定は、新しいサービスの作成時または作成後にサービスの更新で設定できます。詳細については、このガイド1 回限りのレイテンシーの「Networking with App Runner」の章の「」を参照してください。

サービスのオブザーバビリティの設定

AWS App Runner は複数の AWS サービスと統合され、App Runner サービス用の広範なオブザーバビリティツールスイートを提供します。詳細については、「オブザーバビリティ」を参照してください。

App Runner は、オブザーバビリティ機能を有効にし、ObservabilityConfiguration と呼ばれる共有可能なリソースを使用して動作を設定することをサポートしています。サービスを作成または更新するときに、オブザーバビリティ設定リソースを指定できます。App Runner コンソールは、新しい App Runner サービスを作成するときに自動的に作成します。オブザーバビリティ設定の提供はオプションです。指定しない場合、App Runner はデフォルトのオブザーバビリティ設定を提供します。

1 つのオブザーバビリティ設定を複数の App Runner サービス間で共有して、オブザーバビリティ動作を同じにすることができます。詳細については、「the section called “設定リソース”」を参照してください。

オブザーバビリティ設定を使用して、次のオブザーバビリティ機能を設定できます。

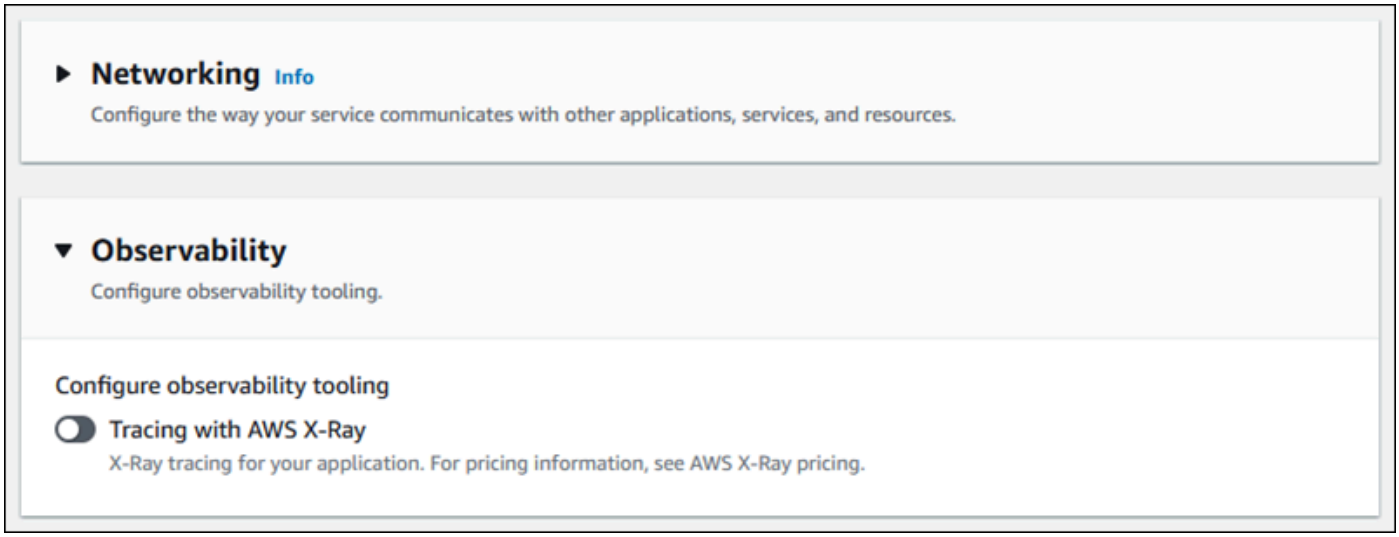
- トレース設定 – アプリケーションが処理するリクエストと、アプリケーションが行うダウンストリーム呼び出しをトレースするための設定。トレースの詳細については、「the section called “トレース (X-Ray)”」を参照してください。

オブザーバビリティの管理

次のいずれかの方法を使用して、App Runner サービスのオブザーバビリティを管理します。

App Runner console

App Runner コンソールを使用してサービスを作成する場合、または後でその設定を更新する場合は、サービスのオブザーバビリティ機能を設定できます。コンソールページのオブザーバビリティ設定セクションを探します。



App Runner API or AWS CLI

CreateService または UpdateService App Runner API アクションを呼び出すときに、`ObservabilityConfiguration`パラメータオブジェクトを使用してオブザーバビリティ機能を有効にし、サービスのオブザーバビリティ設定リソースを指定できます。

次の App Runner API アクションを使用して、オブザーバビリティ設定リソースを管理します。

- CreateObservabilityConfiguration – 新しいオブザーバビリティ設定または既存のオブザーバビリティ設定へのリビジョンを作成します。
- ListObservabilityConfigurations – に関連付けられているオブザーバビリティ設定のリスト AWS アカウントと概要情報を返します。
- DescribeObservabilityConfiguration – オブザーバビリティ設定の完全な説明を返します。
- DeleteObservabilityConfiguration – オブザーバビリティ設定を削除します。特定のリビジョンまたは最新のアクティブなリビジョンを削除できます。のオブザーバビリティ設定クォータに達した場合は、不要なオブザーバビリティ設定を削除する必要がある場合があります AWS アカウント。

共有可能なリソースを使用したサービス設定の構成

一部の機能では、 AWS App Runner サービス間で設定を共有するのが理にかなっています。例えば、一連のサービスに同じ自動スケーリング動作を持たせたい場合があります。または、すべてのサービスに対して同じオブザーバビリティ設定が必要な場合があります。App Runner では、個別の共有可能なリソースを使用して設定を共有できます。機能の一連の設定を定義するリソースを作成し、この設定リソースの Amazon リソースネーム (ARN) を 1 つ以上の App Runner サービスに提供します。

App Runner は、以下の機能に共有可能な設定リソースを実装します。

- Auto scaling
- 可観測性
- VPC アクセス

これらの各機能のドキュメントページには、使用可能な設定と管理手順に関する情報が表示されます。

個別の設定リソースを使用する機能は、いくつかの設計特性と考慮事項を共有します。

- リビジョン – 一部の設定リソースにはリビジョンを含めることができます。自動スケーリングとオブザーバビリティは、リビジョンを使用する 2 つの設定リソースの例です。このような場合、各設定には名前と数値リビジョンがあります。設定の複数のリビジョンには、同じ名前と異なるリビジョン番号があります。シナリオごとに異なる設定名を使用できます。名前ごとに複数のリビジョンを追加して、特定のシナリオの設定を微調整できます。

名前で作成した最初の設定は、リビジョン番号 1 を取得します。同じ名前の後続の設定では、リビジョン番号が連続して取得されます (2 で始まる)。App Runner サービスは、特定の設定リビジョンまたは最新の設定リビジョンに関連付けることができます。

- 共有 – 1 つの設定リソースを複数の App Runner サービス間で共有できます。これは、これらのサービス全体で同じ設定を維持する場合に便利です。特に、リソースがリビジョンをサポートしている場合は、設定の最新リビジョンを使用するように複数のサービスを設定できます。これを行うには、リビジョンではなく、設定名のみを指定します。この方法で設定したサービスはいずれも、サービスの更新時に設定の更新を受け取ります。設定の変更の詳細については、「」を参照してくださいthe section called “設定”。
- リソース管理 – App Runner を使用して設定を作成および削除できます。設定を直接更新することはできません。代わりに、リビジョンをサポートするリソースの場合、既存の設定名に新しいリビジョンを作成して、設定を効果的に更新できます。

> **Note**
>
> 自動スケーリングでは、App Runner コンソールと App Runner API の両方を使用して設定と複数のリビジョンを作成できます。App Runner コンソールと App Runner API の両方が、設定とリビジョンを削除することもできます。詳細については、App Runner の自動スケーリングの管理を参照してください。
> オブザーバビリティ設定など、他の設定タイプでは、App Runner コンソールで作成できる設定は 1 つのリビジョンのみです。さらにリビジョンを作成し、設定を削除するには、App Runner API を使用する必要があります。

- リソースクォータ – 各設定リソースに対して設定できる一意の設定名とリビジョンの数にクォータが設定されています AWS リージョン。これらのクォータに達した場合は、さらに作成する前に、設定名またはそのリビジョンの一部を削除する必要があります。自動スケーリング設定リビジョンの場合は、App Runner コンソールまたは App Runner API を使用してリビジョンを削除できます。詳細については、App Runner の自動スケーリングの管理を参照してください。他のリソースを削除するには、App Runner API を使用する必要があります。クォータの詳細については、「the section called “App Runner リソースクォータ”」を参照してください。
- リソースコストなし – 設定リソースの作成に追加コストは発生しません。機能自体のコストが発生する場合があります (たとえば、X-Ray トレースを有効にすると通常の AWS X-Ray コストが課金されます)。ただし、App Runner サービスの機能を設定する App Runner 設定リソースのコストは発生しません。

サービスのヘルスチェックの設定

AWS App Runner は、ヘルスチェックを実行してサービスの状態をモニタリングします。デフォルトのヘルスチェックプロトコルは TCP です。App Runner は、サービスに割り当てられたドメインに ping を実行します。別の方法として、ヘルスチェックプロトコルを HTTP に設定することもできます。App Runner は、ウェブアプリケーションにヘルスチェック HTTP リクエストを送信します。

ヘルスチェックに関連する設定をいくつか設定できます。次の表に、ヘルスチェックの設定とそのデフォルト値を示します。

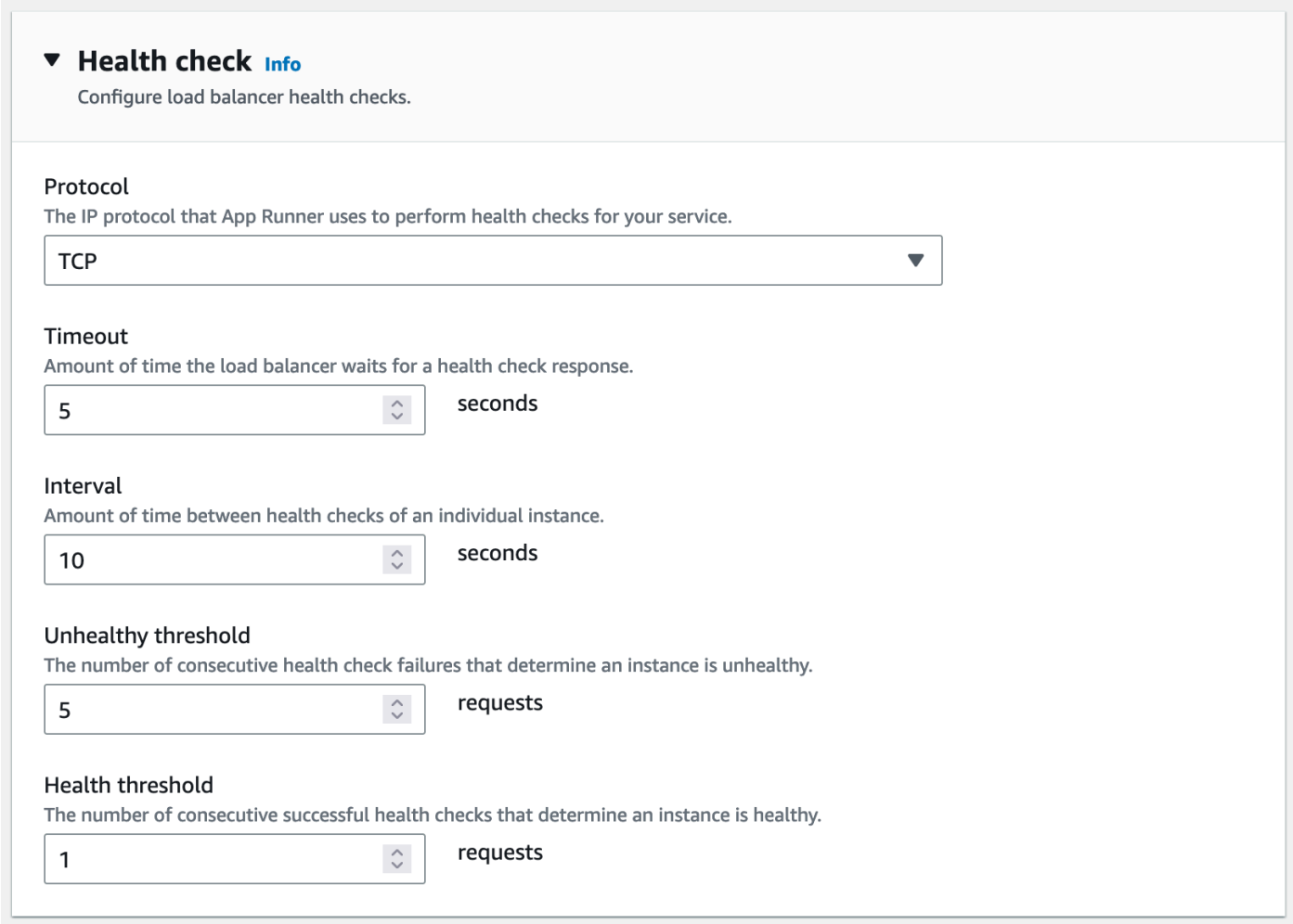
設定	説明	デフォルト
プロトコル	App Runner がサービスのヘルスチェックを実行するために使用する IP プロトコル。 プロトコルを に設定するとTCP、App Runner はアプリケーションがリッスンしているポートでサービスに割り当てられたデフォルトのドメインに ping を実行します。 プロトコルを に設定するとHTTP、App Runner は設定されたパスにヘルスチェックリクエストを送信します。	TCP
パス	App Runner が HTTP ヘルスチェックリクエストを送信する URL。HTTP チェックにのみ適用されます。	/
Interval	ヘルスチェックの間隔 (秒)。	5
タイムアウト	ヘルスチェックの応答が失敗したと判断されるまでの時間 (秒)。	2
正常なしきい値	App Runner がサービスが正常であると判断するために成功する必要がある連続チェックの数。	1
異常なしきい値	App Runner がサービスが異常であると判断するために失敗する必要がある連続チェックの数。	5

ヘルスチェックを設定する

次のいずれかの方法を使用して、App Runner サービスのヘルスチェックを設定します。

App Runner console

App Runner コンソールを使用して App Runner サービスを作成する場合、または後でその設定を更新する場合は、ヘルスチェック設定を設定できます。コンソールの詳細な手順については、the section called “作成”「」および「」を参照してくださいthe section called “設定”。いずれの場合も、コンソールページのヘルスチェック設定セクションを探します。



App Runner API or AWS CLI

CreateService または UpdateService API アクションを呼び出すときは、`HealthCheckConfiguration`パラメータを使用してヘルスチェック設定を指定できます。

パラメータの構造の詳細については、 AWS App Runner API リファレンスのHealthCheckConfiguration」を参照してください。

App Runner 接続の管理

でサービスを作成するときは AWS App Runner、アプリケーションソース、つまりプロバイダーに保存されているコンテナイメージまたはソースリポジトリを設定します。App Runner は、プロバイダーとの認証および認可された接続を確立する必要があります。その後、App Runner はリポジトリを読み取ってサービスにデプロイできます。App Runner は、 に保存されているコードにアクセスするサービスを作成するときに接続確立を必要としません AWS アカウント。

App Runner は、接続と呼ばれるリソースの接続情報を維持します。App Runner コンソールとこのガイドでは、接続を接続アカウントと呼びます。App Runner では、サードパーティーの接続情報を必要とするサービスを作成するときに、接続リソースが必要です。接続に関する重要な情報を次に示します。

- プロバイダー – App Runner は現在、GitHub または Bitbucket との接続リソースを必要とします。
- 共有 – 接続リソースを使用して、同じリポジトリプロバイダーアカウントを使用する複数の App Runner サービスを作成できます。
- リソース管理 – App Runner では、接続を作成および削除できます。ただし、既存の接続を変更することはできません。
- リソースクォータ – 接続リソースには、各 AWS アカウント の に関連付けられたクォータが設定されています AWS リージョン。このクォータに達した場合は、新しいプロバイダーアカウントに接続する前に接続を削除する必要がある場合があります。次のセクション「」で説明するように、App Runner コンソールまたは API を使用して接続を削除できますthe section called “接続の管理”。詳細については、「the section called “App Runner リソースクォータ”」を参照してください。

接続の管理

次のいずれかの方法を使用して App Runner 接続を管理します。

App Runner console

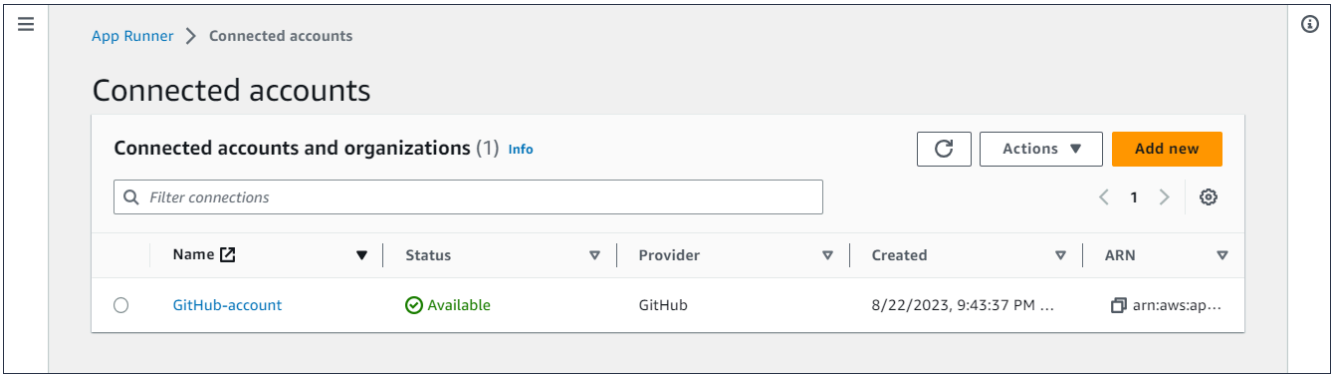
App Runner コンソールを使用してサービスを作成する場合は、接続の詳細を指定します。接続リソースを明示的に作成する必要はありません。コンソールでは、以前に接続した GitHub または Bitbucket アカウントに接続するか、新しいアカウントに接続するかを選択できます。必要に応じて、App Runner が接続リソースを作成します。新しい接続の場合、一部のプロバイダーでは、接続を使用する前に認証ハンドシェイクを完了する必要があります。コンソールでこのプロセスを実行します。

コンソールには、既存の接続を管理するページもあります。サービスの作成時に認証ハンドシェイクを実行しなかった場合は、接続の認証ハンドシェイクを完了できます。使用しなくなった接続を削除することもできます。次の手順は、リポジトリプロバイダー接続を管理する方法を示しています。

アカウントの接続を管理するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、接続されたアカウントを選択します。

   次に、コンソールにアカウントのリポジトリプロバイダー接続のリストが表示されます。



3. リスト上の任意の接続で、次のいずれかのアクションを実行できるようになりました。

   - GitHub/Bitbucket アカウントまたは組織を開く – 接続の名前を選択します。
   - 認証ハンドシェイクの完了 – 接続を選択し、アクションメニューからハンドシェイクの完了を選択します。コンソールでは、認証ハンドシェイクプロセスについて説明します。
   - 接続の削除 – 接続を選択し、アクションメニューから削除を選択します。削除プロンプトの指示に従います。

App Runner API or AWS CLI

次の App Runner API アクションを使用して接続を管理できます。

- CreateConnection – リポジトリプロバイダーアカウントへの接続を作成します。接続を作成したら、App Runner コンソールを使用して認証ハンドシェイクを手動で完了する必要があります。このプロセスについては、前のセクションで説明します。
- ListConnections – に関連付けられた App Runner 接続のリストを返します AWS アカウント。
- DeleteConnection – 接続を削除します。の接続クォータに達した場合は、不要な接続を削除する必要がある場合があります AWS アカウント。

App Runner の自動スケーリングの管理

AWS App Runner は、App Runner アプリケーションのコンピューティングリソース、特にインスタンスを自動的にスケールアップまたはスケールダウンします。自動スケーリングは、トラフィックが重い場合に適切なリクエスト処理を提供し、トラフィックが遅くなった場合にコストを削減します。

自動スケーリング設定

いくつかのパラメータを設定して、サービスの Auto Scaling 動作を調整できます。App Runner は、AutoScalingConfiguration と呼ばれる共有可能なリソースで自動スケーリング設定を維持します。サービスに割り当てる前に、スタンドアロンの自動スケーリング設定を作成して維持できます。サービスに関連付けられたら、設定を引き続き維持できます。新しいサービスの作成中または既存のサービスの設定中に、新しい自動スケーリング設定を作成することもできます。新しい Auto Scaling 設定が作成されたら、その設定をサービスに関連付けて、サービスの作成または設定プロセスを続行できます。

命名とリビジョン

自動スケーリング設定には、名前と数値リビジョンがあります。設定の複数のリビジョンには、同じ名前と異なるリビジョン番号があります。高可用性や低コストなど、さまざまな Auto Scaling シナリオに異なる設定名を使用できます。名前ごとに複数のリビジョンを追加して、特定のシナリオの設定を微調整できます。設定ごとに最大 10 個の一意の自動スケーリング設定名と最大 5 個のリビジョンを持つことができます。制限に達し、さらに作成する必要がある場合は、1 つを削除してから別のものを作成できます。App Runner では、デフォルトとして設定されているか、アクティブなサービスで使用されている設定を削除することはできません。クォータの詳細については、「the section called “App Runner リソースクォータ”」を参照してください。

デフォルト設定の設定

App Runner サービスを作成または更新するときに、Auto Scaling 設定リソースを指定できます。自動スケーリング設定の提供はオプションです。指定しない場合、App Runner は推奨値を含むデフォルトの自動スケーリング設定を提供します。自動スケーリング設定機能を使用すると、App Runner が提供するデフォルトを使用する代わりに、独自のデフォルトの自動スケーリング設定を設定できます。別の Auto Scaling 設定をデフォルトとして指定すると、その設定は今後作成する新しいサービスに自動的にデフォルトとして割り当てられます。新しいデフォルトの指定は、既存のサービスに以前に設定された関連付けには影響しません。

自動スケーリングによるサービスの設定

1 つの Auto Scaling 設定を複数の App Runner サービス間で共有して、サービスに同じ Auto Scaling 動作を持たせることができます。App Runner コンソールまたは App Runner API を使用して自動スケーリング設定を設定する方法の詳細については、このトピックで後述するセクションを参照してください。共有可能なリソースの詳細については、「」を参照してくださいthe section called “設定リソース”。

設定可能な設定

次の Auto Scaling 設定を構成できます。

- 最大同時実行数 – インスタンスが処理する同時リクエストの最大数。同時リクエストの数がこのクォータを超えると、App Runner はサービスをスケールアップします。
- 最大サイズ – サービスがスケールアップできるインスタンスの最大数。これは、サービスのトラフィックを同時に処理できるインスタンスの最大数です。
- 最小サイズ – App Runner がサービスにプロビジョニングできるインスタンスの最小数。サービスには、少なくともこの数のプロビジョニングされたインスタンスがあります。これらのインスタンスの一部は、トラフィックをアクティブに処理します。残りは、費用対効果の高いコンピューティングキャパシティーリザーブの一部であり、すぐにアクティブ化できます。プロビジョニングされたすべてのインスタンスのメモリ使用量に対して料金が発生します。アクティブなサブセットのみの CPU 使用率に対して料金が発生します。

> **Note**
>
> vCPU リソース数は、App Runner がサービスに提供できるインスタンスの数を決定します。これは、 AWS Fargate サービスに存在する Fargate オンデマンド vCPU リソース数の調整可能なクォータ値です。アカウントの vCPU クォータ設定を表示したり、クォータの引き上げをリクエストしたりするには、 の Service Quotas コンソールを使用します AWS マネジメントコンソール。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「サービスAWS Fargate クォータ」を参照してください。

サービスの自動スケーリングを管理する

次のいずれかの方法を使用して、App Runner サービスの自動スケーリングを管理します。

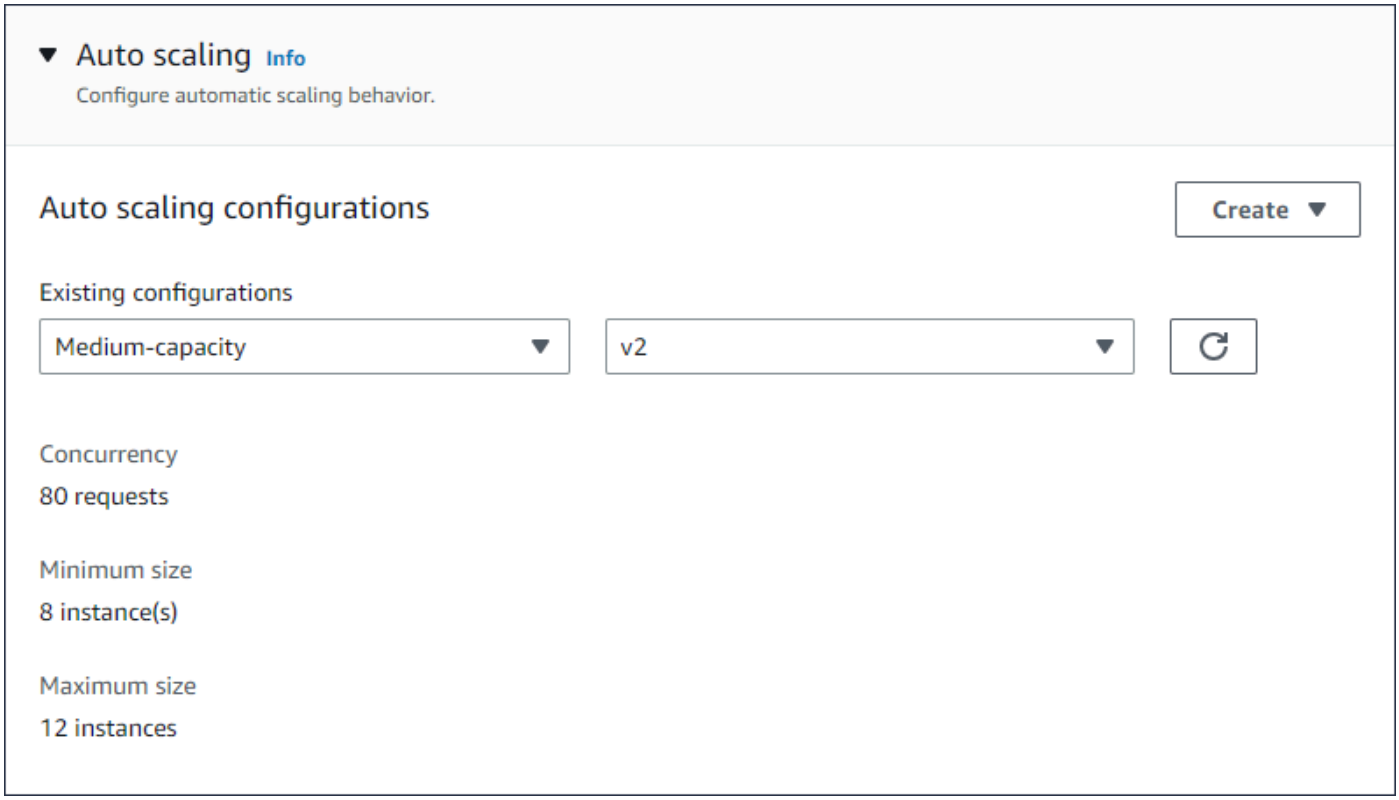
App Runner console

App Runner コンソールを使用してサービスを作成する場合、またはサービス設定を更新する場合は、自動スケーリング設定を指定できます。

> **Note**
>
> サービスに関連付けられている自動スケーリング設定またはリビジョンを変更すると、サービスが再デプロイされます。

Auto Scaling 設定ページには、サービスの Auto Scaling を設定するオプションがいくつか用意されています。

- 既存の設定とリビジョンを割り当てるには — 既存の設定ドロップダウンから値を選択します。最新のリビジョンバージョンは、隣接するドロップダウンでデフォルトになります。選択したい別のリビジョンが存在する場合は、リビジョンのドロップダウンから選択します。リビジョンバージョンの設定値が表示されます。
- 新しい Auto Scaling 設定を作成して割り当てるには – Create メニューから Create new ASC を選択します。これにより、カスタム自動スケーリング設定の追加ページが起動します。自動スケーリングパラメータの設定名と値を入力します。次に、[追加] をクリックします。App Runner は新しい Auto Scaling 設定リソースを作成し、新しい設定を選択して表示した状態で Auto Scaling セクションに戻ります。
- 新しいリビジョンを作成して割り当てるには — まず、既存の設定ドロップダウンから設定名を選択します。次に、作成メニューから ASC リビジョンの作成を選択します。これにより、カスタム自動スケーリング設定の追加ページが起動します。自動スケーリングパラメータの値を入力します。次に、[追加] をクリックします。App Runner は新しい Auto Scaling 設定リビジョンを作成し、新しいリビジョンを選択して表示した状態で Auto Scaling セクションに戻ります。



App Runner API or AWS CLI

CreateService または UpdateService App Runner API アクションを呼び出すときに、`AutoScalingConfigurationArn`パラメータを使用して、サービスの Auto Scaling 設定リソースを指定できます。

次のセクションでは、Auto Scaling 設定リソースを管理するためのガイダンスを提供します。

Auto Scaling 設定リソースの管理

次のいずれかの方法を使用して、アカウントの App Runner 自動スケーリング設定とリビジョンを管理します。

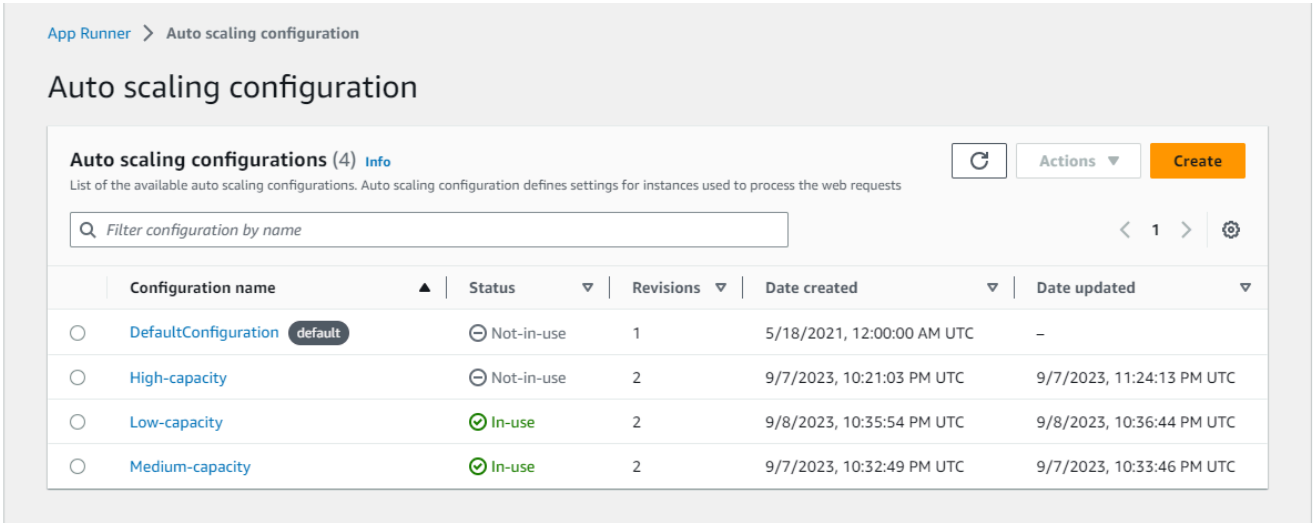
App Runner console

自動スケーリング設定を管理する

Auto Scaling 設定ページには、アカウントで設定した Auto Scaling 設定が一覧表示されます。このページで自動スケーリング設定を作成および管理し、後で 1 つ以上の App Runner サービスに割り当てることができます。

このページから次のいずれかを実行できます。

- 新しい Auto Scaling 設定を作成します。
- 既存の Auto Scaling 設定の新しいリビジョンを作成します。
- 自動スケーリング設定を削除します。
- 自動スケーリング設定をデフォルトとして設定します。



アカウントで自動スケーリング設定を管理するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、Auto Scaling 設定を選択します。コンソールには、アカウントの Auto Scaling 設定のリストが表示されます。

   これで、次のいずれかを実行できます。

   - 新しい Auto Scaling 設定を作成するには、次の手順に従います。

     a. Auto Scaling 設定ページで、作成を選択します。

        自動スケーリング設定の作成ページが表示されます。

     b. 設定名、同時実行数、最小サイズ、最大サイズの値を入力します。

     c. (オプション) タグを追加する場合は、Auto new tag を選択します。次に、表示されるフィールドに名前と値を入力します (オプション)。

     d. [作成] を選択します。

   - 既存の Auto Scaling 設定の新しいリビジョンを作成するには、次の手順に従います。

a. Auto Scaling 設定ページで、新しいリビジョンを必要とする設定の横にあるラジオボタンを選択します。次に、アクションメニューからリビジョンの作成を選択します。

   リビジョンの作成ページが表示されます。

b. で、同時実行、最小サイズ、最大サイズの値を入力します。

c. (オプション) タグを追加する場合は、Auto new tag を選択します。次に、表示されるフィールドに名前と値を入力します (オプション)。

d. [作成] を選択します。

- 自動スケーリング設定を削除するには、次の手順に従います。

a. Auto Scaling 設定ページで、削除する必要がある設定の横にあるラジオボタンを選択します。

b. アクションメニューから削除を選択します。

c. 削除を続行するには、確認ダイアログで削除を選択します。それ以外の場合は、キャンセルを選択します。

> **Note**
>
> App Runner は、削除の選択がデフォルトとして設定されていないか、アクティブなサービスで現在使用されていることを検証します。

- 自動スケーリング設定をデフォルトとして設定するには、次の手順に従います。

a. Auto Scaling 設定ページで、デフォルトとして設定する必要がある設定の横にあるラジオボタンを選択します。

b. アクションメニューからデフォルトとして設定を選択します。

c. App Runner が、作成したすべての新しいサービスのデフォルト設定として最新のリビジョンを使用することを示すダイアログが表示されます。確認を選択して続行します。それ以外の場合は、キャンセルを選択します。

> **Note**
>
> - 自動スケーリング設定をデフォルトに設定すると、今後作成する新しいサービスにデフォルト設定として自動的に割り当てられます。

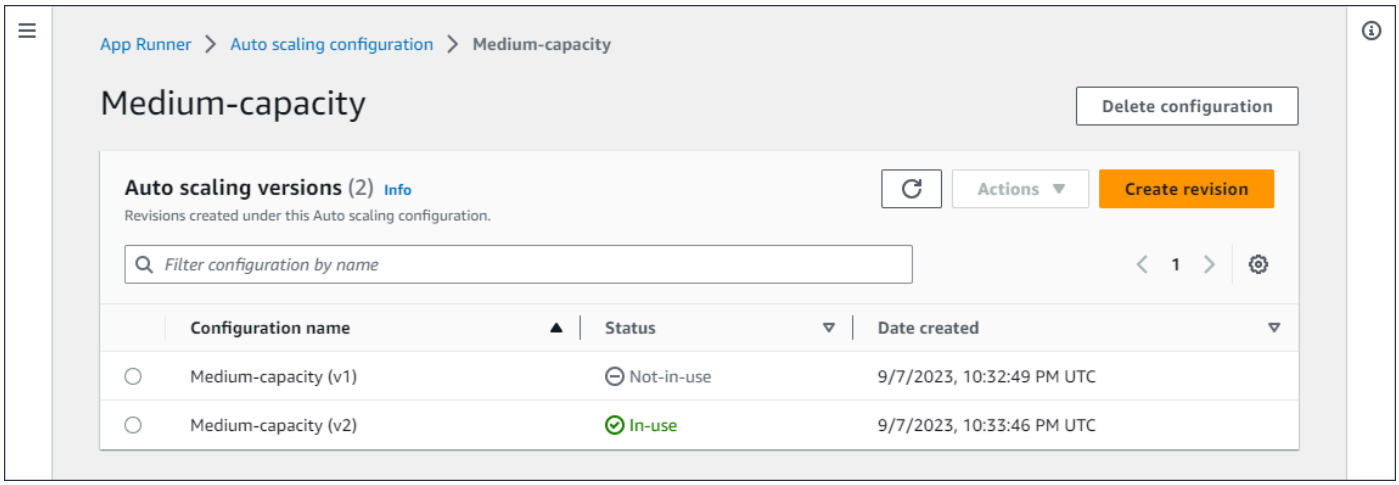
- 新しいデフォルトの指定は、既存のサービスに以前に設定された関連付けには影響しません。
- 指定されたデフォルトの自動スケーリング設定にリビジョンがある場合、App Runner は最新のリビジョンをデフォルトとして割り当てます。

リビジョンの管理

コンソールには、Auto Scaling リビジョンと呼ばれる既存の Auto Scaling リビジョンを作成および管理するためのページもあります。Auto Scaling 設定ページで設定の名前を選択して、このページにアクセスします。

Auto Scaling リビジョンページから次のいずれかを実行できます。

- 新しい Auto Scaling リビジョンを作成します。
- 自動スケーリング設定リビジョンをデフォルトとして設定します。
- リビジョンを削除します。
- 関連するすべてのリビジョンを含め、Auto Scaling 設定全体を削除します。
- リビジョンの設定の詳細を表示します。
- リビジョンに関連付けられたサービスのリストを表示します。
- リストされたサービスのリビジョンを変更します。



アカウントの Auto Scaling リビジョンを管理するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。

2. ナビゲーションペインで、Auto Scaling 設定を選択します。コンソールには、アカウントの Auto Scaling 設定のリストが表示されます。??? セクションの前の手順のセットには、このページの画面イメージが含まれています。
3. これで、特定の Auto Scaling 設定をドリルダウンして、そのすべてのリビジョンを表示および管理できるようになりました。Auto Scaling 設定ペインの Configuration name 列で、Auto Scaling 設定名を選択します。ラジオボタンではなく実際の名前を選択します。これにより、Auto Scaling リビジョンページのその設定のすべてのリビジョンのリストに移動します。
4. これで、次のいずれかを実行できます。
   - 既存の Auto Scaling 設定の新しいリビジョンを作成するには、次の手順に従います。
     a. Auto Scaling リビジョンページで、リビジョンの作成を選択します。

        リビジョンの作成ページが表示されます。
     b. 同時実行、最小サイズ、最大サイズの値を入力します。
     c. (オプション) タグを追加する場合は、Auto new tag を選択します。次に、表示されるフィールドに名前と値を入力します (オプション)。
     d. [作成] を選択します。
   - 関連するすべてのリビジョンを含む Auto Scaling 設定全体を削除するには、次の手順に従います。
     a. ページの右上にある設定の削除を選択します。
     b. 削除を続行するには、確認ダイアログで削除を選択します。それ以外の場合は、キャンセルを選択します。

        > **Note**
        >
        > App Runner は、削除の選択がデフォルトとして設定されていないか、アクティブなサービスで現在使用されていることを検証します。
   - 自動スケーリングリビジョンをデフォルトとして設定するには、次の手順に従います。
     a. デフォルトとして設定する必要があるリビジョンの横にあるラジオボタンを選択します。
     b. アクションメニューからデフォルトとして設定を選択します。

> **Note**
>
> - 自動スケーリング設定をデフォルトに設定すると、今後作成する新しいサービスにデフォルト設定として自動的に割り当てられます。
> - 新しいデフォルトの指定は、既存のサービスに以前に設定された関連付けには影響しません。

- リビジョンの設定の詳細を表示するには、次の手順に従います。
  - リビジョンの横にあるラジオボタンを選択します。

    ARN を含むリビジョンの設定の詳細は、下部の分割パネルに表示されます。この手順の最後にある画面イメージを参照してください。
- リビジョンに関連付けられているサービスのリストを表示するには、次の手順に従います。
  - リビジョンの横にあるラジオボタンを選択します。

    サービスパネルは、下部の分割パネルのリビジョン設定の詳細の下に表示されます。パネルには、この Auto Scaling 設定リビジョンを使用するすべてのサービスが一覧表示されます。この手順の最後にある画面イメージを参照してください。
- リストされたサービスのリビジョンを変更するには、次の手順に従います。

  a. リビジョンの横にあるラジオボタンがまだない場合は、それを選択します。

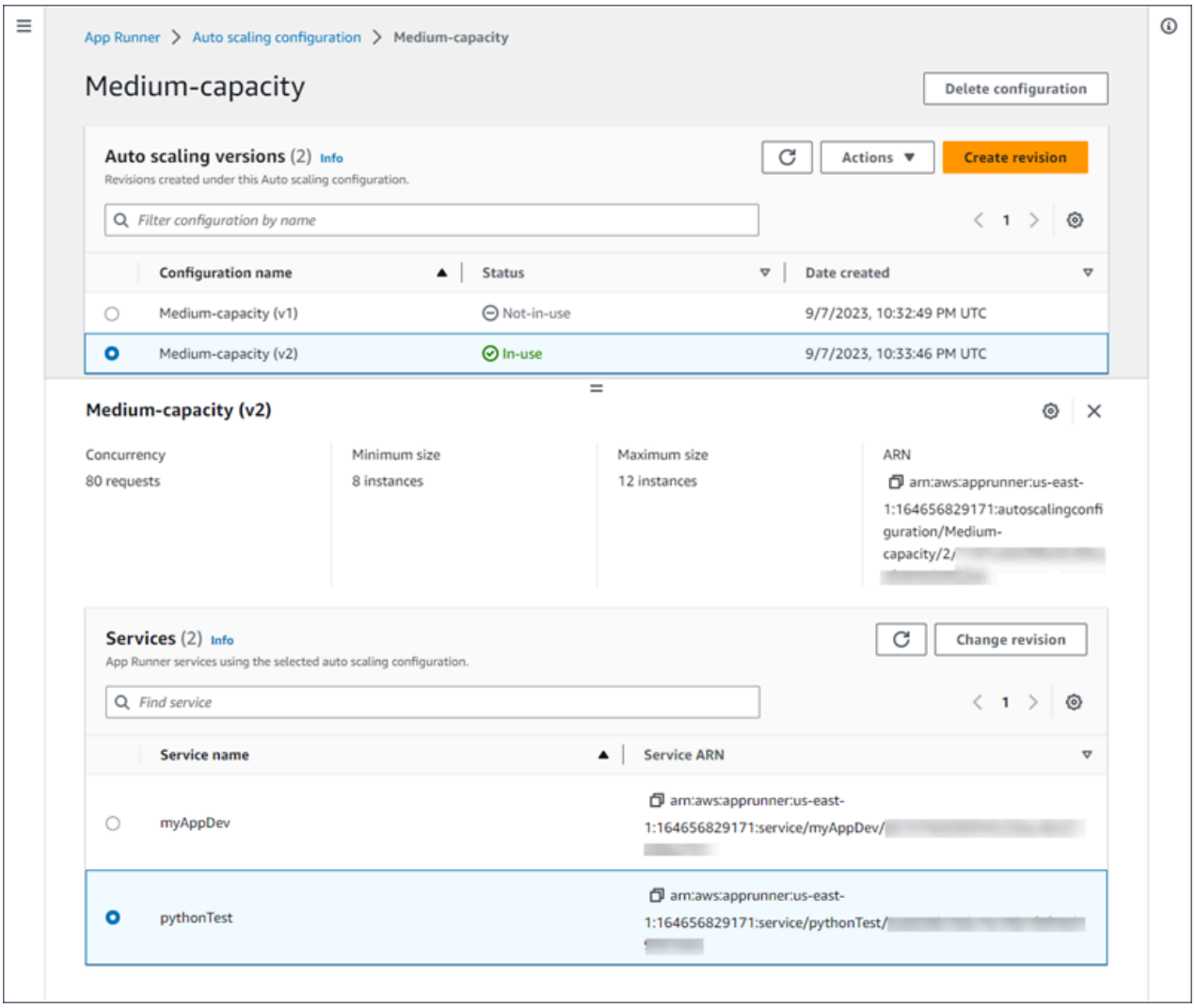
     サービスパネルは、下部の分割パネルのリビジョン設定の詳細の下に表示されます。パネルには、この Auto Scaling 設定リビジョンを使用するすべてのサービスが一覧表示されます。この手順の最後にある画面イメージを参照してください。

  b. サービスパネルで、変更するサービスの横にあるラジオボタンを選択します。次に、リビジョンの変更を選択します。

  c. ASC リビジョンの変更パネルが表示されます。ドロップダウンで使用可能なリビジョンから選択します。前に選択した Auto Scaling 設定のリビジョンのみを使用できます。別の Auto Scaling 設定に変更する必要がある場合は、前のセクションの手順に従ってくださいthe section called “サービスの自動スケーリングを管理する”。

     更新を選択して変更を続行します。それ以外の場合は、キャンセルを選択します。

> **Note**
>
> サービスに関連付けられているリビジョンを変更すると、サービスが再デプロイされます。
> 更新された関連付けを表示するには、このパネルで更新を選択する必要があります。
> 進行中のアクティビティとサービス再デプロイのステータスを確認するには、パネルのパンくずリストを使用して App Runner > サービスに移動し、サービスを選択してから、サービスの概要パネルからログタブを表示します。



App Runner API or AWS CLI

次の App Runner API アクションを使用して、自動スケーリング設定リソースを管理します。

- CreateAutoScalingConfiguration – 新しい自動スケーリング設定または既存の設定へのリビジョンを作成します。
- UpdateDefaultAutoScalingConfiguration – 自動スケーリング設定をデフォルトに設定します。既存のデフォルトの自動スケーリング設定は、デフォルト以外の設定に自動的に設定されます。
- ListAutoScalingConfigurations – に関連付けられている自動スケーリング設定のリスト AWS アカウントと概要情報を返します。
- ListServicesForAutoScalingConfiguration – 自動スケーリング設定を使用して、関連する App Runner サービスのリストを返します。
- DescribeAutoScalingConfiguration – 自動スケーリング設定の完全な説明を返します。
- DeleteAutoScalingConfiguration – 自動スケーリング設定を削除します。最上位の自動スケーリング設定、1 つの特定のリビジョン、または最上位設定に関連付けられたすべてのリビジョンを削除できます。オプションの `DeleteAllRevisions`パラメータを使用して、すべてのリビジョンを削除します。の Auto Scaling 設定リソースクォータに達した場合は AWS アカウント、不要な Auto Scaling 設定を削除する必要がある場合があります。

App Runner サービスのカスタムドメイン名の管理

AWS App Runner サービスを作成すると、App Runner はそのサービスにドメイン名を割り当てます。これは、App Runner が所有する`awsapprunner.com`ドメインのサブドメインです。ドメイン名を使用して、 サービスで実行されているウェブアプリケーションにアクセスできます。

> **Note**
>
> App Runner アプリケーションのセキュリティを強化するために、*.awsapprunner.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化するために、App Runner アプリケーションのデフォルトのドメイン名で機密 Cookie を設定する必要がある場合は、`__Host-`プレフィックス付きの Cookie を使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

ドメイン名を所有している場合は、それを App Runner サービスに関連付けることができます。App Runner が新しいドメインを検証したら、そのドメインを使用して App Runner ドメインに加えてアプリケーションにアクセスできます。最大 5 つのカスタムドメインを関連付けることができます。

> **Note**
>
> オプションで、ドメインのwwwサブドメインを含めることができます。ただし、これは現在 API でのみサポートされています。App Runner コンソールでは、ドメインのwwwサブドメインを含めることはできません。

> **Note**
>
> AWS App Runner は Route 53 プライベートホストゾーンの使用をサポートしていません。プライベートホストゾーンは、Amazon VPC トラフィックのドメイン名解決をカスタマイズします。プライベートホストゾーンの詳細については、Route 53 ドキュメントの「プライベートホストゾーンの使用」を参照してください。

カスタムドメインをサービスに関連付ける (リンクする)

カスタムドメインをサービスに関連付けるときは、CNAME レコードと DNS ターゲットレコードを DNS サーバーに追加する必要があります。以下のセクションでは、CNAME レコードと DNS ターゲットレコード、およびそれらの使用方法について説明します。

> **Note**
>
> DNS プロバイダーとして Amazon Route 53 を使用している場合、App Runner は、App Runner ウェブアプリケーションにリンクするために必要な証明書検証と DNS レコードを使用してカスタムドメインを自動的に設定します。これは、App Runner コンソールを使用してカスタムドメインをサービスにリンクするときに発生します。詳細については、次のカスタムドメインを管理するトピックを参照してください。

CNAME レコード

カスタムドメインをサービスに関連付けると、App Runner は証明書検証用の証明書検証レコードのセットを提供します。これらの証明書検証レコードをドメインネームシステム (DNS) サーバーに

追加する必要があります。App Runner が提供する証明書検証レコードを DNS サーバーに追加します。これにより、App Runner はドメインを所有または管理していることを検証できます。

> **Note**
>
> カスタムドメイン証明書を自動更新するには、DNS サーバーから証明書検証レコードを削除しないようにしてください。証明書の更新に関連する問題を解決する方法については、「」を参照してくださいthe section called “カスタムドメイン証明書の更新”。

App Runner は ACM を使用してドメインを検証します。DNS レコードで CAA レコードを使用している場合は、少なくとも 1 つの CAA レコードが を参照していることを確認してくださいamazon.com。それ以外の場合、ACM はドメインを検証してドメインを正常に作成できません。

CAA に関連するエラーが発生した場合の解決方法については、次のリンクを参照してください。

- 認証機関認可 (CAA) の問題
- ACM 証明書の発行または更新に関する CAA エラーを解決するにはどうすればよいですか?
- カスタムドメイン名

> **Note**
>
> DNS プロバイダーとして Amazon Route 53 を使用している場合、App Runner は、App Runner ウェブアプリケーションにリンクするために必要な証明書検証と DNS レコードを使用してカスタムドメインを自動的に設定します。これは、App Runner コンソールを使用してカスタムドメインをサービスにリンクするときに発生します。詳細については、次のカスタムドメインを管理するトピックを参照してください。

DNS ターゲットレコード

DNS ターゲットレコードを DNS サーバーに追加して、App Runner ドメインをターゲットにします。このオプションを選択した場合は、カスタムドメインに 1 つのレコードを追加し、wwwサブドメインに別のレコードを追加します。次に、App Runner コンソールでカスタムドメインのステータスがアクティブになるまで待ちます。これには通常数分かかりますが、最大 24～48 時間 (1～2 日) かかる場合があります。カスタムドメインが検証されると、App Runner はこのドメインからウェブアプリケーションへのトラフィックのルーティングを開始します。

> **Note**
>
> App Runner サービスとの互換性を高めるには、DNS プロバイダーとして Amazon Route 53 を使用することをお勧めします。Amazon Route 53 を使用してパブリック DNS レコードを管理しない場合は、DNS プロバイダーに連絡してレコードを追加する方法を確認してください。
> DNS プロバイダーとして Amazon Route 53 を使用している場合は、サブドメインに CNAME レコードまたはエイリアスレコードを追加できます。ルートドメインの場合は、エイリアスレコードを使用していることを確認してください。

Amazon Route 53 または別のプロバイダーからドメイン名を購入できます。Amazon Route 53 でドメイン名を購入するには、「Amazon Route 53 デベロッパーガイド」の「新しいドメインの登録」を参照してください。

Route 53 で DNS ターゲットを設定する方法については、Amazon Route 53 デベロッパーガイドの「リソースへのトラフィックのルーティング」を参照してください。

GoDaddy、Shopify、Hover などの他のレジストラで DNS ターゲットを設定する方法については、DNS ターゲットレコードの追加に関する特定のドキュメントを参照してください。

App Runner サービスに関連付けるドメインを指定する

App Runner サービスに関連付けるドメインは、次の方法で指定できます。

- ルートドメイン – DNS には固有の制限があるため、ルートドメイン名の CNAME レコードの作成がブロックされる可能性があります。たとえば、ドメイン名が の場合`example.com`、 のトラフィックを App Runner サービス`acme.example.com`にルーティングする CNAME レコードを作成できます。ただし、 のトラフィックを App Runner サービス`example.com`にルーティングする CNAME レコードを作成することはできません。ルートドメインを作成するには、エイリアスレコードを追加してください。

  エイリアスレコードは Route 53 に固有であり、CNAME レコードよりも以下の利点があります。

  - Route 53 では、ルートドメインまたはサブドメインのエイリアスレコードを作成できるため、柔軟性が向上します。たとえば、ドメイン名が の場合`example.com`、 `example.com`または のリクエストを App Runner サービス`acme.example.com`にルーティングするレコードを作成できます。
  - コスト効率が高くなります。これは、Route 53 がエイリアスレコードを使用してトラフィックをルーティングするリクエストに対して課金しないためです。

- サブドメイン – 例えば、 `login.example.com`または ですadmin.login.example.com。オプションで、同じオペレーションの一部としてwwwサブドメインを関連付けることもできます。サブドメインに CNAME レコードまたはエイリアスレコードを追加できます。
- ワイルドカード – 例: `*.example.com`。この場合、 wwwオプションを使用することはできません。ワイルドカードは、ルートドメインの直接サブドメインとしてのみ指定でき、それ自体でのみ指定できます。これらは有効な仕様ではありません: `login*.example.com`、`*.login.example.com`。このワイルドカード仕様は、すべての即時サブドメインを関連付け、ルートドメイン自体を関連付けません。ルートドメインは別のオペレーションで関連付ける必要があります。

より具体的なドメインの関連付けは、より具体的なドメインの関連付けよりも優先されます。たとえば、 は を`login.example.com`上書きします`*.example.com`。より具体的な関連付けの証明書と CNAME が使用されます。

次の例は、複数のカスタムドメインの関連付けを使用する方法を示しています。

1. サービスのホームページ`example.com`に関連付けます。を有効にwwwして を関連付けます`www.example.com`。
2. サービスのログインページ`login.example.com`に関連付けます。
3. カスタム「見つからない」ページ`*.example.com`に関連付けます。

カスタムドメインの関連付けを解除 (リンク解除)

App Runner サービスからカスタムドメインの関連付けを解除 (リンク解除) できます。ドメインのリンクを解除すると、App Runner はこのドメインからウェブアプリケーションへのトラフィックのルーティングを停止します。

> **Note**
>
> DNS サーバーから関連付けを解除したドメインのレコードを削除する必要があります。

App Runner は、ドメインの有効性を追跡する証明書を内部で作成します。これらの証明書は AWS Certificate Manager (ACM) に保存されます。App Runner は、ドメインがサービスから関連付け解除されてから、またはサービスが削除されてから 7 日間、これらの証明書を削除しません。

カスタムドメインを管理する

次のいずれかの方法を使用して、App Runner サービスのカスタムドメインを管理します。

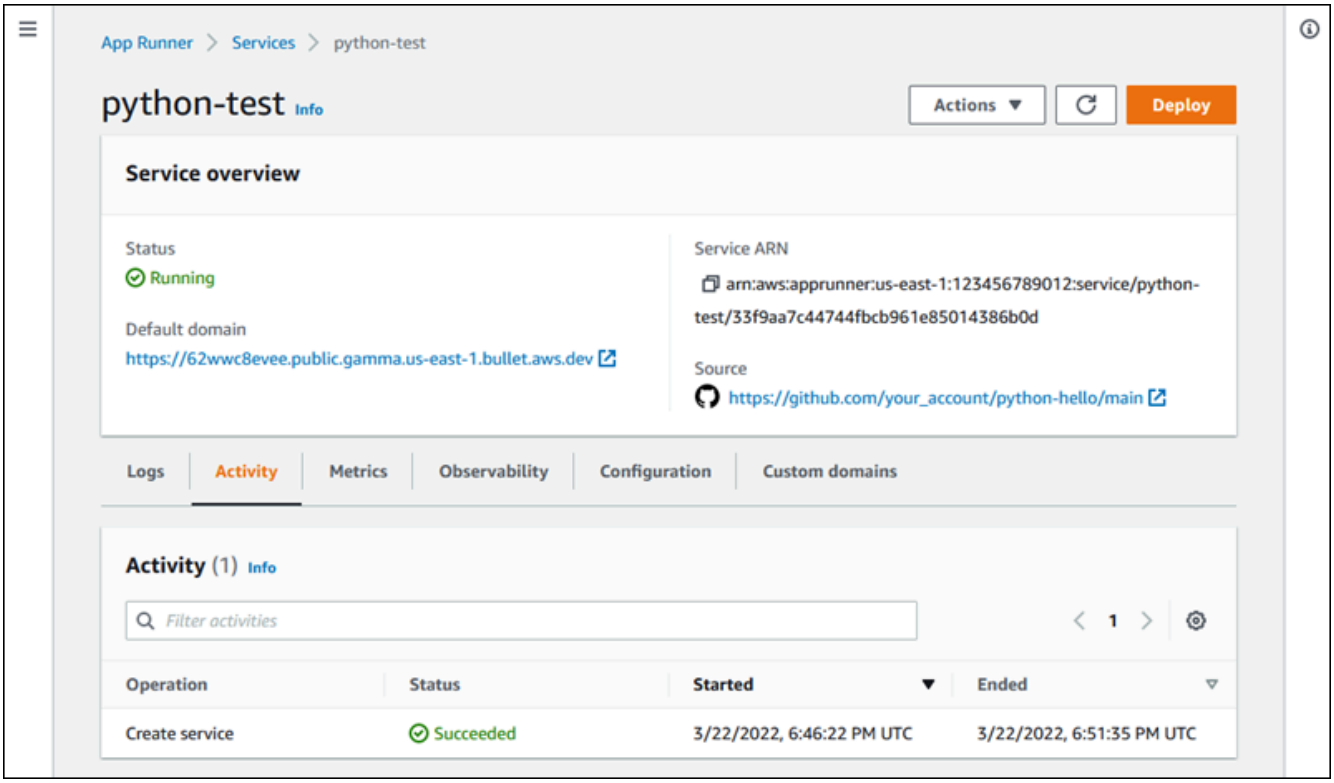
> **Note**
>
> App Runner サービスとの互換性を高めるには、DNS プロバイダーとして Amazon Route 53 を使用することをお勧めします。Amazon Route 53 を使用してパブリック DNS レコードを管理しない場合は、DNS プロバイダーに連絡してレコードを追加する方法を確認してください。
> DNS プロバイダーとして Amazon Route 53 を使用している場合は、サブドメインに CNAME レコードまたはエイリアスレコードを追加できます。ルートドメインの場合は、エイリアスレコードを使用していることを確認してください。

App Runner console

App Runner コンソールを使用してカスタムドメインを関連付け (リンク) するには

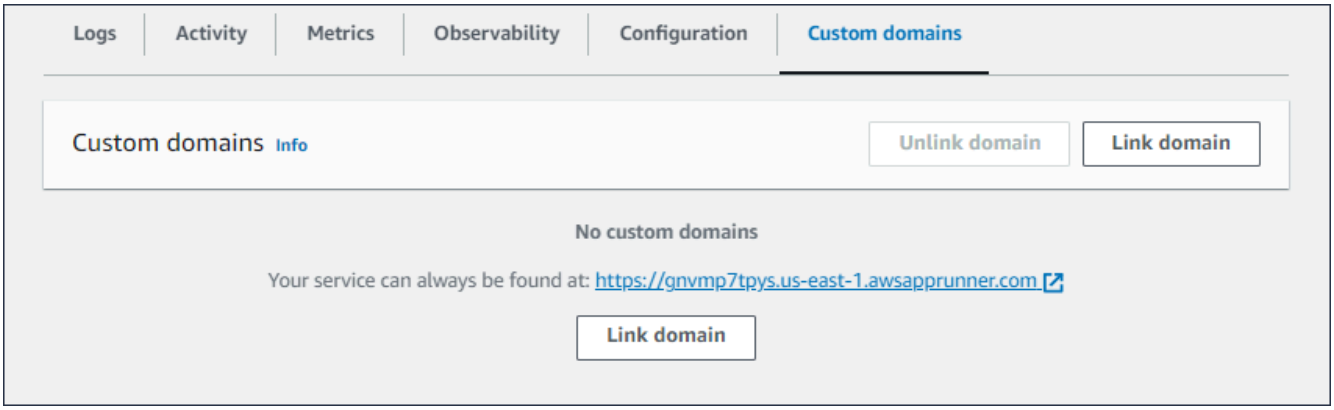
1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

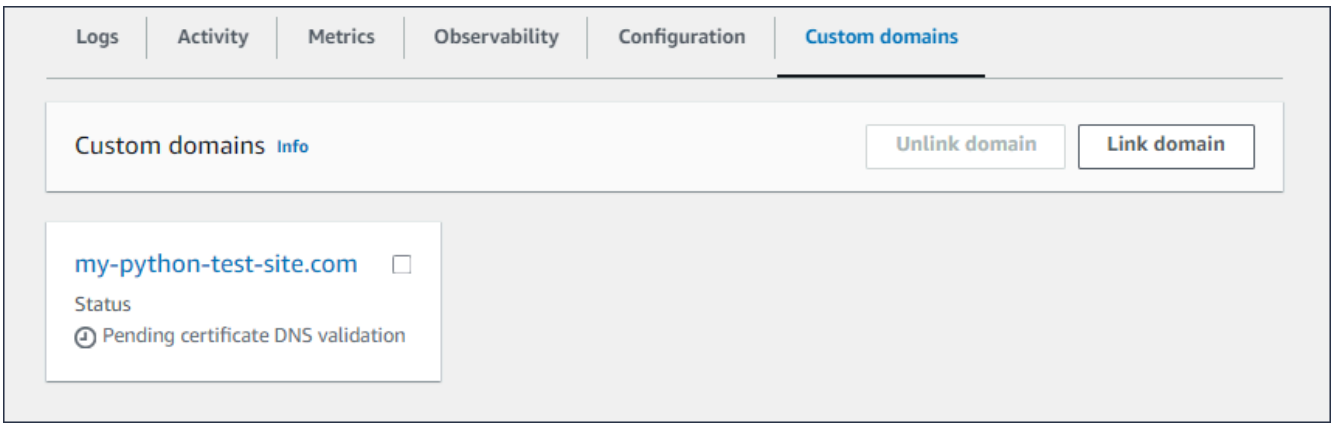
   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. サービスダッシュボードページで、カスタムドメインタブを選択します。

コンソールには、サービスに関連付けられているカスタムドメイン、またはカスタムドメインなしが表示されます。

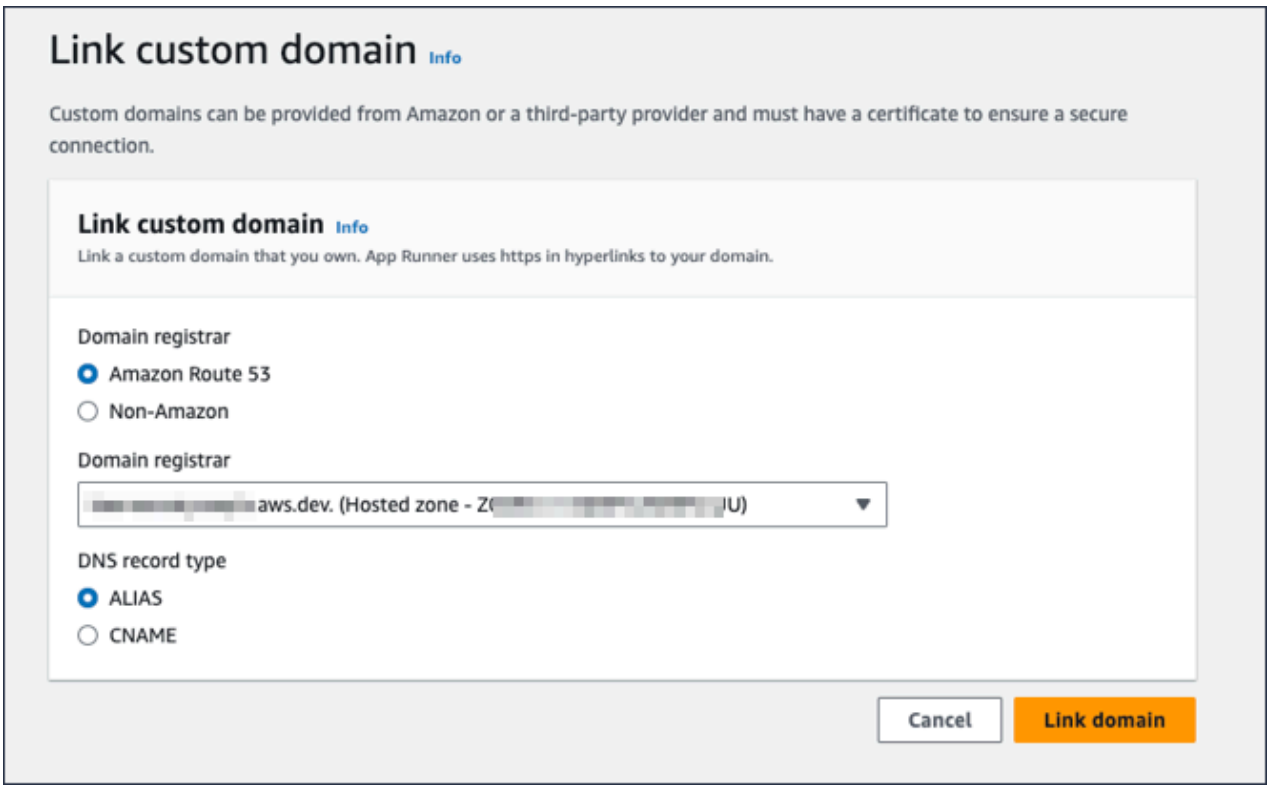




4. カスタムドメインタブで、ドメインのリンクを選択します。
5. カスタムドメインのリンクページが表示されます。
   - カスタムドメインが Amazon Route 53 に登録されている場合は、ドメインレジストラに Amazon Route 53 を選択します。
     a. ドロップダウンリストからドメイン名を選択します。このリストには、Route 53 ドメイン名とホストゾーン ID が表示されます。

     > **Note**
     >
     > まず、他の App Runner リソースの管理に使用するのと同じ AWS アカウントから Amazon Route 53 サービスを使用して Route 53 ドメインを作成する必要があります。

     b. DNS レコードタイプを選択します。
     c. Link domain を選択します。

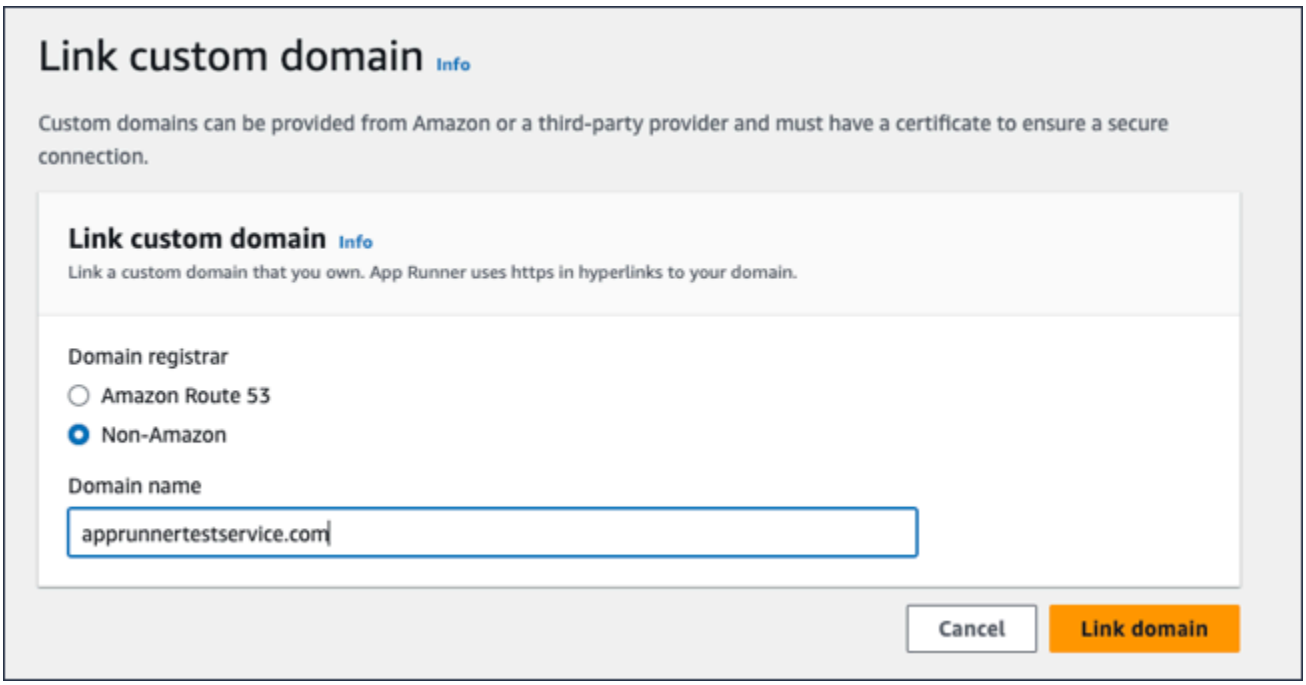


> **Note**
>
> App Runner で自動設定の試行が失敗したことを示すエラーメッセージが表示された場合は、DNS レコードを手動で設定して続行できます。この問題は、同じドメイン名が以前にサービスからリンク解除されていて、後でサービスを指す DNS プロバイダーレコードが削除されていない場合に発生する可能性があります。この場合、App Runner はこれらのレコードを自動的に上書きすることをブロックされます。DNS 設定を完了するには、この手順の残りのステップをスキップし、「」の手順に従いますAmazon Route 53 エイリアスレコードを設定する。

- カスタムドメインが別のドメインレジストラに登録されている場合は、ドメインレジストラに Amazon 以外の を選択します。

  a. ドメイン名を入力します。

  b. Link domain を選択します。



6. DNS の設定ページが表示されます。

   - Amazon Route 53 が DNS プロバイダーの場合、このステップはオプションです。

     この時点で、App Runner は、必要な証明書の検証と DNS レコードを使用して Route 53 ドメインを自動的に設定しました。

     > **Note**
     >
     > この同じドメイン名が以前にサービスからリンク解除され、後でサービスを指す DNS プロバイダーレコードが削除されなかった場合、App Runner が試行した自動設定が失敗した可能性があります。この問題を回避して DNS 関連付けを完了するには、DNS の設定ページのステップ (1) と (2) に進み、現在のターゲットレコードと証明書レコードを DNS プロバイダーにコピーします。

   - 証明書検証レコードと DNS ターゲットレコードをコピーし、DNS サーバーに追加します。App Runner は、ドメインを所有または管理していることを検証できます。

> **Note**
>
> カスタムドメイン証明書を自動更新するには、DNS サーバーから証明書検証レコードを削除しないでください。

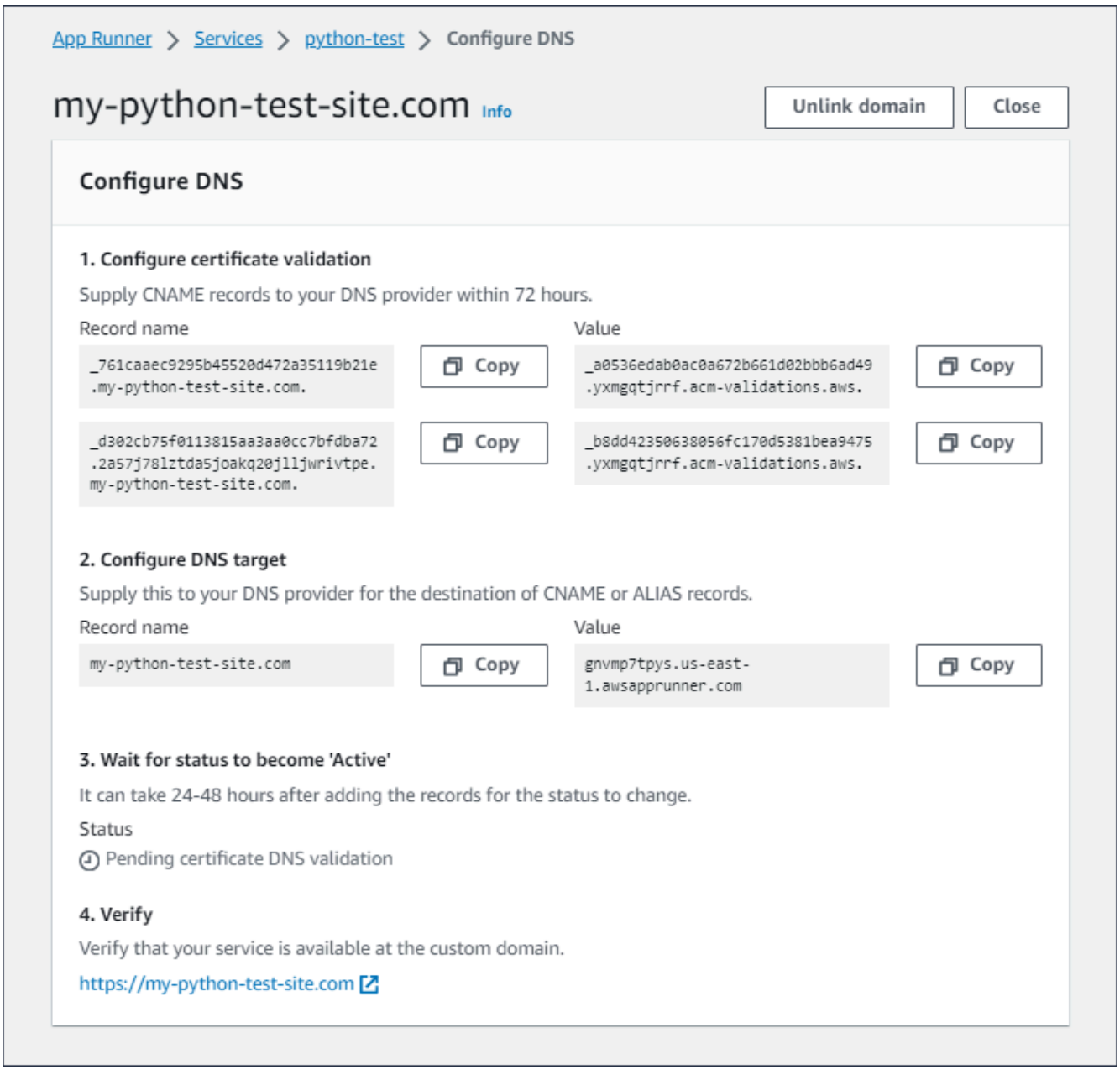
- 証明書検証の設定の詳細については、「AWS Certificate Manager ユーザーガイド」の「DNS 検証」を参照してください。
- Amazon Route 53 エイリアスレコードで DNS ターゲットを設定する方法については、「」を参照してくださいthe section called “Amazon Route 53 エイリアスレコードを設定する”。

- Amazon Route 53 以外の DNS プロバイダーを使用している場合は、以下の手順に従います。

  - 証明書検証レコードと DNS ターゲットレコードをコピーし、DNS サーバーに追加します。App Runner は、ドメインを所有または管理していることを検証できます。

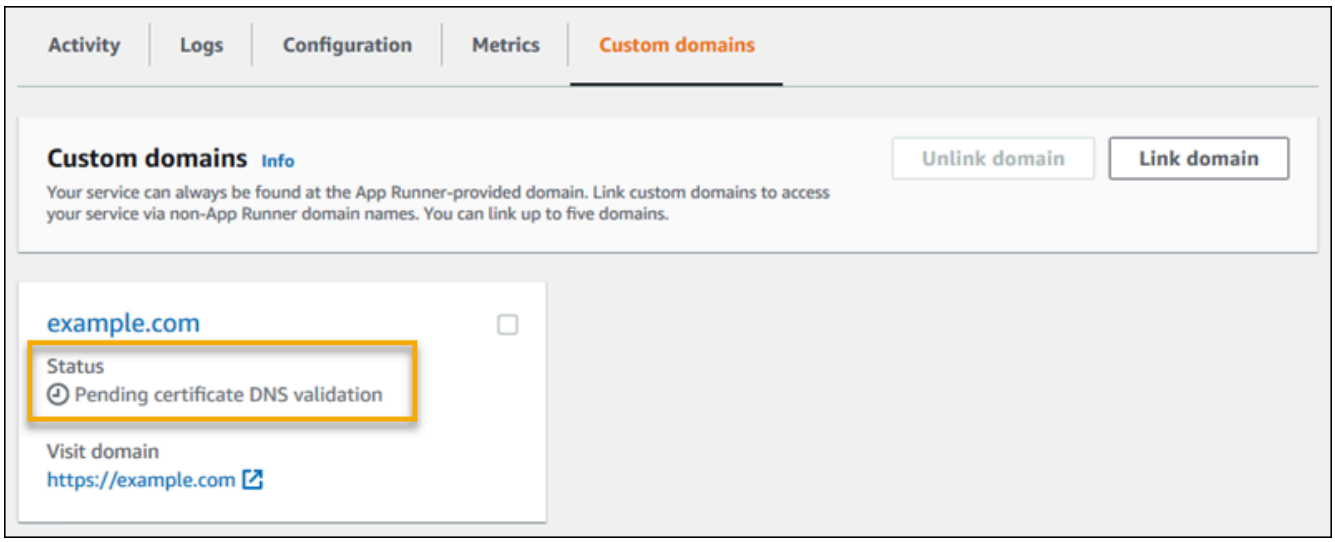
    > **Note**
    >
    > カスタムドメイン証明書を自動更新するには、DNS サーバーから証明書検証レコードを削除しないでください。

    - 証明書検証の設定の詳細については、「AWS Certificate Manager ユーザーガイド」の「DNS 検証」を参照してください。
    - GoDaddy、Shopify、Hover などの他のレジストラで DNS ターゲットを設定する方法については、DNS ターゲットの追加に関する特定のドキュメントを参照してください。

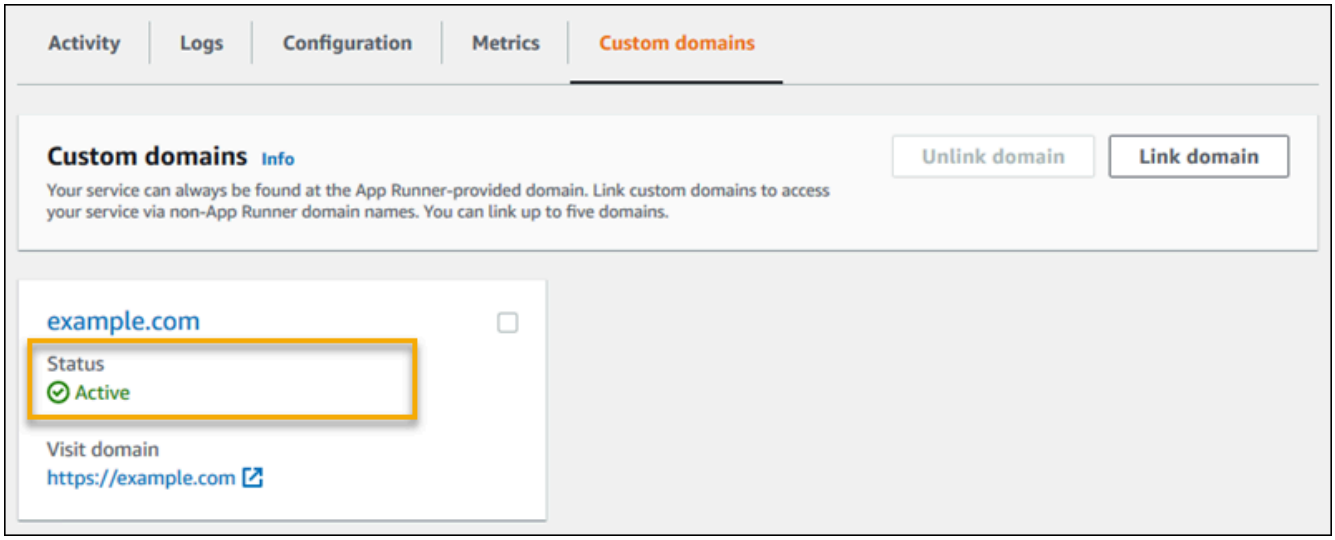


7. 閉じるを選択する

コンソールにダッシュボードが再度表示されます。カスタムドメインタブには、保留中の証明書 DNS 検証ステータスでリンクしたドメインを示す新しいタイルがあります。



8. ドメインのステータスがアクティブに変わったら、ドメインを参照してトラフィックをルーティングできることを確認します。



> **Note**
>
> カスタムドメインに関連するエラーをトラブルシューティングする方法については、「」を参照してくださいthe section called “カスタムドメイン名”。

App Runner コンソールを使用してカスタムドメインの関連付けを解除 (リンク解除) するには

1. カスタムドメインタブで、関連付けを解除するドメインのタイルを選択し、ドメインのリンク解除を選択します。

2. Unlink domain ダイアログで、Unlink domain を選択してアクションを確認します。

> **Note**
>
> DNS サーバーから関連付けを解除したドメインのレコードを削除する必要があります。

App Runner API or AWS CLI

App Runner API または を使用してカスタムドメインをサービスに関連付けるには AWS CLI、AssociateCustomDomain API アクションを呼び出します。呼び出しが成功すると、サービスに関連付けられているカスタムドメインを記述する CustomDomain オブジェクトが返されます。オブジェクトには`CREATING`ステータスが表示され、CertificateValidationRecord オブジェクトのリストが含まれます。この呼び出しは、DNS ターゲットの設定に使用できるターゲットエイリアスも返します。これらは、DNS に追加できるレコードです。

App Runner API または を使用してサービスからカスタムドメインの関連付けを解除するには AWS CLI、DisassociateCustomDomain API アクションを呼び出します。呼び出しが成功すると、サービスとの関連付けを解除するカスタムドメインを記述する CustomDomain オブジェクトが返されます。オブジェクトには`DELETING`ステータスが表示されます。

トピック

- ターゲット DNS の Amazon Route 53 エイリアスレコードを設定する

ターゲット DNS の Amazon Route 53 エイリアスレコードを設定する

> **Note**
>
> Amazon Route 53 が DNS プロバイダーである場合は、この手順に従う必要はありません。この場合、App Runner は、App Runner ウェブアプリケーションにリンクするために必要な証明書の検証と DNS レコードを使用して Route 53 ドメインを自動的に設定します。
> App Runner の自動設定の試行が失敗した場合は、次の手順に従って DNS 設定を完了します。同じドメイン名が以前にサービスからリンク解除され、後でサービスを指す DNS プロバイダーレコードが削除されなかった場合、App Runner はこれらのレコードを自動的に上

書きすることをブロックされます。この手順では、Route 53 DNS に手動でコピーする方法について説明します。

Amazon Route 53 を DNS プロバイダーとして使用して、トラフィックを App Runner サービスにルーティングできます。可用性が高くスケーラブルなドメインネームシステム (DNS) ウェブサービスです。Amazon Route 53 レコードには、トラフィックが App Runner サービスにルーティングされる方法を制御する設定が含まれています。CNAME レコードまたは ALIAS レコードのいずれかを作成します。CNAME レコードとエイリアスレコードの比較については、「Amazon Route 53 デベロッパーガイド」の「エイリアスレコードと非エイリアスレコードの選択」を参照してください。

> **Note**
>
> Amazon Route 53 は現在、2022 年 8 月 1 日以降に作成されたサービスのエイリアスレコードをサポートしています。

Amazon Route 53 console

Amazon Route 53 エイリアスレコードを設定するには

1. にサインイン AWS マネジメントコンソール し、Route 53 コンソールを開きます。
2. ナビゲーションペインで [Hosted zones] を選択します。
3. App Runner サービスへのトラフィックのルーティングに使用するホストゾーンの名前を選択します。
4. [Create record (レコードを作成)] を選択します。
5. 次の値を指定します。

   - ルーティングポリシー: 該当するルーティングポリシーを選択します。詳細については、「ルーティングポリシーの選択」を参照してください。
   - レコード名: App Runner サービスへのトラフィックのルーティングに使用するドメイン名を入力します。デフォルト値はホストゾーンの名前です。たとえば、ホストゾーンの名前が `example.com`で、 `acme.example.com`を使用してトラフィックを環境にルーティングする場合は、 と入力しますacme。
   - 値/ルートトラフィックの送信先: App Runner アプリケーションへのエイリアスを選択し、エンドポイントの送信元のリージョンを選択します。トラフィックをルーティングするアプリケーションのドメイン名を選択します。

- レコードタイプ: デフォルトの A – IPv4 アドレスを受け入れます。
- ターゲットの状態を評価する: デフォルト値を受け入れます。はい。

6. [レコードを作成] を選択します。

作成した Route 53 エイリアスレコードは、60 秒以内にすべての Route 53 サーバーに伝播されます。Route 53 サーバーにエイリアスレコードが伝播されると、作成したエイリアスレコードの名前を使用してトラフィックを App Runner サービスにルーティングできます。

DNS の変更が反映されるまでに時間がかかりすぎる場合のトラブルシューティング方法については、「DNS の変更が Route 53 とパブリックリゾルバーに反映されるまでに時間がかかるのはなぜですか?」を参照してください。

Amazon Route 53 API or AWS CLI

Amazon Route 53 API を使用して Amazon Route 53 エイリアスレコードを設定するには、または ChangeResourceRecordSets API アクションを AWS CLI 呼び出します。Route 53 のターゲットホストゾーン ID については、「サービスエンドポイント」を参照してください。

App Runner サービスの一時停止と再開

ウェブアプリケーションを一時的に無効にしてコードの実行を停止する必要がある場合は、 AWS App Runner サービスを一時停止できます。App Runner は、サービスのコンピューティング容量をゼロに削減します。

アプリケーションを再度実行する準備ができたら、App Runner サービスを再開できます。App Runner は、新しいコンピューティングキャパシティをプロビジョニングし、アプリケーションをデプロイして、アプリケーションを実行します。アプリケーションソースは再デプロイされず、ビルドは必要ありません。代わりに、App Runner は現在デプロイされているバージョンで再開されます。アプリケーションは App Runner ドメインを保持します。

> **Important**
>
> - サービスを一時停止すると、アプリケーションの状態は失われます。たとえば、コードが使用したエフェメラルストレージは失われます。コードの場合、サービスの一時停止と再開は、新しいサービスへのデプロイと同等です。
> - コードの欠陥 (検出されたバグやセキュリティの問題など) が原因でサービスを一時停止した場合、サービスを再開する前に新しいバージョンをデプロイすることはできません。

したがって、サービスを実行し続け、代わりに最後の安定したアプリケーションバージョンにロールバックすることをお勧めします。

- サービスを再開すると、App Runner はサービスを一時停止する前に使用された最後のアプリケーションバージョンをデプロイします。サービスの一時停止以降に新しいソースバージョンを追加した場合、自動デプロイが選択されていても、App Runner はそれらのバージョンを自動的にデプロイしません。たとえば、イメージリポジトリに新しいイメージバージョンがあるか、コードリポジトリに新しいコミットがあるとします。これらのバージョンは、自動的にデプロイされません。

  新しいバージョンをデプロイするには、App Runner サービスを再開した後に、手動デプロイを実行するか、別のバージョンをソースリポジトリに追加します。

比較した一時停止と削除

App Runner サービスを一時停止して一時的に無効にします。コンピューティングリソースのみが終了し、保存されたデータ (アプリケーションバージョンのコンテナイメージなど) はそのまま残ります。サービスの再開は迅速で、アプリケーションを新しいコンピューティングリソースにデプロイする準備が整います。App Runner ドメインは同じままです。

App Runner サービスを削除して、完全に削除します。保存されたデータは削除されます。サービスを再作成する必要がある場合、App Runner はソースを再度取得し、コードリポジトリの場合はビルドする必要があります。ウェブアプリケーションは、新しい App Runner ドメインを取得します。

サービスが一時停止したとき

サービスを一時停止し、一時停止ステータスになると、API コールやコンソールオペレーションなど、アクションリクエストに異なる方法で応答します。サービスを一時停止しても、ランタイムに影響する方法でサービスの定義や設定を変更しない App Runner アクションを実行できます。つまり、実行中のサービスの動作、スケール、またはその他の特性がアクションによって変更された場合、一時停止したサービスに対してそのアクションを実行することはできません。

次のリストは、一時停止したサービスで実行できる API アクションと実行できない API アクションに関する情報を示しています。同等のコンソールオペレーションも同様に許可または拒否されます。

一時停止したサービスで実行できるアクション

- *List** および *Describe**アクション – 情報のみを読み取るアクション。

- *DeleteService* – サービスはいつでも削除できます。
- *TagResource*、 *UntagResource* – タグはサービスに関連付けられていますが、その定義の一部ではなく、ランタイムの動作には影響しません。

一時停止したサービスで実行できないアクション

- *StartDeployment* アクション (または コンソールを使用した手動デプロイ)
- *UpdateService* (または、変更のタグ付けを除くコンソールを使用した設定変更)
- *CreateCustomDomainAssociations*, *DeleteCustomDomainAssociations*
- *CreateConnection*, *DeleteConnection*

サービスの一時停止と再開

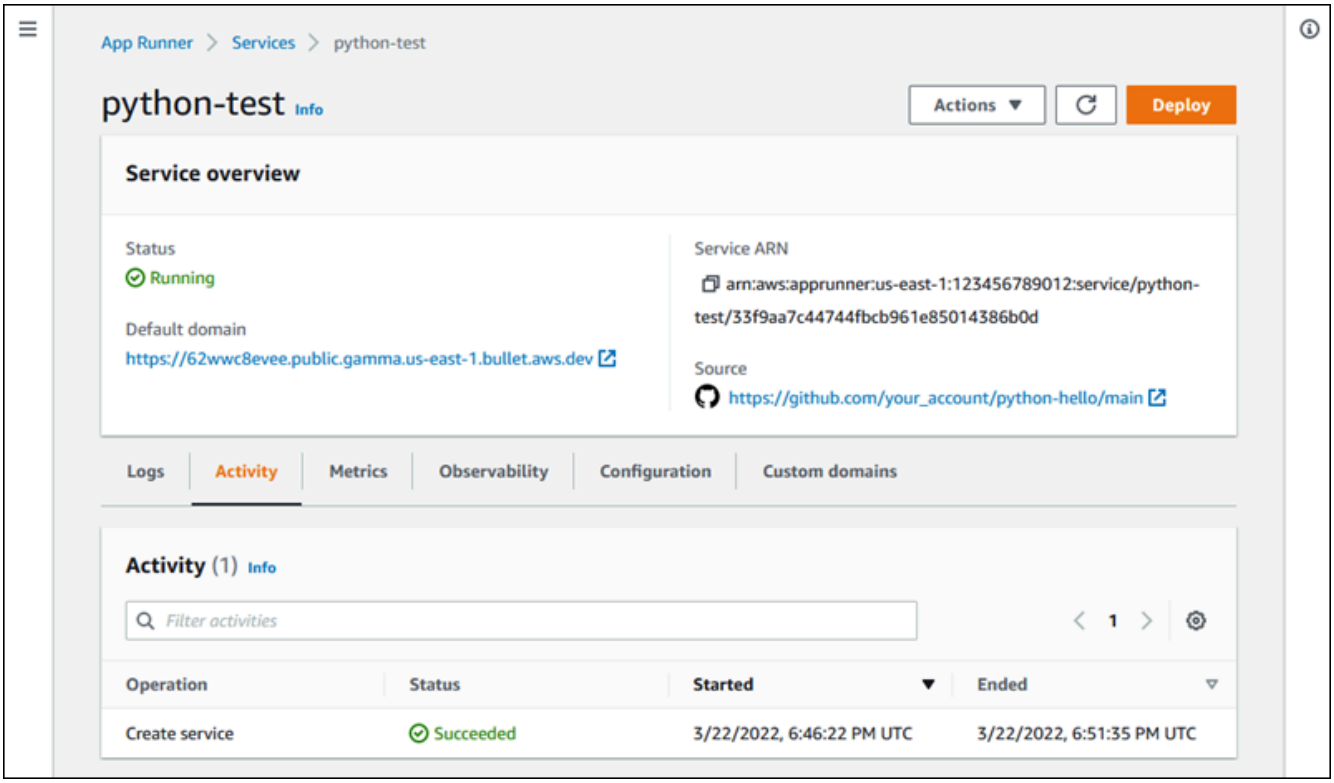
次のいずれかの方法を使用して、App Runner サービスを一時停止して再開します。

App Runner console

App Runner コンソールを使用してサービスを一時停止するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. アクションを選択し、一時停止を選択します。

   サービスダッシュボードページで、サービスのステータスが進行中のオペレーションに変わり、一時停止に変わります。これで、サービスは一時停止されます。

App Runner コンソールを使用してサービスを再開するには

1. アクションを選択し、再開を選択します。

   サービスダッシュボードページで、サービスのステータスが進行中のオペレーションに変わります。

2. サービスが再開されるまで待ちます。サービスダッシュボードページで、サービスのステータスが実行中に戻ります。

3. サービスの再開が成功したことを確認するには、サービスダッシュボードページで App Runner ドメイン値を選択します。これは、サービスのウェブサイトの URL です。ウェブアプリケーションが正しく実行されていることを確認します。

App Runner API or AWS CLI

App Runner API または を使用してサービスを一時停止するには AWS CLI、PauseService API アクションを呼び出します。呼び出しが を示す Service オブジェクトで成功したレスポンスを返した場合`"Status": "OPERATION_IN_PROGRESS"`、App Runner はサービスの一時停止を開始します。

App Runner API または を使用してサービスを再開するには AWS CLI、ResumeService API アクションを呼び出します。呼び出しが を示す Service オブジェクトで成功したレスポンスを返した場合`"Status": "OPERATION_IN_PROGRESS"`、App Runner はサービスの再開を開始します。

App Runner サービスの削除

AWS App Runner サービスで実行されているウェブアプリケーションを終了する場合は、サービスを削除できます。サービスを削除すると、実行中のウェブサービスが停止し、基盤となるリソースが削除され、関連付けられたデータが削除されます。

次のいずれかの理由で App Runner サービスを削除できます。

- ウェブアプリケーションが不要になりました。たとえば、廃止されたり、使用が終了した開発バージョンです。
- App Runner サービスクォータに達しました – 同じ に新しいサービスを作成し AWS リージョン 、アカウントに関連付けられたクォータに達しました。詳細については、「the section called “App Runner リソースクォータ”」を参照してください。
- セキュリティまたはプライバシーに関する考慮事項 – App Runner がサービスに保存するデータを削除する必要があります。

比較した一時停止と削除

App Runner サービスを一時停止して一時的に無効にします。コンピューティングリソースのみが終了し、保存されたデータ (アプリケーションバージョンのコンテナイメージなど) はそのまま残ります。サービスの再開は迅速で、アプリケーションを新しいコンピューティングリソースにデプロイする準備が整います。App Runner ドメインは同じままです。

App Runner サービスを削除して、完全に削除します。保存されたデータは削除されます。サービスを再作成する必要がある場合、App Runner はソースを再度フェッチし、コードリポジトリの場合はビルドする必要があります。ウェブアプリケーションは、新しい App Runner ドメインを取得します。

App Runner は何を削除しますか?

サービスを削除すると、App Runner は関連する項目の一部を削除し、他の項目は削除しません。次のリストに詳細を示します。

App Runner が削除する項目:

- コンテナイメージ – デプロイしたイメージまたは App Runner がソースコードから構築したイメージのコピー。App Runner AWS アカウント が所有する内部 を使用して Amazon Elastic Container Registry (Amazon ECR) に保存されます。
- サービス設定 – App Runner サービスに関連付けられている設定。App Runner AWS アカウント が所有する内部 を使用して Amazon DynamoDB に保存されます。

App Runner が削除しない項目:

- 接続 – サービスに関連付けられている接続がある可能性があります。App Runner 接続は、複数の App Runner サービス間で共有できる別のリソースです。接続が不要になった場合は、明示的に削除できます。詳細については、「the section called “Connections”」を参照してください。
- カスタムドメイン証明書 – カスタムドメインを App Runner サービスにリンクすると、App Runner はドメインの有効性を追跡する証明書を内部で作成します。これらは AWS Certificate Manager (ACM) に保存されます。App Runner は、ドメインがサービスからリンク解除されてから、またはサービスが削除されてから 7 日間、証明書を削除しません。詳細については、「the section called “カスタムドメイン名”」を参照してください。

サービスを削除する

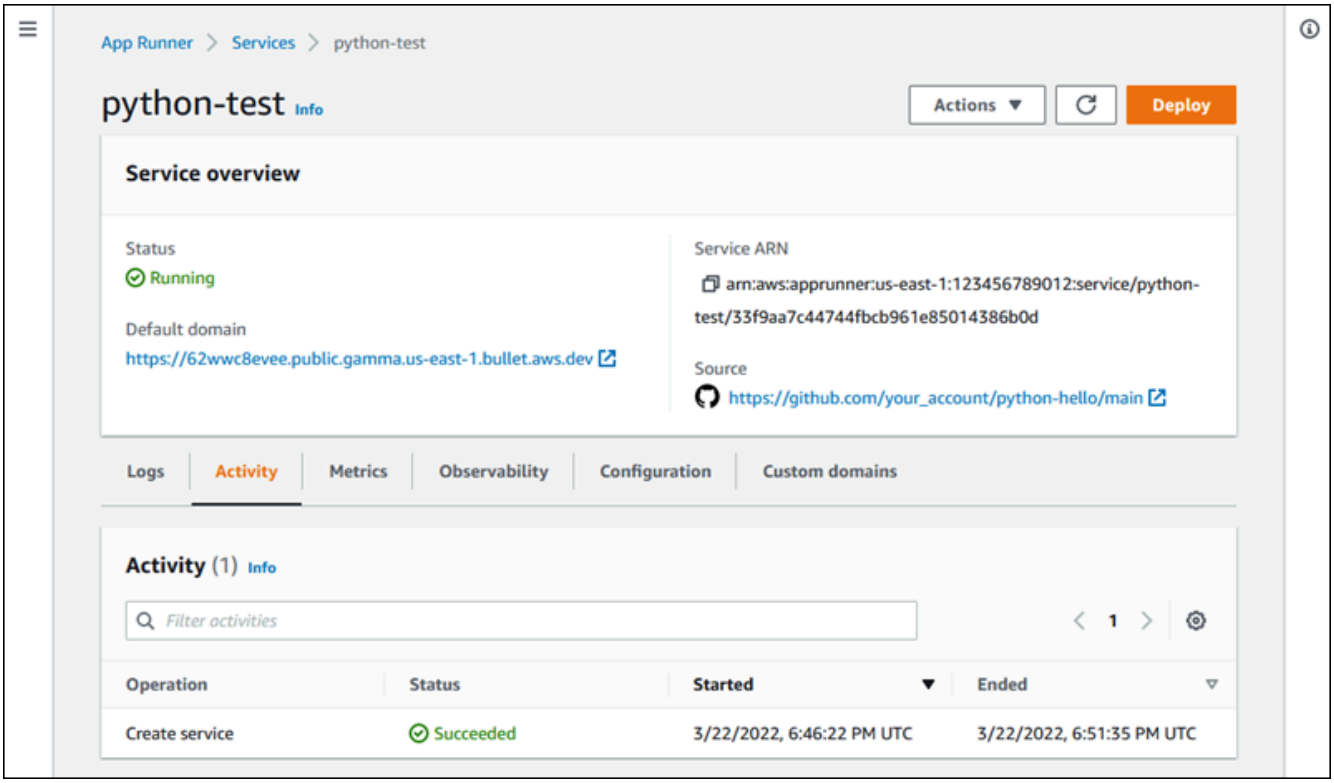
次のいずれかの方法を使用して App Runner サービスを削除します。

App Runner console

App Runner コンソールを使用してサービスを削除するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. [アクション]、[削除] の順に選択します。

   コンソールで サービスページに移動します。削除されるサービスに [進行中] とステータスが表示された後、サービスがリストから削除されます。これで、サービスが削除されます。

App Runner API or AWS CLI

App Runner API または を使用してサービスを削除するには AWS CLI、DeleteService API アクションを呼び出します。呼び出しが を示す Service オブジェクトで成功したレスポンスを返した場合`"Status": "OPERATION_IN_PROGRESS"`、App Runner はサービスの削除を開始します。

環境変数の参照

App Runner を使用すると、サービスの作成時またはサービスの更新時に、シークレットと設定をサービス内の環境変数として参照できます。

タイムアウトや再試行回数などの機密性のない設定データは、キーと値のペアとしてプレーンテキストで参照できます。プレーンテキストで参照する設定データは暗号化されず、App Runner サービス設定とアプリケーションログで他のユーザーに表示されます。

> **Note**
>
> セキュリティ上の理由から、App Runner サービスのプレーンテキストで機密データを参照しないでください。

機密データを環境変数として参照する

App Runner は、機密データをサービス内の環境変数として安全に参照することをサポートしています。参照する機密データを AWS Secrets Managerまたは AWS Systems Manager Parameter Store に保存することを検討してください。その後、App Runner コンソールから、または API を呼び出して、サービス内の環境変数として安全に参照できます。これにより、シークレットとパラメータの管理がアプリケーションコードとサービス設定から効果的に分離され、App Runner で実行されているアプリケーションの全体的なセキュリティが向上します。

> **Note**
>
> App Runner では、Secrets Manager と SSM パラメータストアを環境変数として参照しても料金は発生しません。ただし、Secrets Manager と SSM パラメータストアの使用には標準料金がかかります。
> 料金の詳細については、以下を参照してください。
>
> - AWS Secrets Manager の料金
> - AWS SSM パラメータストアの料金

以下は、機密データを環境変数として参照するプロセスです。

1. API キー、データベース認証情報、データベース接続パラメータ、アプリケーションバージョンなどの機密データをシークレットまたはパラメータとして AWS Secrets Manager または AWS Systems Manager パラメータストアに保存します。
2. App Runner が Secrets Manager と SSM パラメータストアに保存されているシークレットとパラメータにアクセスできるように、インスタンスロールの IAM ポリシーを更新します。詳細については、アクセス許可を参照してください。
3. 名前を割り当てて Amazon リソースネーム (ARN) を指定することで、シークレットとパラメータを環境変数として安全に参照します。サービスを作成するとき、またはサービスの設定を更新するときに、環境変数を追加できます。次のいずれかのオプションを使用して、環境変数を追加できます。
   - App Runner コンソール
   - App Runner API
   - `apprunner.yaml` 設定ファイル

   > **Note**
   >
   > App Runner サービスを作成または更新するときに、 を環境変数の名前PORTとして割り当てることはできません。これは App Runner サービスの予約済み環境変数です。

   シークレットとパラメータを参照する方法の詳細については、「環境変数の管理」を参照してください。

> **Note**
>
> App Runner はシークレットとパラメータ ARNs への参照のみを保存するため、機密データは App Runner サービス設定とアプリケーションログの他のユーザーには表示されません。

考慮事項

- AWS Secrets Manager または Parameter Store の AWS Systems Manager シークレットとパラメータにアクセスするための適切なアクセス許可でインスタンスロールを更新してください。詳細については、アクセス許可を参照してください。

- AWS Systems Manager Parameter Store が、起動または更新するサービス AWS アカウント と同じ にあることを確認します。現在、アカウント間で SSM パラメータストアパラメータを参照することはできません。
- シークレットとパラメータ値がローテーションまたは変更されると、App Runner サービスでは自動的に更新されません。App Runner はデプロイ中にシークレットとパラメータのみをプルするため、App Runner サービスを再デプロイします。
- また、App Runner サービスの SDK を使用して AWS Secrets Manager と AWS Systems Manager Parameter Store を直接呼び出すこともできます。
- エラーを回避するには、それらを環境変数として参照するときに、以下を確認してください。
  - シークレットの適切な ARN を指定します。
  - パラメータの正しい名前または ARN を指定します。

アクセス許可

AWS Secrets Manager または SSM パラメータストアに保存されているシークレットとパラメータの参照を有効にするには、インスタンスロールの IAM ポリシーに適切なアクセス許可を追加して、Secrets Manager と SSM パラメータストアにアクセスします。

> **Note**
>
> App Runner は、 アクセス許可がないと アカウントのリソースにアクセスできません。アクセス許可を指定するには、IAM ポリシーを更新します。

次のポリシーテンプレートを使用して、IAM コンソールでインスタンスロールを更新できます。これらのポリシーテンプレートは、特定の要件を満たすように変更できます。インスタンスロールの更新の詳細については、IAM ユーザーガイドの「ロールの変更」を参照してください。

> **Note**
>
> 環境変数を作成するときに、App Runner コンソールから次のテンプレートをコピーすることもできます。

次のテンプレートをインスタンスロールにコピーして、 からシークレットを参照するアクセス許可を追加しますAWS Secrets Manager。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue",
            "kms:Decrypt*"
        ],
        "Resource": [
            "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret",
            "arn:aws:kms:us-east-1:111122223333:key/my-key"
         ]
     }
   ]
}
```

次のテンプレートをインスタンスロールにコピーして、Parameter Store AWS Systems Manager からパラメータを参照するアクセス許可を追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
            "arn:aws:ssm:us-east-1:111122223333:parameter/my-parameter"
            ]
        }
    ]
}
```

環境変数の管理

次のいずれかの方法を使用して、App Runner サービスの環境変数を管理します。

- the section called “App Runner コンソール”
- the section called “App Runner API または AWS CLI”

App Runner コンソール

App Runner コンソールでサービスを作成またはサービスを更新するときに、環境変数を追加できます。

環境変数の追加

環境変数を追加するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. サービスを作成または更新するかどうかに基づいて、次のいずれかの手順を実行します。
   - 新しいサービスを作成する場合は、App Runner サービスの作成を選択し、サービスの設定に移動します。
   - 既存のサービスを更新する場合は、更新するサービスを選択し、サービスの設定タブに移動します。
3. サービス設定の「環境変数 - オプション」に移動します。
4. 要件に基づいて、次のいずれかのオプションを選択します。
   - 環境変数のソースからプレーンテキストを選択し、それぞれ環境変数名と環境変数値の下にキーと値のペアを入力します。

     > **Note**
     >
     > 非機密データを参照する場合は、プレーンテキストを選択します。このデータは暗号化されず、App Runner サービス設定とアプリケーションログで他のユーザーに表示されます。

   - 環境変数ソースから Secrets Manager を選択して、サービス内の環境変数 AWS Secrets Manager として に保存されているシークレットを参照します。参照するシークレットの環境変数名と Amazon リソースネーム (ARN) をそれぞれ環境変数名と環境変数値で指定します。

- 環境変数ソースから SSM パラメータストアを選択して、SSM パラメータストアに保存されているパラメータをサービスの環境変数として参照します。参照するパラメータの環境変数名と ARN をそれぞれ環境変数名と環境変数値で指定します。

> **Note**
>
> - App Runner サービスを作成または更新するときに、 を環境変数の名前PORTとして割り当てることはできません。これは App Runner サービスの予約済み環境変数です。
> - SSM パラメータストアパラメータが起動するサービス AWS リージョン と同じ にある場合は、完全な Amazon リソースネーム (ARN) またはパラメータの名前を指定できます。パラメータが別のリージョンにある場合は、完全な ARN を指定する必要があります。
> - 参照するパラメータが、起動または更新するサービスと同じアカウントにあることを確認します。現在、アカウント間で SSM パラメータストアパラメータを参照することはできません。

5. 環境変数を追加 を選択して、別の環境変数を参照します。
6. IAM ポリシーテンプレートを展開して、 と SSM パラメータストアに用意されている IAM ポリシーテンプレートを表示 AWS Secrets Manager およびコピーします。必要なアクセス許可でインスタンスロールの IAM ポリシーをまだ更新していない場合にのみ、これを行う必要があります。詳細については、アクセス許可を参照してください。

環境変数の削除

環境変数を削除する前に、アプリケーションコードが同じ内容を反映するように更新されていることを確認してください。アプリケーションコードが更新されない場合、App Runner サービスが失敗する可能性があります。

環境変数を削除するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. 更新するサービスの設定タブに移動します。
3. サービス設定の「環境変数 - オプション」に移動します。
4. 削除する環境変数の横にある「削除」を選択します。削除を確認するメッセージが表示されます。

5. [削除] を選択します。

App Runner API または AWS CLI

Secrets Manager と SSM パラメータストアに保存されている機密データは、サービス内の環境変数として追加することで参照できます。

> **Note**
>
> App Runner が Secrets Manager と SSM パラメータストアに保存されているシークレットとパラメータにアクセスできるように、インスタンスロールの IAM ポリシーを更新します。詳細については、アクセス許可を参照してください。

シークレットと設定を環境変数として参照するには

1. Secrets Manager または SSM パラメータストアでシークレットまたは設定を作成します。

   次の例は、SSM パラメータストアを使用してシークレットとパラメータを作成する方法を示しています。

   Exampleシークレットの作成 - リクエスト

   次の例は、データベース認証情報を表すシークレットを作成する方法を示しています。

   ```
   aws secretsmanager create-secret \
   –name DevRdsCredentials \
   –description "Rds credentials for development account." \
   –secret-string "{\"user\":\"diegor\",\"password\":\"EXAMPLE-PASSWORD\"}"
   ```

   Exampleシークレットの作成 - レスポンス

   ```
   arn:aws:secretsmanager:<region>:<aws_account_id>:secret:DevRdsCredentials
   ```

   Example設定の作成 - リクエスト

   次の例は、RDS 接続文字列を表すパラメータを作成する方法を示しています。

   ```
   aws systemsmanager put-parameter \
   ```

```
–name DevRdsConnectionString \
–value "mysql2://dev-mysqlcluster-rds.com:3306/diegor" \
–type "String" \
–description "Rds connection string for development account."
```

Example設定の作成 - レスポンス

```
arn:aws:ssm:<region>:<aws_account_id>:parameter/DevRdsConnectionString
```

2. Secrets Manager と SSM パラメータストアに保存されているシークレットと設定を参照するには、環境変数として追加します。App Runner サービスを作成または更新するときに、環境変数を追加できます。

   次の例は、コードベースおよびイメージベースの App Runner サービスでシークレットと設定を環境変数として参照する方法を示しています。

   Exampleイメージベースの App Runner サービス用の Input.json ファイル

```
{
  "ServiceName": "example-secrets",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageIdentifier": "<image-identifier>",
      "ImageConfiguration": {
        "Port": "<port>",
        "RuntimeEnvironmentSecrets": {

 "Credential1":"arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
          "Credential2":"arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
        }
      },
      "ImageRepositoryType": "ECR_PUBLIC"
    }
  },
  "InstanceConfiguration": {
    "Cpu": "1 vCPU",
    "Memory": "3 GB",
    "InstanceRoleArn": "<instance-role-arn>"
  }
}
```

Exampleイメージベースの App Runner サービス – リクエスト

```
aws apprunner create-service \
--cli-input-json file://input.json
```

Exampleイメージベースの App Runner サービス – レスポンス

```
{
...
      "ImageRepository": {
         "ImageIdentifier":"<image-identifier>",
         "ImageConfiguration":{
            "Port": "<port>",
            "RuntimeEnvironmentSecrets":{
               "Credential1":
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
               "Credential2": "arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
            },
            "ImageRepositoryType":"ECR"
      }
   },
   "InstanceConfiguration": {
        "CPU": "1 vCPU",
        "Memory": "3 GB",
        "InstanceRoleArn: "<instance-role-arn>"
   }
...
}
```

Exampleコードベースの App Runner サービス用の Input.json ファイル

```
{
  "ServiceName": "example-secrets",
  "SourceConfiguration": {
    "AuthenticationConfiguration": {
      "ConnectionArn": "arn:aws:apprunner:us-east-1:123456789012:connection/my-
github-connection/XXXXXXXXXX"
    },
    "AutoDeploymentsEnabled": false,
    "CodeRepository": {
```

```
      "RepositoryUrl": "<repository-url>",
      "SourceCodeVersion": {
        "Type": "BRANCH",
        "Value": "main"
      },
      "CodeConfiguration": {
        "ConfigurationSource": "API",
        "CodeConfigurationValues": {
          "Runtime": "<runtime>",
          "BuildCommand": "<build-command>",
          "StartCommand": "<start-command>",
          "Port": "<port>",
          "RuntimeEnvironmentSecrets": {

 "Credential1":"arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
            "Credential2":"arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
          }
        }
      }
    }
  },
  "InstanceConfiguration": {
    "Cpu": "1 vCPU",
    "Memory": "3 GB",
    "InstanceRoleArn": "<instance-role-arn>"
  }
}
```

Exampleコードベースの App Runner サービス – リクエスト

```
aws apprunner create-service \
--cli-input-json file://input.json
```

Exampleコードベースの App Runner サービス – レスポンス

```
{
...
   "SourceConfiguration":{
      "CodeRepository":{
         "RepositoryUrl":"<repository-url>",
         "SourceCodeVersion":{
```

```
            "Type":"Branch",
            "Value":"main"
         },
         "CodeConfiguration":{
            "ConfigurationSource":"API",
            "CodeConfigurationValues":{
               "Runtime":"<runtime>",
               "BuildCommand":"<build-command>",
               "StartCommand":"<start-command>",
               "Port":"<port>",
               "RuntimeEnvironmentSecrets":{
                  "Credential1" :
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXX",
                  "Credential2" : "arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
               }
            }
         }
      },
      "InstanceConfiguration": {
        "CPU": "1 vCPU",
        "Memory": "3 GB",
        "InstanceRoleArn: "<instance-role-arn>"
   }
...
}
```

3. apprunner.yaml モデルは、追加されたシークレットを反映するように更新されます。

   更新されたapprunner.yamlモデルの例を次に示します。

   Example **apprunner.yaml**

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - python -m pip install flask
run:
  command: python app.py
  network:
    port: 8080
  env:
```

```
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from:
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX"
    - name: my-parameter
      value-from: "arn:aws:ssm:<region>:<aws_account_id>:parameter/<parameter-
name>"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

App Runner を使用したネットワーキング

この章では、 AWS App Runner サービスのネットワーク設定について説明します。

この章では、以下について説明します。

- プライベートエンドポイントとパブリックエンドポイントの受信トラフィックを設定する方法。詳細については、「受信トラフィックのネットワーク設定のセットアップ」を参照してください。
- Amazon VPC で実行されている他のアプリケーションにアクセスするように送信トラフィックを設定する方法。詳細については、「送信トラフィックの VPC アクセスの有効化」を参照してください。

トピック

用語

ニーズに合わせてネットワークトラフィックをカスタマイズする方法を知るために、この章で使用されている以下の用語を理解しましょう。

一般的な用語

Amazon Virtual Private Cloud (VPC) に関連付けるために必要なことを知るために、次の用語を理解しましょう。

- VPC: Amazon VPC は、リソースの配置、接続、セキュリティなど、仮想ネットワーク環境を完全に制御できる論理的に隔離された仮想ネットワークです。これは、独自のデータセンターで運用する従来のネットワークによく似た仮想ネットワークです。
- VPC インターフェイスエンドポイント: リソースである VPC インターフェイスエンドポイントは、VPC をエンドポイントサービスに接続します。 AWS PrivateLink VPC インターフェイスエンドポイントを作成して、Network Load Balancer を使用してトラフィックを分散するエンドポイントサービスにトラフィックを送信します。エンドポイントサービス宛てのトラフィックは DNS を使用して解決されます。

- リージョン: 各リージョンは、App Runner サービスをホストできる個別の地理的エリアです。
- アベイラビリティーゾーン: アベイラビリティーゾーンは、 AWS リージョン内の独立した場所です。これは、冗長電源、ネットワーク、および接続を備えた 1 つ以上の個別のデータセンターです。アベイラビリティーゾーンは、本番環境アプリケーションの可用性、耐障害性、およびスケーラビリティを向上させるために役立ちます。
- サブネット: サブネットは VPC 内の IP アドレスの範囲です。サブネットは、1 つのアベイラビリティーゾーンに存在する必要があります。指定されたサブネットで AWS リソースを起動できます。インターネットに接続する必要があるリソースにはパブリックサブネットを、インターネットに接続しないリソースにはプライベートサブネットを使用してください。
- セキュリティグループ: セキュリティグループは、関連付けられているリソースに到達および残すことが許可されるトラフィックを制御します。セキュリティグループは、各サブネットの AWS リソースを保護するためのセキュリティレイヤーを追加し、ネットワークトラフィックをより詳細に制御できるようにします。VPC を作成すると、デフォルトのセキュリティグループが使用されます。VPC ごとに追加のセキュリティグループを作成できます。セキュリティグループは、それが作成された VPC 内のリソースにのみ関連付けることができます。
- デュアルスタック: デュアルスタックは、IPv4 エンドポイントと IPv6 エンドポイントの両方からのネットワークトラフィックをサポートするアドレスタイプです。

送信トラフィックの設定に固有の用語

VPC コネクタ

VPC Connector は、App Runner サービスがプライベート Amazon VPC で実行されるアプリケーションにアクセスできるようにする App Runner リソースです。

受信トラフィックの設定に固有の用語

Amazon VPC 内からのみサービスにプライベートにアクセスできるようにする方法については、次の用語を理解しましょう。

- VPC Ingress Connection: VPC Ingress Connection は、受信トラフィックの App Runner エンドポイントを提供する App Runner リソースです。App Runner コンソールで受信トラフィックのプライベートエンドポイントを選択すると、App Runner は VPC Ingress Connection リソースをバックグラウンドで割り当てます。VPC Ingress Connection リソースは、App Runner サービスを Amazon VPC の VPC インターフェイスエンドポイントに接続します。

> **Note**
>
> App Runner API を使用している場合、VPC Ingress Connection リソースは自動的に作成されません。

- プライベートエンドポイント: プライベートエンドポイントは、Amazon VPC 内からのみアクセスできるように受信ネットワークトラフィックを設定するために選択する App Runner コンソールオプションです。

受信トラフィックのネットワーク設定をセットアップする

プライベートエンドポイントまたはパブリックエンドポイントから受信トラフィックを受信するようにサービスを設定できます。

パブリックエンドポイントはデフォルト設定です。これにより、パブリックインターネットからの受信トラフィックに対してサービスが開きます。また、サービスの IPv4 またはデュアルスタック (IPv4 および IPv6) アドレスタイプを柔軟に選択できます。

プライベートエンドポイントでは、Amazon VPC からのトラフィックのみが App Runner サービスにアクセスできます。これは、App Runner サービスの VPC インターフェイスエンドポイントである AWS PrivateLink リソースをセットアップすることで実現されます。これにより、Amazon VPC と App Runner サービス間のプライベート接続が作成されます。また、サービスの IPv4 またはデュアルスタック (IPv4 および IPv6) アドレスタイプを柔軟に選択できます。

以下は、受信トラフィックのネットワーク設定の設定の一環として説明するトピックです。

- Amazon VPC 内からのみサービスをプライベートに利用できるように受信トラフィックを設定する方法。詳細については、「受信トラフィックのプライベートエンドポイントの有効化」を参照してください。
- デュアルスタックアドレスタイプからインターネットトラフィックを受信するようにサービスを設定する方法。詳細については、「パブリック受信トラフィックのデュアルスタックの有効化」を参照してください。

ヘッダー

App Runner を使用すると、アプリケーションに入るトラフィックの元のソース IPv4 アドレスと IPv6 アドレスにアクセスできます。元の送信元 IP アドレスは、`X-Forwarded-For`リクエストヘッ

ダーを割り当てることで保持されます。これにより、アプリケーションは必要に応じて元のソース IP アドレスを取得できます。

> **Note**
>
> サービスがプライベートエンドポイントを使用するように設定されている場合、`X-Forwarded-For`リクエストヘッダーを使用して元のソース IP アドレスにアクセスすることはできません。使用する場合、false 値を取得します。

受信トラフィックのプライベートエンドポイントの有効化

デフォルトでは、 AWS App Runner サービスを作成すると、そのサービスはインターネット経由でアクセスできます。ただし、App Runner サービスをプライベートにして、Amazon Virtual Private Cloud (Amazon VPC) 内からのみアクセスできるようにすることもできます。

App Runner サービスプライベートを使用すると、受信トラフィックを完全に制御でき、セキュリティレイヤーが追加されます。これは、内部 APIs、企業ウェブアプリケーション、またはより高いレベルのプライバシーとセキュリティを必要とする開発中のアプリケーション、または特定のコンプライアンス要件を満たす必要があるアプリケーションの実行など、さまざまなユースケースに役立ちます。

> **Note**
>
> App Runner アプリケーションにソース IP/CIDR 受信トラフィック制御ルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。これは、現在 WAF に関連付けられた App Runner プライベートサービスへのリクエストソース IP データの転送をサポートしていないためです。その結果、WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。
> ベストプラクティスを含むインフラストラクチャセキュリティとセキュリティグループの詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループを使用してネットワークトラフィックを制御し、AWS リソースへのトラフィックを制御する」のトピックを参照してください。 https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

App Runner サービスがプライベートの場合は、Amazon VPC 内からサービスにアクセスできます。インターネットゲートウェイ、NAT デバイス、または VPN 接続は必要ありません。

> **Note**
>
> App Runner は、受信トラフィックと送信トラフィックの両方で IPv4 とデュアルスタック (IPv4 と IPv6 の両方) をサポートします。

考慮事項

- App Runner の VPC インターフェイスエンドポイントを設定する前に、「 AWS PrivateLink ガイド」の「考慮事項」を参照してください。
- VPC エンドポイントポリシーは App Runner ではサポートされていません。デフォルトでは、VPC インターフェイスエンドポイントを介して App Runner へのフルアクセスが許可されます。または、セキュリティグループをエンドポイントネットワークインターフェイスに関連付けて、VPC インターフェイスエンドポイントを介して App Runner へのトラフィックを制御することもできます。
- App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。これは、現在 WAF に関連付けられた App Runner プライベートサービスへのリクエストソース IP データの転送をサポートしていないためです。その結果、WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。
- プライベートエンドポイントを有効にすると、サービスは VPC からのみアクセスでき、インターネットからアクセスすることはできません。
- 可用性を高めるには、VPC インターフェイスエンドポイントとは異なるアベイラビリティーゾーン全体で少なくとも 2 つのサブネットを選択することをお勧めします。サブネットを 1 つだけ使用することはお勧めしません。
- IP アドレスタイプのデュアルスタックオプションを選択する場合は、サブネットがデュアルスタックトラフィックをサポートできることを確認してください。
- 同じ VPC インターフェイスエンドポイントを使用して、VPC 内の複数の App Runner サービスにアクセスできます。

このセクションで使用される用語の詳細については、「 用語」を参照してください。

権限

プライベートエンドポイントを有効にするために必要なアクセス許可のリストを次に示します。

- ec2:CreateTags
- ec2:CreateVpcEndpoint
- ec2:ModifyVpcEndpoint
- ec2:DeleteVpcEndpoints
- ec2:DescribeSubnets
- ec2:DescribeVpcEndpoints
- ec2:DescribeVpcs

VPC インターフェイスエンドポイント

VPC インターフェイスエンドポイントは、Amazon VPC をエンドポイントサービスに接続する AWS PrivateLinkリソースです。VPC インターフェイスエンドポイントを渡すことで、App Runner サービスにアクセスできるようにする Amazon VPC を指定できます。VPC インターフェイスエンドポイントを作成するには、以下を指定します。

- 接続を有効にする Amazon VPC。
- セキュリティグループを追加します。デフォルトでは、セキュリティグループは VPC インターフェイスエンドポイントに割り当てられます。カスタムセキュリティグループを関連付けて、受信ネットワークトラフィックをさらに制御することを選択できます。
- サブネットを追加します。可用性を高めるために、App Runner サービスにアクセスするアベイラビリティーゾーンごとに少なくとも 2 つのサブネットを選択することをお勧めします。ネットワークインターフェイスエンドポイントは、VPC インターフェイスエンドポイントに対して有効にする各サブネットに作成されます。これらは、App Runner 宛てのトラフィックのエントリポイントとして機能するリクエスタ管理のネットワークインターフェイスです。リクエスタ管理のネットワークインターフェイスは、 AWS サービスがユーザーに代わって VPC に作成するネットワークインターフェイスです。
- API を使用している場合は、App Runner VPC インターフェイスエンドポイント を追加しますServicename。例えば、

```
com.amazonaws.region.apprunner.requests
```

VPC インターフェイスエンドポイントは、次のいずれか AWS のサービスを使用して作成できます。

- App Runner コンソール。詳細については、「プライベートエンドポイントの管理」を参照してください。
- Amazon VPC コンソールまたは API、および AWS Command Line Interface (AWS CLI)。詳細については、「AWS PrivateLink ガイド」の「AWS PrivateLinkから AWS のサービス にアクセスする」を参照してください。

> **Note**
>
> AWS PrivateLink 料金に基づいて、使用する VPC インターフェイスエンドポイントごとに課金されます。したがって、コスト効率を向上させるために、同じ VPC インターフェイスエンドポイントを使用して VPC 内の複数の App Runner サービスにアクセスできます。ただし、分離を改善するには、App Runner サービスごとに異なる VPC インターフェイスエンドポイントを関連付けることを検討してください。

VPC イングレス接続

VPC Ingress Connection は、受信トラフィックの App Runner エンドポイントを指定する App Runner リソースです。App Runner コンソールで受信トラフィックのプライベートエンドポイントを選択すると、App Runner は VPC Ingress Connection リソースをバックグラウンドで割り当てます。このオプションを選択すると、Amazon VPC からのトラフィックのみが App Runner サービスにアクセスできるようになります。VPC Ingress Connection リソースは、App Runner サービスを Amazon VPC の VPC インターフェイスエンドポイントに接続します。VPC Ingress Connection リソースは、API オペレーションを使用して受信トラフィックのネットワーク設定を構成している場合にのみ作成できます。VPC Ingress Connection リソースを作成する方法の詳細については、 AWS App Runner API リファレンスのCreateVpcIngressConnection」を参照してください。

> **Note**
>
> App Runner の 1 つの VPC Ingress Connection リソースは、Amazon VPC の 1 つの VPC インターフェイスエンドポイントに接続できます。また、App Runner サービスごとに作成できる VPC Ingress Connection リソースは 1 つだけです。

プライベートエンドポイント

プライベートエンドポイントは、Amazon VPC から受信トラフィックのみを受信する場合に選択できる App Runner コンソールオプションです。App Runner コンソールでプライベートエンドポイントオプションを選択すると、VPC インターフェイスエンドポイントを設定してサービスを VPC に接続するオプションが提供されます。バックグラウンドでは、App Runner は設定した VPC インターフェイスエンドポイントに VPC Ingress Connection リソースを割り当てます。

概要

Amazon VPC からのトラフィックのみが App Runner サービスにアクセスできるようにすることで、サービスをプライベートにします。これを実現するには、App Runner または Amazon VPC を使用して、選択した Amazon VPC の VPC インターフェイスエンドポイントを作成します。App Runner コンソールで、着信トラフィックのプライベートエンドポイントを有効にすると、VPC インターフェイスエンドポイントが作成されます。App Runner は自動的に VPC Ingress Connection リソースを作成し、VPC インターフェイスエンドポイントと App Runner サービスに接続します。これにより、選択した VPC からのトラフィックのみが App Runner サービスにアクセスできるようにするプライベートサービス接続が作成されます。

プライベートエンドポイントの管理

次のいずれかの方法を使用して、受信トラフィックのプライベートエンドポイントを管理します。

- the section called “App Runner コンソール”
- the section called “App Runner API または AWS CLI”

> **Note**
>
> App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。その結果、WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。
> ベストプラクティスを含むインフラストラクチャセキュリティとセキュリティグループの詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループを使用してネットワークトラフィックを制御する」および「AWS リソースへのトラフィックを制御する」

のトピックを参照してください。 https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

App Runner コンソール

App Runner コンソールを使用してサービスを作成する場合、または後でその設定を更新する場合は、受信トラフィックの設定を選択できます。

受信トラフィックを設定するには、次のいずれかを選択します。

- パブリックエンドポイント: インターネット経由ですべてのサービスがサービスにアクセスできるようにします。デフォルトでは、パブリックエンドポイントが選択されています。
- プライベートエンドポイント: Amazon VPC 内からのみ App Runner サービスにアクセスできるようにします。

プライベートエンドポイントを有効にする

アクセスする Amazon VPC の VPC インターフェイスエンドポイントに関連付けることで、プライベートエンドポイントを有効にします。新しい VPC インターフェイスエンドポイントを作成するか、既存のエンドポイントを選択できます。

VPC インターフェイスエンドポイントを作成するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. 「サービスの設定」の「ネットワーク」セクションに移動します。
3. 受信ネットワークトラフィックにプライベートエンドポイントを選択します。VPC インターフェイスエンドポイントを使用して VCP に接続するオプションが開きます。
4. 新しいエンドポイントの作成 を選択します。新しい VPC インターフェイスエンドポイントの作成ダイアログボックスが開きます。
5. VPC インターフェイスエンドポイントの名前を入力します。
6. ドロップダウンリストから必要な VPC インターフェイスエンドポイントを選択します。
7. ドロップダウンリストからセキュリティグループを選択します。セキュリティグループを追加すると、VPC インターフェイスエンドポイントにセキュリティレイヤーが追加されます。2 つ以上のセキュリティグループを選択することをお勧めします。セキュリティグループを選択しない場合、App Runner は VPC インターフェイスエンドポイントにデフォルトのセキュリティグ

ループを割り当てます。セキュリティグループルールが App Runner サービスと通信するリソースをブロックしないようにしてください。セキュリティグループルールでは、App Runner サービスとやり取りするリソースを許可する必要があります。

> **Note**
>
> App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。その結果、WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。
> ベストプラクティスを含むインフラストラクチャセキュリティとセキュリティグループの詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループを使用してネットワークトラフィックを制御し、AWS リソースへのトラフィックを制御する」のトピックを参照してください。 https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

8. ドロップダウンリストから必要なサブネットを選択します。App Runner サービスにアクセスするアベイラビリティーゾーンごとに少なくとも 2 つのサブネットを選択することをお勧めします。

> **Note**
>
> デュアルスタック用にエンドポイントを設定する場合は、インフラストラクチャと VPC エンドポイントがデュアルスタックトラフィックをサポートしていることを確認してください。

9. (オプション) 新しいタグを追加を選択し、タグキーとタグ値を入力します。
10. [作成] を選択します。サービスの設定ページが開き、上部のバーに VPC インターフェイスエンドポイントが正常に作成されたことを示すメッセージが表示されます。

既存の VPC インターフェイスエンドポイントを選択するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. 「サービスの設定」の「ネットワーク」セクションに移動します。

3. 受信ネットワークトラフィックにプライベートエンドポイントを選択します。VPC インターフェイスエンドポイントを使用して VPC に接続するオプションが開きます。使用可能な VPC インターフェイスエンドポイントのリストが表示されます。
4. VPC インターフェイスエンドポイントにリストされている必要な VPC インターフェイスエンドポイントを選択します。
5. 次へ を選択してサービスを作成します。App Runner はプライベートエンドポイントを有効にします。

> **Note**
>
> サービスを作成したら、必要に応じて VPC インターフェイスエンドポイントに関連付けられたセキュリティグループとサブネットを編集できます。

プライベートエンドポイントの詳細を確認するには、サービスに移動し、設定タブのネットワークセクションを展開します。プライベートエンドポイントに関連付けられた VPC と VPC インターフェイスエンドポイントの詳細が表示されます。

VPC インターフェイスエンドポイントを更新する

App Runner サービスを作成したら、プライベートエンドポイントに関連付けられた VPC インターフェイスエンドポイントを編集できます。

> **Note**
>
> エンドポイント名と VPC フィールドを更新することはできません。

VPC インターフェイスエンドポイントを更新するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. サービスに移動し、左側のパネルでネットワーク設定を選択します。
3. 受信トラフィックを選択して、それぞれのサービスに関連付けられている VPC インターフェイスエンドポイントを表示します。
4. 編集する VPC インターフェイスエンドポイントを選択します。
5. [編集] を選択します。VPC インターフェイスエンドポイントを編集するダイアログボックスが開きます。

6. 必要なセキュリティグループとサブネットを選択し、更新をクリックします。VPC インターフェイスエンドポイントの詳細を示すページが開き、上部のバーに VPC インターフェイスエンドポイントが正常に更新されたというメッセージが表示されます。

> **Note**
>
> App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。その結果、WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。
> ベストプラクティスを含むインフラストラクチャセキュリティとセキュリティグループの詳細については、「Amazon VPC ユーザーガイド」の次のトピックを参照してください。セキュリティグループを使用してネットワークトラフィックを制御し、AWS リソースへのトラフィックを制御します。 https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

VPC インターフェイスエンドポイントを削除する

App Runner サービスにプライベートアクセスを許可しない場合は、受信トラフィックを Public に設定できます。をパブリックに変更すると、プライベートエンドポイントは削除されますが、VPC インターフェイスエンドポイントは削除されません。

VPC インターフェイスエンドポイントを削除するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. サービスに移動し、左側のパネルでネットワーク設定を選択します。
3. 受信トラフィックを選択して、それぞれのサービスに関連付けられている VPC インターフェイスエンドポイントを表示します。

> **Note**
>
> VPC インターフェイスエンドポイントを削除する前に、サービスを更新して、接続されているすべてのサービスから削除します。

4. [削除] を選択します。

   VPC インターフェイスエンドポイントに接続されているサービスがある場合、VPC インターフェイスエンドポイントを削除できないというメッセージが表示されます。VPC インターフェイスエンドポイントに接続されているサービスがない場合は、削除を確認するメッセージが表示されます。

5. [削除] を選択します。ネットワーク設定ページが開き、上部のバーに VPC インターフェイスエンドポイントが正常に削除されたというメッセージが表示されます。

App Runner API または AWS CLI

Amazon VPC 内からのみアクセスできるアプリケーションを App Runner にデプロイできます。

サービスをプライベートにするために必要なアクセス許可については、「」を参照してくださいthe section called “権限”。

Amazon VPC へのプライベートサービス接続を作成するには

1. App Runner に接続する VPC インターフェイスエンドポイント、 AWS PrivateLink リソースを作成します。これを行うには、アプリケーションに関連付けるサブネットとセキュリティグループを指定します。VPC インターフェイスエンドポイントを作成する例を次に示します。

   > **Note**
   >
   > App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。その結果、WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。
   > ベストプラクティスを含むインフラストラクチャセキュリティとセキュリティグループの詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループを使用したネットワークトラフィックの制御」および「AWS リソースへのトラフィックの制御」のトピックを参照してください。 https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

Example

```
aws ec2 create-vpc-endpoint
 --vpc-endpoint-type: Interface
 --service-name: com.amazonaws.us-east-1.apprunner.requests
 --subnets: subnet1, subnet2
 --security-groups: sg1
```

2. CLI で CreateService または UpdateService App Runner API アクションを使用して、VPC インターフェイスエンドポイントを参照します。パブリックにアクセスできないようにサービスを設定します。NetworkConfiguration パラメータIngressConfigurationのメンバーFalseで を IsPubliclyAccessibleに設定します。必要に応じて、IpAddressTypeフィールドを IPV4または に設定できますDUAL_STACK。設定しない場合、この値はデフォルトで IPV4 になります。次の例では、VPC インターフェイスエンドポイントを参照しています。

   Example

```
aws apprunner create-service
 --network-configuration:
   {
   "IngressConfiguration":
      {
      "IsPubliclyAccessible": False
      },
      "IpAddressType": "IPV4"
   }
 --service-name: com.amazonaws.us-east-1.apprunner.requests
 --source-configuration: <source_configuration>
```

3. create-vpc-ingress-connection API アクションを呼び出して App Runner の VPC Ingress Connection リソースを作成し、前のステップで作成した VPC インターフェイスエンドポイントに関連付けます。指定された VPC 内のサービスへのアクセスに使用されるドメイン名を返します。VPC Ingress Connection リソースを作成する例を次に示します。

   Exampleリクエスト

```
aws apprunner create-vpc-ingress-connection
 --service-arn: <apprunner_service_arn>
 --ingress-vpc-configuration: {"VpcId":<vpc_id>, "VpceId": <vpce_id>}
```

```
  --vpc-ingress-connection-name: <vic_connection_name>
```

Exampleレスポンス

```
{
   "VpcIngressConnectionArn": <vpc_ingress_connection_arn>,
   "VpcIngressConnectionName": <vic_connection_name>,
   "ServiceArn": <apprunner_service_arn>,
   "Status": "PENDING_CREATION",
   "AccountId": <connection_owner_id>,
   "DomainName": <domain_name_associated_with_vpce>,
   "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
   "CreatedAt": <date_created>
}
```

VPC Ingress Connection の更新

VPC Ingress Connection リソースを更新できます。VPC Ingress Connection を更新するには、次のいずれかの状態である必要があります。

- 利用可能
- FAILED_CREATION
- FAILED_UPDATE

VPC Ingress Connection リソースを更新する例を次に示します。

Exampleリクエスト

```
aws apprunner update-vpc-ingress-connection
      --vpc-ingress-connection-arn: <vpc_ingress_connection_arn>
```

Exampleレスポンス

```
{
    "VpcIngressConnectionArn":  <vpc_ingress_connection_arn>,
    "VpcIngressConnectionName":  <vic_connection_name>,
    "ServiceArn":  <apprunner_service_arn>,
    "Status": "FAILED_UPDATE",
    "AccountId":  <connection_owner_id>,
```

```
    "DomainName":  <domain_name_associated_with_vpce>,
    "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
    "CreatedAt":  <date_created>
}
```

VPC Ingress Connection を削除する

Amazon VPC へのプライベート接続が不要になった場合は、VPC Ingress Connection リソースを削除できます。

VPC Ingress Connection を削除するには、次のいずれかの状態である必要があります。

- 利用可能
- 失敗した作成
- 更新に失敗
- 削除に失敗した

VPC Ingress Connection を削除する例を次に示します。

Exampleリクエスト

```
aws apprunner delete-vpc-ingress-connection
      --vpc-ingress-connection-arn: <vpc_ingress_connection_arn>
```

Exampleレスポンス

```
{
   "VpcIngressConnectionArn": <vpc_ingress_connection_arn>,
   "VpcIngressConnectionName": <vic_connection_name>,
   "ServiceArn": <apprunner_service_arn>,
   "Status": "PENDING_DELETION",
   "AccountId": <connection_owner_id>,
   "DomainName": <domain_name_associated_with_vpce>,
   "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
   "CreatedAt": <date_created>,
   "DeletedAt": <date_deleted>
}
```

次の App Runner API アクションを使用して、サービスのプライベートインバウンドトラフィックを管理します。

- CreateVpcIngressConnection – 新しい VPC Ingress Connection リソースを作成します。App Runner サービスを Amazon VPC エンドポイントに関連付ける場合、App Runner にはこのリソースが必要です。
- ListVpcIngressConnections – AWS アカウントに関連付けられている AWS App Runner VPC Ingress Connection エンドポイントのリストを返します。
- DescribeVpcIngressConnection – AWS App Runner VPC Ingress Connection リソースの完全な説明を返します。
- UpdateVpcIngressConnection – AWS App Runner VPC Ingress Connection リソースを更新します。
- DeleteVpcIngressConnection – App Runner サービスに関連付けられている App Runner VPC Ingress Connection リソースを削除します。

App Runner API の使用の詳細については、「App Runner API リファレンスガイド」を参照してください。

受信トラフィックの IPv6 の有効化

サービスが IPv6 アドレスまたは IPv4 アドレスと IPv6 アドレスの両方から受信ネットワークトラフィックを受信する場合は、エンドポイントのデュアルスタックアドレスタイプを選択します。新しいアプリケーションを作成するときは、「サービスの設定 > ネットワーク」セクションでこの設定を確認できます。次の手順では、App Runner コンソールまたは App Runner API を使用して IPv4 またはデュアルスタック (IPv6 および IPv4) を有効にする方法について説明します。

受信トラフィックのデュアルスタックの管理

次のいずれかの方法を使用して、受信トラフィックのデュアルスタックアドレスタイプを管理します。

- the section called “App Runner コンソール”
- the section called “App Runner API または AWS CLI”

> **Note**
>
> 次の手順では、パブリック受信トラフィックのネットワークアドレスタイプを管理する方法について説明します。プライベートエンドポイントのデュアルスタックまたは IPv4 アドレ

スタイプの管理については、「」を参照してくださいthe section called “プライベートエンドポイントの管理”。

App Runner コンソール

着信インターネットトラフィックのデュアルスタックアドレスタイプは、App Runner コンソールを使用してサービスを作成するとき、または後で設定を更新するときに選択できます。

デュアルスタックのアドレスタイプを有効にするには

1. サービスを作成または更新するときは、「サービスの設定」の「ネットワーク」セクションを展開します。
2. 受信ネットワークトラフィックのパブリックエンドポイントを選択します。パブリックエンドポイントを選択すると、エンドポイント IP アドレスタイプオプションが開きます。

   プライベートエンドポイントのデュアルスタックまたは IPv4 アドレスタイプを管理する手順the section called “プライベートエンドポイントの管理”については、「」を参照してください。
3. エンドポイント IP アドレスタイプを展開して、次の IP アドレスタイプを表示します。

   - IPv4
   - デュアルスタック (IPv4 および IPv6)

   > **Note**
   >
   > エンドポイント IP アドレスタイプを展開して選択しない場合、App Runner は IPv4 をデフォルト設定として割り当てます。
4. デュアルスタック (IPv4 および IPv6) を選択します。
5. サービスを作成する場合は、次へを選択し、次に作成とデプロイを選択します。それ以外の場合は、サービスを更新する場合は変更の保存を選択します。

   サービスがデプロイされると、アプリケーションは IPv4 エンドポイントと IPv6 エンドポイントの両方からネットワークトラフィックの受信を開始します。

アドレスタイプを変更するには

1. 手順に従ってサービスを更新し、ネットワーキングに移動します。

2. 着信ネットワークトラフィックのエンドポイント IP アドレスタイプに移動し、必要なアドレスタイプを選択します。
3. [Save changes] (変更の保存) をクリックします。選択した内容でサービスが更新されます。

App Runner API または AWS CLI

CreateService または UpdateService App Runner API アクションを呼び出すときは、NetworkConfigurationパラメータIpAddressTypeのメンバーを使用してアドレスタイプを指定します。指定できるサポートされている値は、 IPv4と ですDUAL_STACK。サービスが IPv4 および IPv6 エンドポイントからインターネットトラフィックを受信するDUAL_STACKかどうかを指定します。に値を指定しない場合IpAddressType、デフォルトで IPv4 が適用されます。

> **Note**
>
> プライベートエンドポイントの例については、「」を参照してくださいthe section called "App Runner API または AWS CLI"。

以下は、デュアルスタックを IP アドレスとしてサービスを作成する例です。この例では、input.jsonファイルを呼び出します。

Exampleデュアルスタックをサポートするサービスを作成するリクエスト

```
aws apprunner create-service \
 --cli-input-json file://input.json
```

Example **input.json** の内容

```
{
  "ServiceName": "example-service",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageIdentifier": "public.ecr.aws/aws-containers/hello-app-runner:latest",
      "ImageConfiguration": {
        "Port": "8000"
      },
      "ImageRepositoryType": "ECR_PUBLIC"
    },
    "NetworkConfiguration": {
```

```
      "IpAddressType": "DUAL_STACK"
    }
  }
}
```

Example応答

```
{
  "Service": {
    "ServiceName": "example-service",
    "ServiceId": "<service-id>",
    "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/example-
service/<service-id>",
    "ServiceUrl": "1234567890.us-east-2.awsapprunner.com",
    "CreatedAt": "2023-10-16T12:30:51.724000-04:00",
    "UpdatedAt": "2023-10-16T12:30:51.724000-04:00",
    "Status": "OPERATION_IN_PROGRESS",
    "SourceConfiguration": {
      "ImageRepository": {
        "ImageIdentifier": "public.ecr.aws/aws-containers/hello-app-runner:latest",
        "ImageConfiguration": {
          "Port": "8000"
        },
        "ImageRepositoryType": "ECR_PUBLIC"
      },
      "AutoDeploymentsEnabled": false
    },
    "InstanceConfiguration": {
      "Cpu": "1024",
      "Memory": "2048"
    },
    "HealthCheckConfiguration": {
      "Protocol": "TCP",
      "Path": "/",
      "Interval": 5,
      "Timeout": 2,
      "HealthyThreshold": 1,
      "UnhealthyThreshold": 5
    },
    "AutoScalingConfigurationSummary": {
      "AutoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
```

```
      "AutoScalingConfigurationName": "DefaultConfiguration",
      "AutoScalingConfigurationRevision": 1
    },
    "NetworkConfiguration": {
      "IpAddressType": "DUAL_STACK",
      "EgressConfiguration": {
        "EgressType": "DEFAULT"
      },
      "IngressConfiguration": {
        "IsPubliclyAccessible": true
      }
    }
  },
  "OperationId": "24bd100b1e111ae1a1f0e1115c4f11de"
}
```

API パラメータの詳細については、NetworkConfiguration」を参照してください。

送信トラフィックの VPC アクセスの有効化

デフォルトでは、 AWS App Runner アプリケーションはパブリックエンドポイントにメッセージを送信できます。これには、独自のソリューション AWS のサービス、およびその他のパブリックウェブサイトまたはウェブサービスが含まれます。アプリケーションは、Amazon Virtual Private Cloud (Amazon VPC) から VPC で実行されるアプリケーションのパブリックエンドポイントにメッセージを送信することもできます。環境の起動時に VPC を設定しない場合、App Runner はパブリックであるデフォルトの VPC を使用します。

カスタム VPC で環境を起動して、送信トラフィックのネットワーク設定とセキュリティ設定をカスタマイズできます。 AWS App Runner サービスが Amazon Virtual Private Cloud (Amazon VPC) からプライベート VPC で実行されるアプリケーションにアクセスできるようにします。これを行うと、アプリケーションは に接続し、Amazon Virtual Private Cloud (Amazon VPC) でホストされている他のアプリケーションにメッセージを送信できます。例としては、Amazon RDS データベース、Amazon ElastiCache、およびプライベート VPC でホストされているその他のプライベートサービスがあります。

VPC コネクタ

VPC Connector と呼ばれる App Runner コンソールから VPC エンドポイントを作成することで、サービスを VPC に関連付けることができます。VPC コネクタを作成するには、VPC、1 つ以上のサ

ブネット、およびオプションで 1 つ以上のセキュリティグループを指定します。VPC Connector を設定したら、1 つ以上の App Runner サービスで使用できます。

1 回限りのレイテンシー

アウトバウンドトラフィック用にカスタム VPC コネクタを使用して App Runner サービスを設定すると、1 回限りの起動レイテンシーが 2～5 分になることがあります。起動プロセスは、VPC Connector が他のリソースに接続する準備ができるまで待機してから、サービスステータスを実行中に設定します。サービスの作成時にカスタム VPC コネクタを使用してサービスを設定することも、サービスの更新を行って後で設定することもできます。

同じ VPC コネクタ設定を別のサービスに再利用する場合、レイテンシーは発生しないことに注意してください。VPC コネクタの設定は、セキュリティグループとサブネットの組み合わせに基づいています。特定の VPC コネクタ設定では、レイテンシーは VPC Connector Hyperplane ENIs (エラスティックネットワークインターフェイス) の初回作成時に 1 回のみ発生します。

カスタム VPC コネクタと AWS Hyperplane の詳細

App Runner の VPC コネクタは、Network Load Balancer、NAT Gateway、AWS PrivateLink などの複数の AWS リソースの背後にある内部 Amazon ネットワークシステムである AWS Hyperplane に基づいています。 AWS Hyperplane テクノロジーは、高スループットと低レイテンシーの機能に加えて、より高いレベルの共有を提供します。Hyperplane ENI は、VPC コネクタを作成してサービスに関連付けるときにサブネットに作成されます。VPC コネクタの設定はセキュリティグループとサブネットの組み合わせに基づいており、複数の App Runner サービスで同じ VPC コネクタを参照できます。その結果、基盤となる Hyperplane ENIs は App Runner サービス間で共有されます。この共有は、リクエストの負荷を処理するために必要なタスクの数をスケールアップしても実行可能であり、VPC 内の IP スペースをより効率的に活用できます。詳細については、AWS コンテナブログの「Deep Dive on AWS App Runner VPC Networking」を参照してください。

サブネット

各サブネットは、特定のアベイラビリティーゾーンにあります。高可用性を実現するには、少なくとも 3 つのアベイラビリティーゾーンのサブネットを選択することをお勧めします。リージョンのアベイラビリティーゾーンが 3 つ未満の場合は、サポートされているすべてのアベイラビリティーゾーンでサブネットを選択することをお勧めします。

VPC のサブネットを選択するときは、パブリックサブネットではなくプライベートサブネットを選択してください。これは、VPC コネクタを作成すると、App Runner サービスが各サブネットに Hyperplane ENI を作成するためです。各 Hyperplane ENI にはプライベート IP アドレスのみが割り当てられ、AWSAppRunnerManaged キーのタグが付けられます。パブリックサブネットを選択する

と、App Runner サービスの実行時にエラーが発生します。ただし、インターネット上の一部のサービスや他のパブリックサービスにアクセスする必要がある場合は AWS のサービス、「」を参照してくださいthe section called “サブネットを選択する際の考慮事項”。

サブネットを選択する際の考慮事項

- サービスを VPC に接続すると、アウトバウンドトラフィックはパブリックインターネットにアクセスできません。アプリケーションからのすべてのアウトバウンドトラフィックは、サービスが接続されている VPC を介して送信されます。VPC のすべてのネットワークルールは、アプリケーションのアウトバウンドトラフィックに適用されます。つまり、サービスはパブリックインターネットと AWS APIs にアクセスできません。アクセスを取得するには、次のいずれかを実行します。
  - NAT ゲートウェイを介してサブネットをインターネットに接続します。
  - アクセス AWS のサービス する の VPC エンドポイントを設定します。サービスは、 を使用して Amazon VPC 内にとどまります AWS PrivateLink。
- の一部のアベイラビリティーゾーンは、App Runner サービスで使用できるサブネットをサポート AWS リージョン していません。これらのアベイラビリティーゾーンでサブネットを選択すると、サービスの作成または更新に失敗します。このような状況では、App Runner はサポートされていないサブネットとアベイラビリティーゾーンを示す詳細なエラーメッセージを提供します。これが発生した場合は、サポートされていないサブネットをリクエストから削除してトラブルシューティングを行い、もう一度試してください。
- 選択したサブネットはすべて、IPv4 またはデュアルスタックのいずれかの同じ IP アドレスタイプである必要があります。

セキュリティグループ

オプションで、指定したサブネット AWS で App Runner がアクセスするために使用するセキュリティグループを指定できます。セキュリティグループを指定しない場合、App Runner は VPC のデフォルトのセキュリティグループを使用します。デフォルトのセキュリティグループでは、すべてのアウトバウンドトラフィックを許可します。

セキュリティグループを追加すると、VPC コネクタにセキュリティレイヤーが追加され、ネットワークトラフィックをより詳細に制御できます。VPC Connector は、アプリケーションからのアウトバウンド通信にのみ使用されます。アウトバウンドルールを使用して、目的の送信先エンドポイントへの通信を許可します。また、送信先リソースに関連付けられているセキュリティグループに適切なインバウンドルールがあることを確認する必要があります。それ以外の場合、これらのリソースは VPC Connector セキュリティグループからのトラフィックを受け入れることができません。

> **Note**
>
> サービスを VPC に関連付けると、次のトラフィックは影響を受けません。
>
> - インバウンドトラフィック – アプリケーションが受信する受信メッセージは、関連付けられた VPC の影響を受けません。メッセージは、サービスに関連付けられているパブリックドメイン名を介してルーティングされ、VPC とやり取りしません。
> - App Runner トラフィック – App Runner は、ソースコードとイメージのプル、ログのプッシュ、シークレットの取得など、ユーザーに代わっていくつかのアクションを管理します。これらのアクションが生成するトラフィックは、VPC 経由でルーティングされません。

と Amazon VPC AWS App Runner の統合方法の詳細については、AWS 「App Runner VPC Networking」を参照してください。

VPC アクセスの管理

> **Note**
>
> サービスのアウトバウンドトラフィック VPC コネクタを作成すると、次のサービス起動プロセスで 1 回限りのレイテンシーが発生します。この設定は、新しいサービスの作成時または作成後にサービスの更新で設定できます。詳細については、このガイド1 回限りのレイテンシーの「Networking with App Runner」の章の「」を参照してください。

次のいずれかの方法を使用して、App Runner サービスの VPC アクセスを管理します。

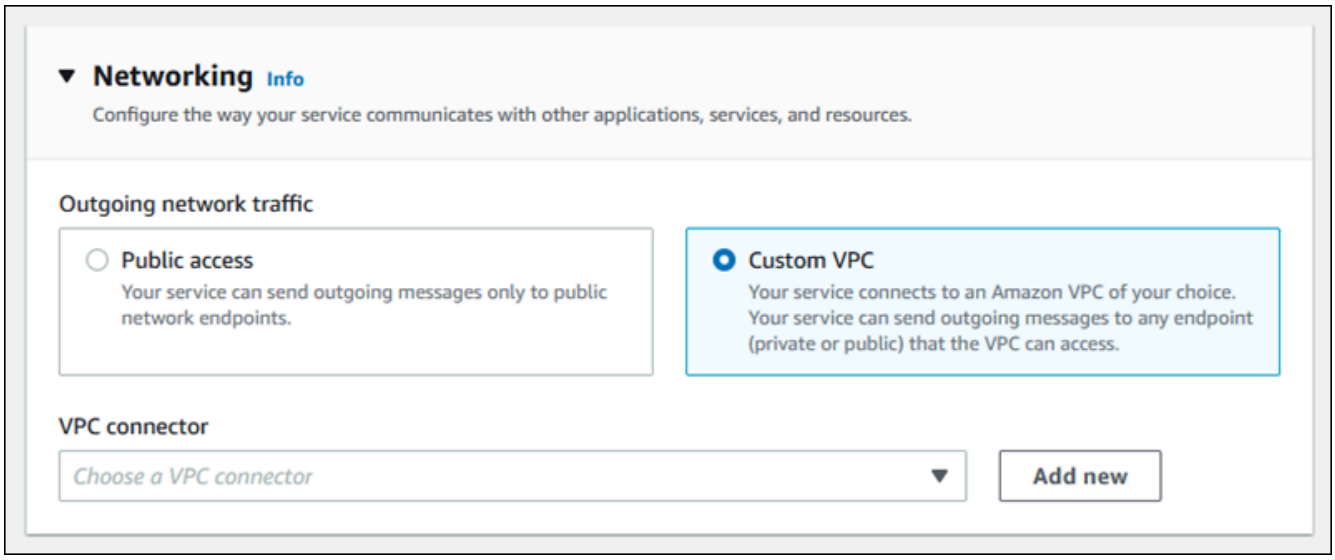
App Runner console

App Runner コンソールを使用してサービスを作成する場合、または後でその設定を更新する場合は、送信トラフィックの設定を選択できます。コンソールページのネットワーク設定セクションを探します。送信ネットワークトラフィックの場合は、以下から を選択します。

- パブリックアクセス: サービスを他の のパブリックエンドポイントに関連付けるには AWS のサービス。
- カスタム VPC: サービスを Amazon VPC の VPC に関連付けるには。アプリケーションは に接続し、Amazon VPC でホストされている他のアプリケーションにメッセージを送信できます。

カスタム VPC を有効にするには

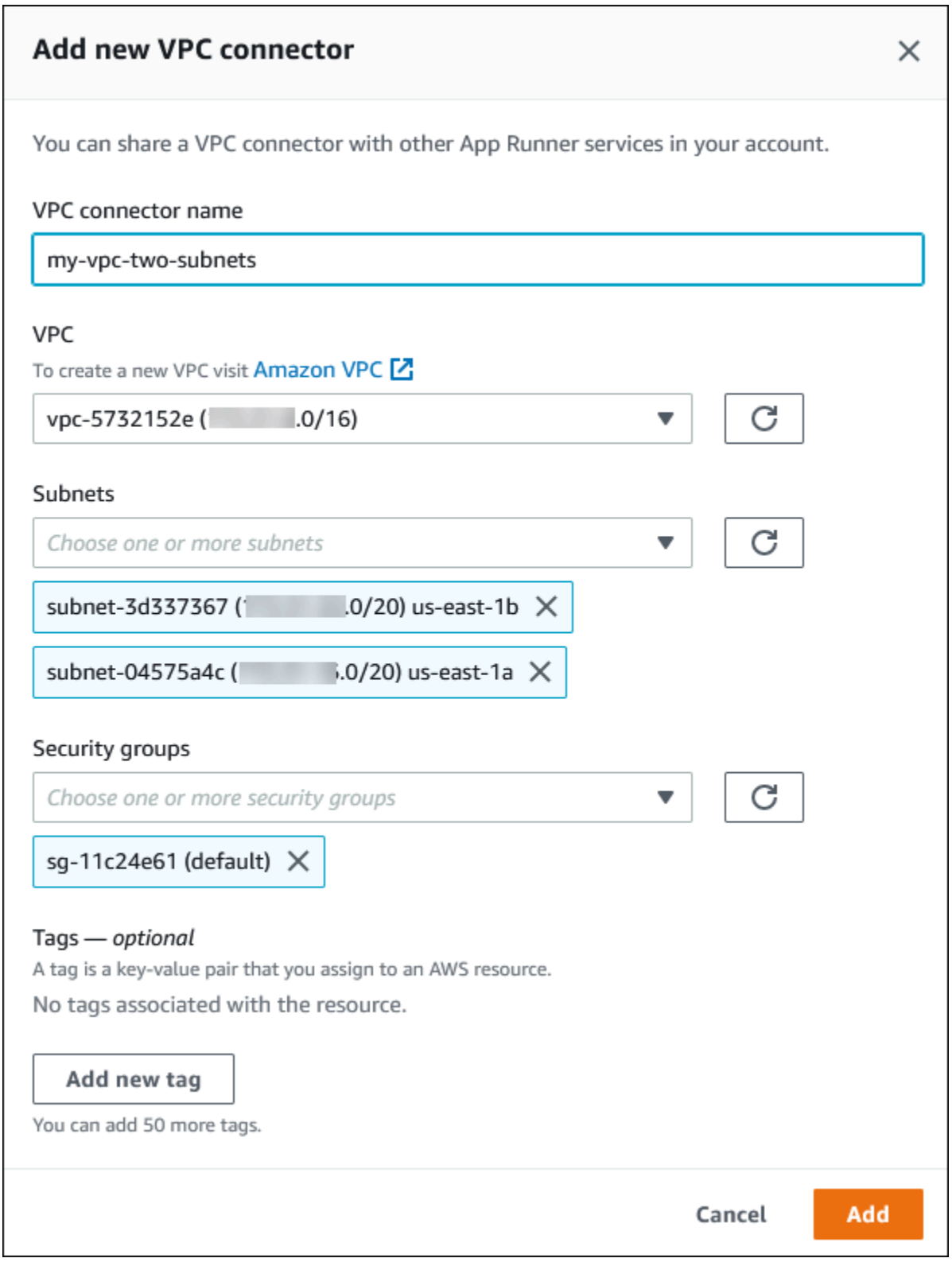
1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. 「サービスの設定」の「ネットワーク」セクションに移動します。



3. 送信ネットワークトラフィックのカスタム VPC を選択します。
4. ナビゲーションペインで、VPC コネクタを選択します。

   VPC コネクタを作成した場合、コンソールにはアカウント内の VPC コネクタのリストが表示されます。既存の VPC コネクタを選択し、次へ を選択して設定を確認できます。次に、最後のステップに進みます。または、次のステップを使用して新しい VPC コネクタを追加することもできます。
5. 新規追加 を選択して、サービスの新しい VPC コネクタを作成します。

   次に、新しい VPC コネクタの追加ダイアログボックスが開きます。



6. VPC コネクタの名前を入力し、使用可能なリストから必要な VPC を選択します。

7. Subnets では、App Runner サービスにアクセスする予定のアベイラビリティーゾーンごとに 1 つのサブネットを選択します。可用性を高めるには、3 つのサブネットを選択します。または、サブネットが 3 つ未満の場合は、使用可能なすべてのサブネットを選択します。

   > **Note**
   >
   > - VPC コネクタにプライベートサブネットを割り当ててください。VPC コネクタにパブリックサブネットを割り当てると、更新中にサービスが自動的に作成またはロールバックされません。
   > - 送信トラフィックがデュアルスタックの場合は、選択したすべてのサブネットが VPC コンソールでデュアルスタック用に設定されていることを確認します。

8. (オプション) セキュリティグループで、エンドポイントネットワークインターフェイスに関連付けるセキュリティグループを選択します。
9. (オプション) タグを追加するには、[新しいタグを追加] を選択し、そのタグのキーと値を入力してください。
10. [Add] (追加) を選択します。

    作成した VPC コネクタの詳細は、VPC コネクタの下に表示されます。
11. Next を選択して設定を確認し、Create and deploy を選択します。

    App Runner は VPC コネクタリソースを作成し、それをサービスに関連付けます。サービスが正常に作成されると、コンソールにサービスダッシュボードが表示され、新しいサービスの概要が表示されます。

App Runner API or AWS CLI

CreateService または UpdateService App Runner API アクションを呼び出すときは、`NetworkConfiguration`パラメータ`EgressConfiguration`のメンバーを使用して、サービスの VPC コネクタリソースを指定します。

次の App Runner API アクションを使用して、VPC Connector リソースを管理します。

- CreateVpcConnector – 新しい VPC コネクタを作成します。
- ListVpcConnectors – に関連付けられている VPC コネクタのリストを返します AWS アカウント。リストには詳細な説明が含まれています。
- DescribeVpcConnector – VPC コネクタの完全な説明を返します。

- DeleteVpcConnector – VPC コネクタを削除します。の VPC コネクタクォータに達した場合は AWS アカウント、不要な VPC コネクタを削除する必要がある場合があります。

VPC へのアウトバウンドアクセス権を持つアプリケーションを App Runner にデプロイするには、まず VPC Connector を作成する必要があります。これを行うには、アプリケーションに関連付ける 1 つ以上のサブネットとセキュリティグループを指定します。その後、次の例に示すように、CLI を使用して Create または UpdateService で VPC Connector を参照できます。

```
            cat > vpc-connector.json <<EOF
{
"VpcConnectorName": "my-vpc-connector",
"Subnets": [
"subnet-a",
"subnet-b",
"subnet-c"
],
"SecurityGroups": [
"sg-1",
"sg-2"
]
}
EOF

aws apprunner create-vpc-connector \
--cli-input-json file:///vpc-connector.json

cat > service.json <<EOF

{
"ServiceName": "my-vpc-connected-service",
"SourceConfiguration": {
"ImageRepository": {
"ImageIdentifier": "<ecr-image-identifier> ",
"ImageConfiguration": {
"Port": "8000"
},
"ImageRepositoryType": "ECR"
}
},
"NetworkConfiguration": {
"EgressConfiguration": {
```

```
"EgressType": "VPC",
"VpcConnectorArn": "arn:aws:apprunner:..../my-vpc-connector"
}
}
}
EOF

aws apprunner create-service \
--cli-input-json file:///service.js
```

App Runner サービスのオブザーバビリティ

AWS App Runner は複数の AWS サービスと統合され、App Runner サービス用の広範なオブザーバビリティツールスイートを提供します。この章のトピックでは、これらの機能について説明します。

トピック

App Runner サービスアクティビティの追跡

AWS App Runner はオペレーションのリストを使用して、App Runner サービスのアクティビティを追跡します。オペレーションは、サービスの作成、設定の更新、サービスのデプロイなど、API アクションへの非同期呼び出しを表します。以下のセクションでは、App Runner コンソールで API を使用してアクティビティを追跡する方法を示します。

App Runner サービスアクティビティを追跡する

次のいずれかの方法を使用して、App Runner サービスアクティビティを追跡します。

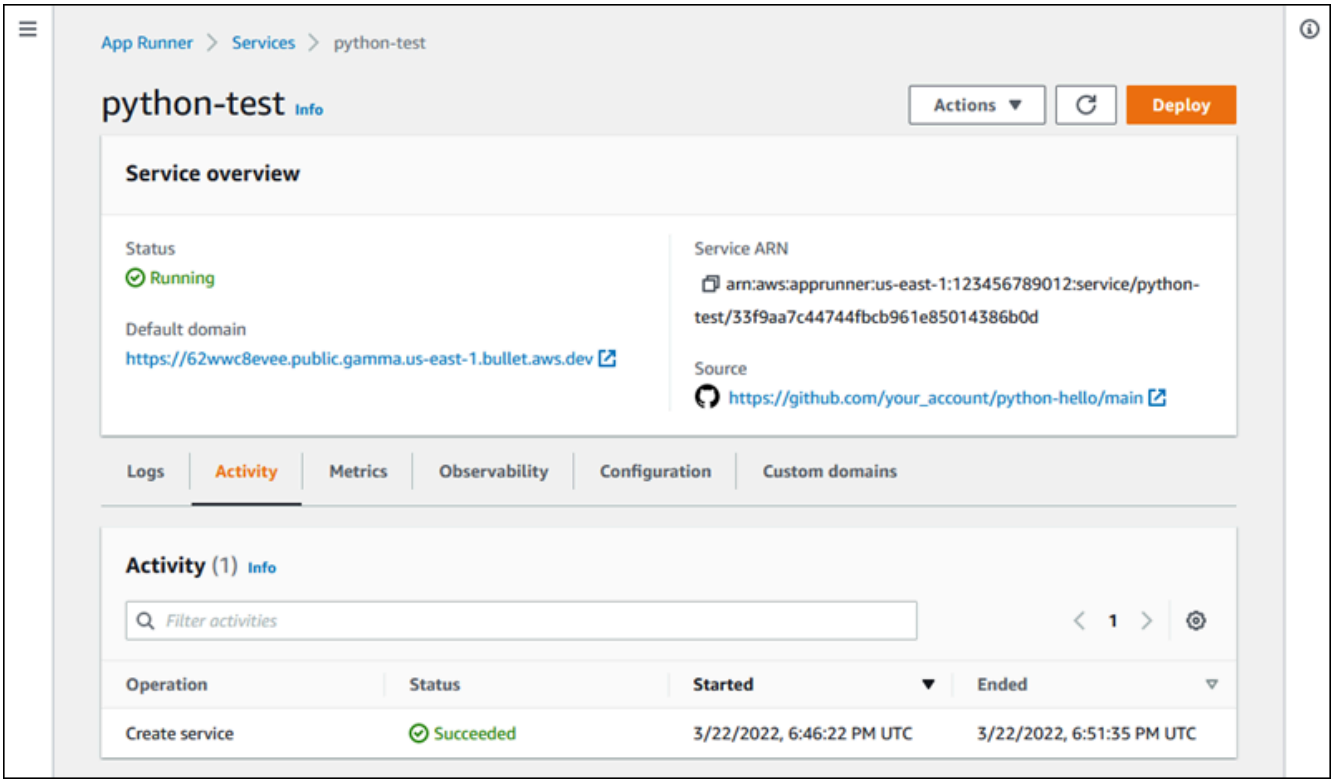
App Runner console

App Runner コンソールには App Runner サービスアクティビティが表示され、オペレーションを探索するさまざまな方法が用意されています。

サービスのアクティビティを表示するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. サービスダッシュボードページで、まだ選択されていない場合はアクティビティタブを選択します。

   コンソールにオペレーションのリストが表示されます。

4. 特定のオペレーションを検索するには、検索語を入力してリストの範囲を絞り込みます。テーブルに表示される任意の値を検索できます。
5. リストされたオペレーションを選択して、関連するログを表示またはダウンロードします。

App Runner API or AWS CLI

ListOperations アクションは、App Runner サービスの Amazon リソースネーム (ARN) を指定して、このサービスで発生したオペレーションのリストを返します。各リスト項目には、オペレーション ID といくつかの追跡の詳細が含まれています。

CloudWatch Logs にストリーミングされた App Runner ログの表示

Amazon CloudWatch Logs を使用して、さまざまな AWS サービスのリソースが生成するログファイルをモニタリング、保存、およびアクセスできます。詳細については、Amazon CloudWatch Logs ユーザーガイドを参照してください。

AWS App Runner は、アプリケーションデプロイとアクティブなサービスの出力を収集し、CloudWatch Logs にストリーミングします。以下のセクションでは、App Runner ログストリームを一覧表示し、App Runner コンソールでそれらを表示する方法を示します。

App Runner ロググループとストリーム

CloudWatch Logs はログデータをログストリームに保持し、さらにロググループに整理します。ログストリームは、特定のソースからの一連のログイベントです。ロググループは、保持、モニタリング、アクセス制御について同じ設定を共有するログストリームのグループです。

App Runner は、 内の App Runner サービスごとに、それぞれに複数のログストリームがある 2 つの CloudWatch Logs ロググループを定義します AWS アカウント。

サービスログ

サービスロググループには、App Runner サービスを管理し、そのサービスを操作する際に App Runner によって生成されたログ出力が含まれます。

ロググループ名	例
`/aws/apprunner/ service-name /service-id /service`	`/aws/apprunner/python-test/ac7ec8b51ff34746bcb6654e0bcb23da/service`

サービスロググループ内で、App Runner はイベントログストリームを作成して、App Runner サービスのライフサイクルのアクティビティをキャプチャします。例えば、アプリケーションの起動や一時停止などです。

さらに、App Runner は、サービスに関連する長時間実行される非同期オペレーションごとにログストリームを作成します。ログストリーム名には、オペレーションタイプと特定のオペレーション ID が反映されます。

デプロイはオペレーションの一種です。デプロイログには、サービスの作成時またはアプリケーションの新しいバージョンのデプロイ時に App Runner が実行するビルドおよびデプロイステップのログ出力が含まれます。デプロイログストリーム名は で始まりdeployment/、デプロイを実行するオペレーションの ID で終わります。このオペレーションは、最初のアプリケーションデプロイの場合は CreateService 呼び出し、以降のデプロイの場合は StartDeployment 呼び出しです。

デプロイログ内では、各ログメッセージはプレフィックスで始まります。

- [AppRunner] – デプロイ中に App Runner が生成する出力。
- [Build] – 独自のビルドスクリプトの出力。

ログストリーム名	例
events	該当なし (固定名)
operation-type /*operation-id*	deployment/c2c8eeedea164f45 9cf78f12a8953390

アプリケーションログ

アプリケーションロググループには、実行中のアプリケーションコードの出力が含まれます。

ロググループ名	例
/aws/apprunner/ *service-n* *ame* /*service-id* /application	/aws/apprunner/python-test/ ac7ec8b51ff34746bcb6654e0bcb23da/ application

アプリケーションロググループ内で、App Runner はアプリケーションを実行しているインスタンス (スケーリングユニット) ごとにログストリームを作成します。

ログストリーム名	例
instance/ *instance-id*	instance/1a80bc9134a84699b7 b3432ebeebb591

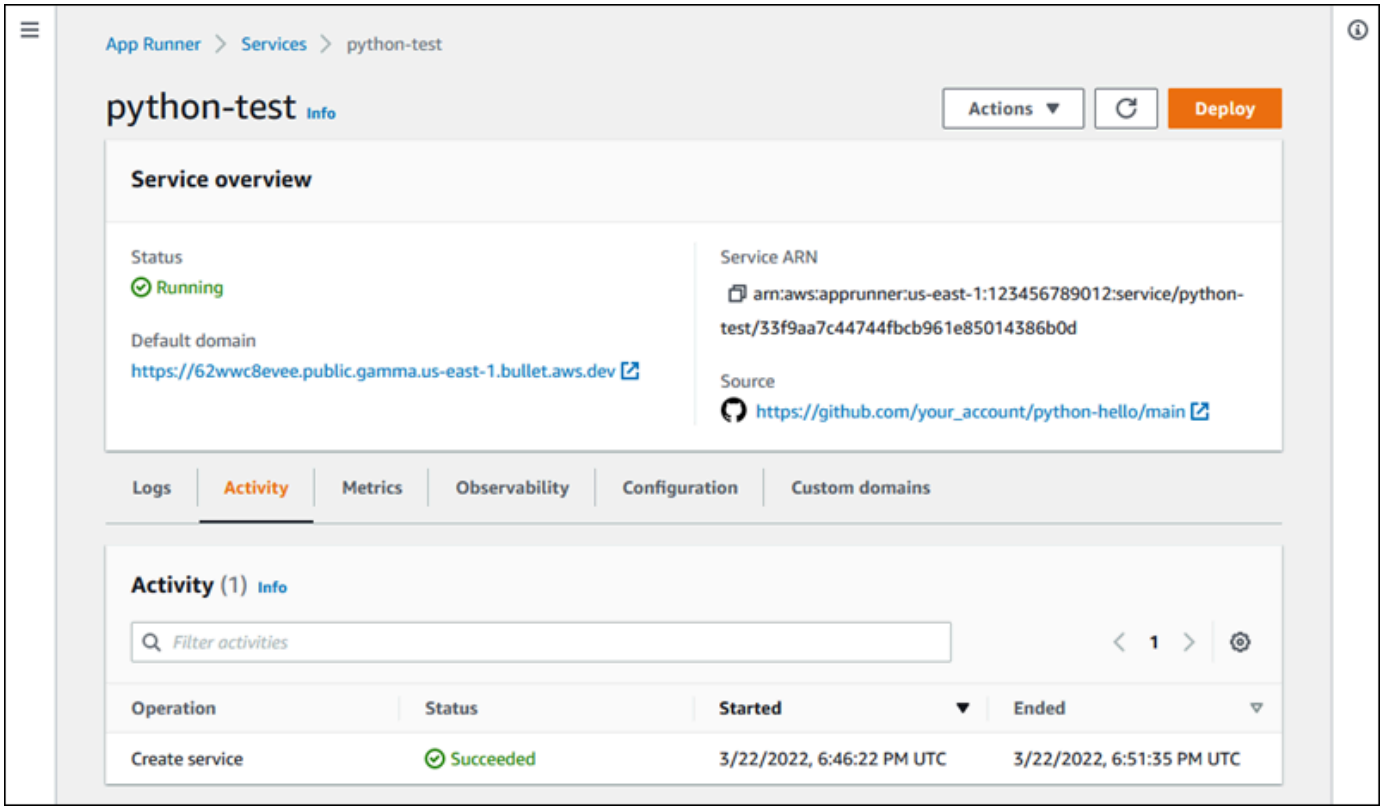
コンソールでの App Runner ログの表示

App Runner コンソールには、サービスのすべてのログの概要が表示され、それらを表示、探索、ダウンロードできます。

サービスのログを表示するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

   コンソールには、サービスダッシュボードにサービスの概要が表示されます。

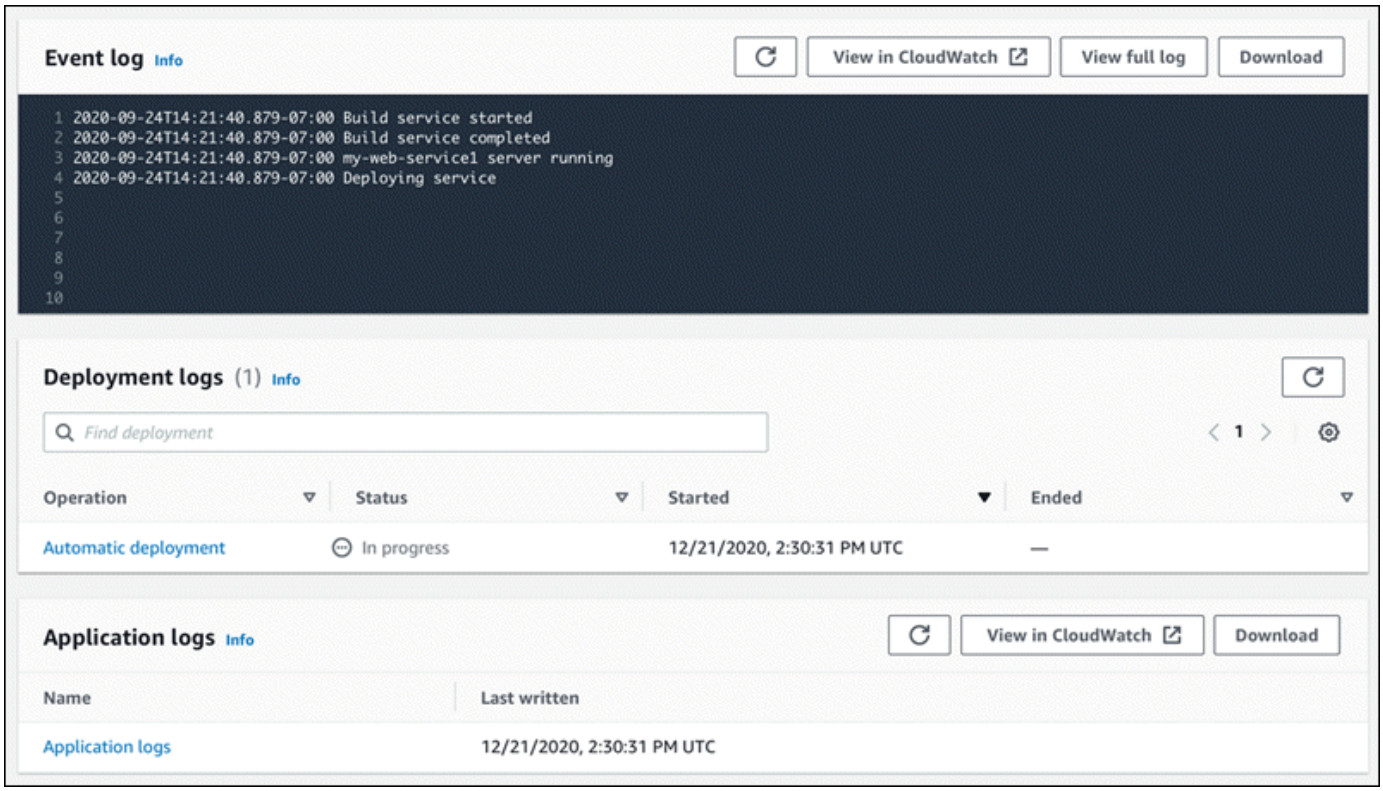


3. サービスダッシュボードページで、ログタブを選択します。

   コンソールには、いくつかのセクションにいくつかのタイプのログが表示されます。

   - イベントログ – App Runner サービスのライフサイクルにおけるアクティビティ。コンソールに最新のイベントが表示されます。
   - デプロイログ – App Runner サービスへのソースリポジトリのデプロイ。コンソールには、デプロイごとに個別のログストリームが表示されます。

- アプリケーションログ – App Runner サービスにデプロイされたウェブアプリケーションの出力。コンソールは、実行中のすべてのインスタンスからの出力を 1 つのログストリームに結合します。



4. 特定のデプロイを検索するには、検索語を入力してデプロイログリストの範囲を絞り込みます。テーブルに表示される任意の値を検索できます。
5. ログの内容を表示するには、フルログの表示 (イベントログ) またはログストリーム名 (デプロイログとアプリケーションログ) を選択します。
6. ダウンロードを選択してログをダウンロードします。デプロイログストリームの場合は、まずログストリームを選択します。
7. CloudWatch で表示 を選択して CloudWatch コンソールを開き、その完全な機能を使用して App Runner サービスログを調べます。デプロイログストリームの場合は、まずログストリームを選択します。

> **Note**
>
> CloudWatch コンソールは、結合されたアプリケーションログではなく、特定のインスタンスのアプリケーションログを表示する場合に特に便利です。

CloudWatch にレポートされた App Runner サービスメトリクスの表示

Amazon CloudWatch は、Amazon Web Services (AWS) リソースと、 で実行されているアプリケーションを AWS リアルタイムでモニタリングします。CloudWatch を使用してメトリクスを収集および追跡できます。メトリクスとは、リソースやアプリケーションについて測定できる変数です。これを使用して、メトリクスを監視するアラームを作成することもできます。特定のしきい値に達すると、CloudWatch は通知を送信するか、モニタリング対象のリソースに自動的に変更を加えます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

AWS App Runner は、App Runner サービスの使用、パフォーマンス、可用性をより詳細に可視化するさまざまなメトリクスを収集します。一部のメトリクスはウェブサービスを実行する個々のインスタンスを追跡しますが、他のメトリクスはサービスレベル全体です。以下のセクションでは、App Runner メトリクスを一覧表示し、App Runner コンソールでそれらを表示する方法を示します。

App Runner メトリクス

App Runner は、サービスに関連する以下のメトリクスを収集し、 `AWS/AppRunner`名前空間の CloudWatch に公開します。

> **Note**
>
> 2023 年 8 月 23 日より前は、CPU 使用率とメモリ使用率のメトリクスは、現在計算されている使用率ではなく、vCPU ユニットと使用されたメモリのメガバイトに基づいていました。アプリケーションがこの日付より前に App Runner で実行され、App Runner または CloudWatch コンソールでこの日付のメトリクスの表示に戻ることを選択した場合、両方のユニットにメトリクスが表示され、結果としていくつかの異常も表示されます。

> **Important**
>
> CPU 使用率とメモリ使用率のメトリクス値に基づく CloudWatch アラームは、2023 年 8 月 23 日より前に更新する必要があります。vCPU やメガバイトではなく使用率に基づいてトリガーするようにアラームを更新します。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

インスタンスレベルのメトリクスは、インスタンス (スケーリングユニット) ごとに個別に収集されます。

何を測定しますか?	メトリクス	説明
CPU utilization	`CPUUtilization`	サービス設定によって予約された合計 CPU 使用率のうち、1 分間の平均 CPU 使用率の割合。
Memory utilization	`MemoryUtilization`	サービス設定によって予約された合計メモリのうち、1 分間の平均メモリ使用量の割合。

サービスレベルのメトリクスは、サービス全体で収集されます。

何を測定しますか?	メトリクス	説明
CPU utilization	`CPUUtilization`	サービス設定によって予約された合計 CPU 使用率のうち、1 分間にすべてのインスタンスで集計された CPU 使用率の割合。
Memory utilization	`MemoryUtilization`	サービス設定によって予約された合計メモリのうち、1 分間のすべてのインスタンスでの合計メモリ使用量の割合。
Concurrency	`Concurrency`	サービスによって処理される同時リクエストのおおよその数。
HTTP request count	`Requests`	サービスが受信した HTTP リクエストの数。

HTTP status counts	`2xxStatusResponses`
	
	`4xxStatusResponses`
		カテゴリ (2XX、4XX, 5XX) 別にグループ化された各レスポンスステータスを返した HTTP リクエストの数。

何を測定しますか?	メトリクス	説明
	`5xxStatus Responses`	
HTTP request latency	`RequestLatency`	ウェブサービスが HTTP リクエストを処理するのにかかったミリ秒単位の時間。
Instance counts	`ActiveInstances`	サービスの HTTP リクエストを処理しているインスタンスの数。 **Note** `ActiveInstances` メトリクスにゼロが表示された場合は、サービスに対するリクエストがないことを意味します。サービスのインスタンス数がゼロであることを示すものではありません。

コンソールでの App Runner メトリクスの表示

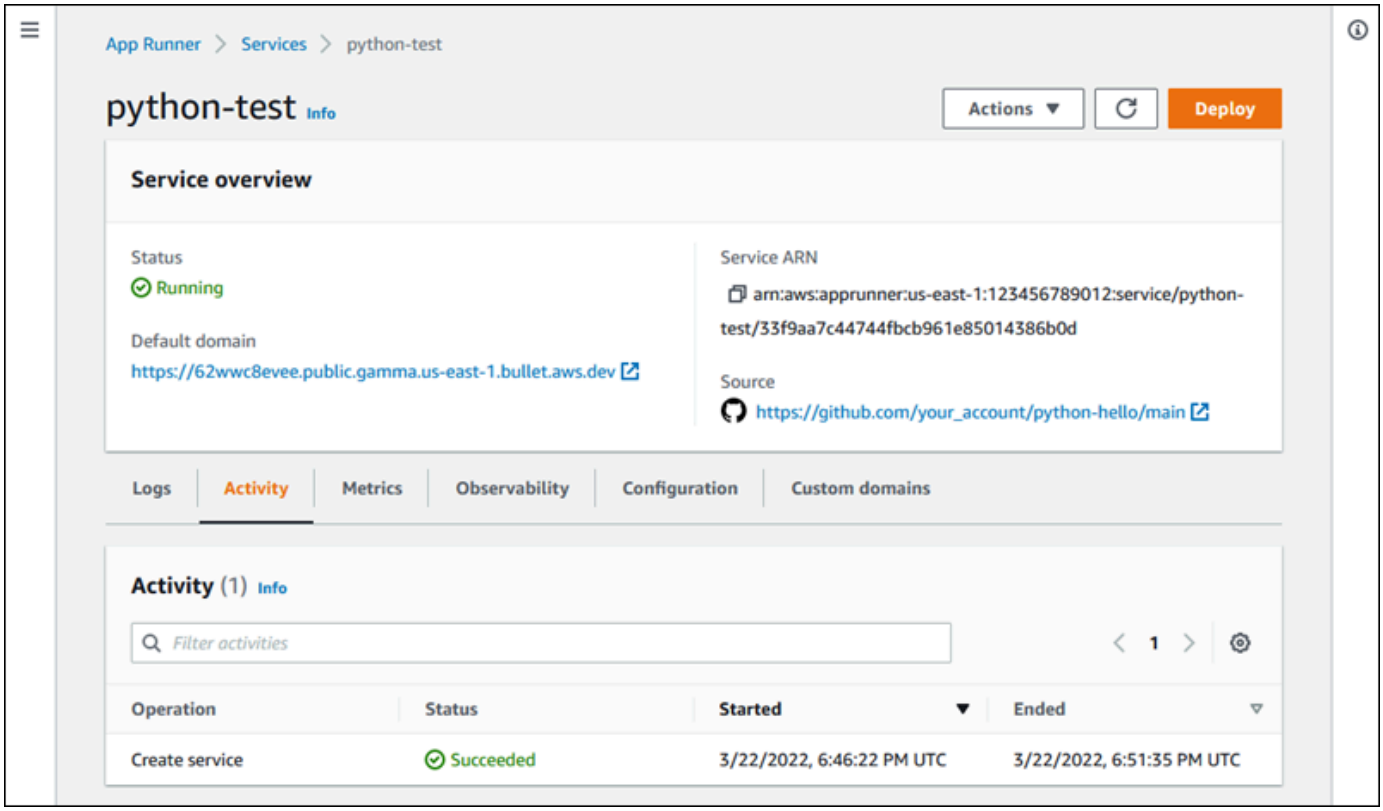
App Runner コンソールは、App Runner がサービス用に収集するメトリクスをグラフィカルに表示し、それらを探索するより多くの方法を提供します。

> **Note**
>
> 現時点では、コンソールにはサービスメトリクスのみが表示されます。インスタンスメトリクスを表示するには、CloudWatch コンソールを使用します。

サービスのログを表示するには

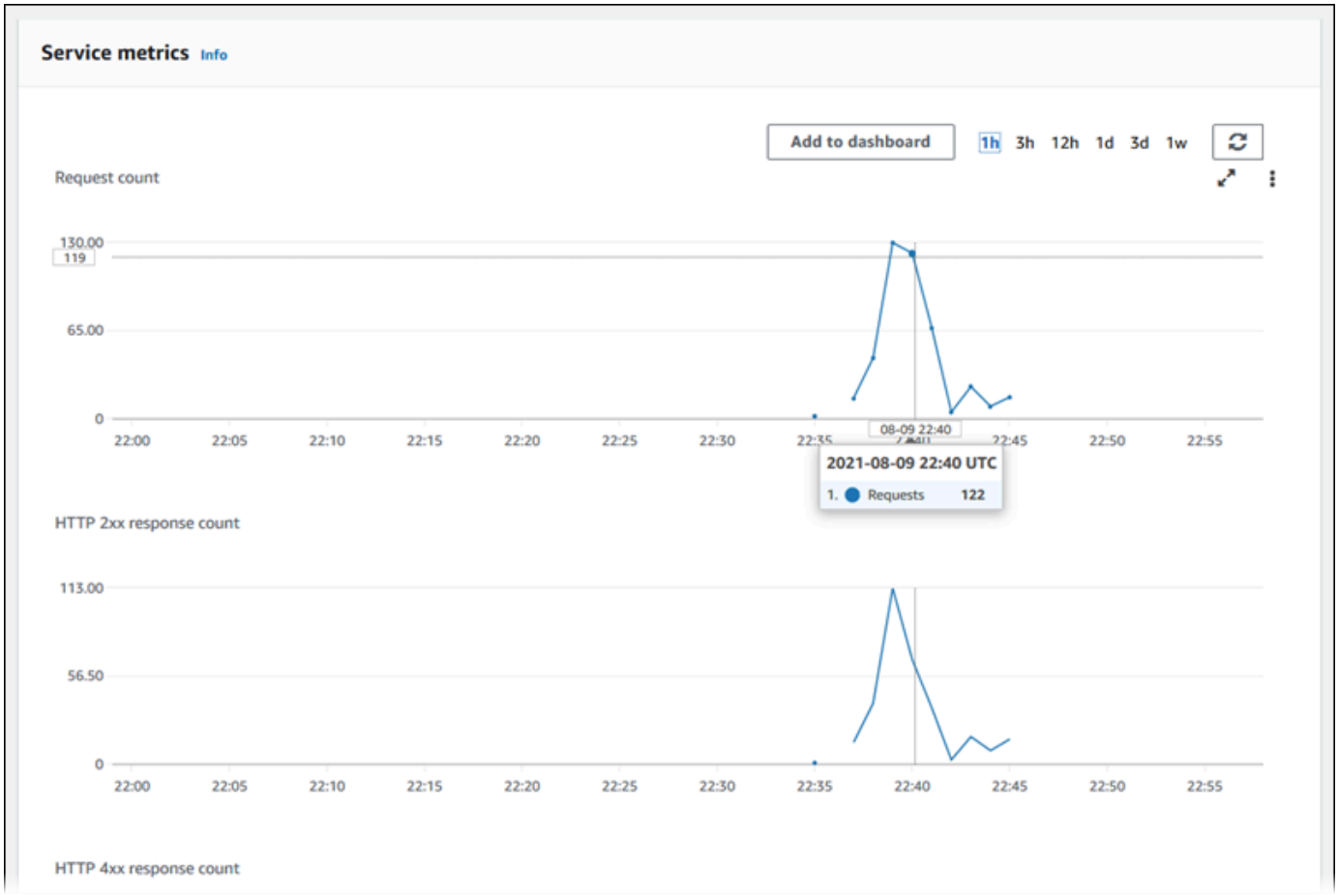
1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. ナビゲーションペインで、サービスを選択し、App Runner サービスを選択します。

   コンソールには、サービスダッシュボードにサービスの概要が表示されます。



3. サービスダッシュボードページで、メトリクスタブを選択します。

   コンソールには、一連のメトリクスグラフが表示されます。



4. 期間 (12 時間など) を選択して、メトリクスグラフをその期間の最近の期間にスコープします。
5. グラフセクションのいずれかの上部にあるダッシュボードに追加を選択するか、任意のグラフのメニューを使用して、CloudWatch コンソールのダッシュボードに関連メトリクスを追加して詳細な調査を行います。

EventBridge での App Runner イベントの処理

Amazon EventBridge を使用すると、特定のパターンについて AWS App Runner サービスからのリアルタイムデータのストリームをモニタリングするイベント駆動型ルールを設定できます。ルールのパターンが一致する AWS Batchと、EventBridge は Amazon ECS AWS Lambda、Amazon SNS などのターゲットでアクションを開始します。例えば、サービスへのデプロイが失敗するたびに Amazon SNS トピックをシグナリングすることで、E メール通知を送信するルールを設定できます。または、サービスの更新が失敗するたびに Slack チャネルに通知するように Lambda 関数を設定できます。EventBridge の詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。

App Runner が EventBridge に次のイベントタイプを送信する

- サービスステータスの変更 – App Runner サービスのステータスの変更。たとえば、サービスのステータスが に変更されましたDELETE_FAILED。
- サービスオペレーションステータスの変更 – App Runner サービスでの長い非同期オペレーションのステータスの変更。たとえば、サービスの作成が開始された、サービスの更新が正常に完了した、サービスデプロイがエラーで完了したなどです。

App Runner イベントを処理する EventBridge ルールの作成

EventBridge イベントは、ソース AWS サービスや詳細 (イベント) タイプ、イベントの詳細を含むイベント固有のフィールドセットなど、一部の標準 EventBridge フィールドを定義するオブジェクトです。EventBridge ルールを作成するには、EventBridge コンソールを使用してイベントパターン (追跡する必要があるイベント) を定義し、ターゲットアクション (一致に対して何をすべきか) を指定します。イベントパターンは、一致するイベントに似ています。一致するフィールドのサブセットを指定し、フィールドごとに可能な値のリストを指定します。このトピックでは、App Runner イベントとイベントパターンの例を示します。

EventBridge ルールの作成の詳細については、「Amazon EventBridge ユーザーガイド」の AWS 「 サービスのルールの作成」を参照してください。 EventBridge

> **Note**
>
> 一部のサービスは、EventBridge で事前定義されたパターンをサポートしています。これは、イベントパターンが作成される方法を簡素化します。フォーム上のフィールド値を選択すると、EventBridge がパターンを生成します。現時点では、App Runner は事前定義されたパターンをサポートしていません。パターンを JSON オブジェクトとして入力する必要があります。このトピックの例は、開始点として使用できます。

App Runner イベントの例

以下は、App Runner が EventBridge に送信するイベントの例です。

- サービスステータス変更イベント。具体的には、 から RUNNINGステータスOPERATION_IN_PROGRESSに変更されたサービスです。

```
{
```

```
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "AppRunner Service Status Change",
  "source": "aws.apprunner",
  "account": "111122223333",
  "time": "2021-04-29T11:54:23Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:apprunner:us-east-2:123456789012:service/my-
app/8fe1e10304f84fd2b0df550fe98a71fa"
  ],
  "detail": {
    "previousServiceStatus": "OPERATION_IN_PROGRESS",
    "currentServiceStatus": "RUNNING",
    "serviceName": "my-app",
    "serviceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "message": "Service status is set to RUNNING.",
    "severity": "INFO"
  }
}
```

- オペレーションステータス変更イベント。具体的には、正常に完了した UpdateService オペレーションです。

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "AppRunner Service Operation Status Change",
  "source": "aws.apprunner",
  "account": "111122223333",
  "time": "2021-04-29T18:43:48Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:apprunner:us-east-2:123456789012:service/my-
app/8fe1e10304f84fd2b0df550fe98a71fa"
  ],
  "detail": {
    "operationStatus": "UpdateServiceCompletedSuccessfully",
    "serviceName": "my-app",
    "serviceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "message": "Service update completed successfully. New application and
 configuration is deployed.",
    "severity": "INFO"
```

```
  }
}
```

App Runner イベントパターンの例

次の例は、EventBridge ルールで 1 つ以上の App Runner イベントを照合するために使用できるイベントパターンを示しています。イベントパターンはイベントに似ています。一致させるフィールドのみを含め、それぞれにスカラーの代わりにリストを指定します。

- サービスがステータスでなくなった特定のアカウントのサービスのサービスRUNNINGステータス変更イベントをすべて一致させます。

```
{
  "detail-type": [ "AppRunner Service Status Change" ],
  "source": [ "aws.apprunner" ],
  "account": [ "111122223333" ],
  "detail": {
    "previousServiceStatus": [ "RUNNING" ]
  }
}
```

- オペレーションが失敗した特定のアカウントのサービスのオペレーションステータス変更イベントをすべて一致させます。

```
{
  "detail-type": [ "AppRunner Service Operation Status Change" ],
  "source": [ "aws.apprunner" ],
  "account": [ "111122223333" ],
  "detail": {
    "operationStatus": [
      "CreateServiceFailed",
      "DeleteServiceFailed",
      "UpdateServiceFailed",
      "DeploymentFailed",
      "PauseServiceFailed",
      "ResumeServiceFailed"
    ]
  }
}
```

App Runner イベントリファレンス

サービスステータスの変更

サービスステータス変更イベントが にdetail-type設定されていますAppRunner Service Status Change。次の詳細フィールドと値があります。

```
"serviceId": "your service ID",
"serviceName": "your service name",
"message": "Service status is set to CurrentStatus.",
"previousServiceStatus": "any valid service status",
"currentServiceStatus": "any valid service status",
"severity": "varies"
```

オペレーションステータスの変更

オペレーションステータスの変更イベントが にdetail-type設定されていますAppRunner Service Operation Status Change。次の詳細フィールドと値があります。

```
"operationStatus": "see following table",
"serviceName": "your service name",
"serviceId": "your service ID",
"message": "see following table",
"severity": "varies"
```

次の表に、使用可能なすべてのステータスコードと関連するメッセージを示します。

ステータス	メッセージ
CreateServiceStarted	サービスの作成が開始されました。
CreateServiceCompletedSuccessfully	サービスの作成が正常に完了しました。
CreateServiceFailed	サービスの作成に失敗しました。詳細については、「 サービスログ」を参照してください。
DeleteServiceStarted	サービスの削除が開始されました。

ステータス	メッセージ
DeleteServiceCompletedSuccessfully	サービスの削除が正常に完了しました。
DeleteServiceFailed	サービスの削除に失敗しました。
UpdateServiceStarted	
UpdateServiceCompletedSuccessfully	サービスの更新が正常に完了しました。新しいアプリケーションと設定がデプロイされます。
	サービスの更新が正常に完了しました。新しい設定がデプロイされます。
UpdateServiceFailed	サービスの更新に失敗しました。詳細については、「 サービスログ」を参照してください。
DeploymentStarted	デプロイが開始されました。
DeploymentCompletedSuccessfully	デプロイが正常に完了しました。
DeploymentFailed	デプロイに失敗しました。詳細については、「 サービスログ」を参照してください。
PauseServiceStarted	サービスの一時停止が開始されました。
PauseServiceCompletedSuccessfully	サービスの一時停止が正常に完了しました。
PauseServiceFailed	サービスの一時停止に失敗しました。
ResumeServiceStarted	サービス再開が開始されました。
ResumeServiceCompletedSuccessfully	サービス再開は正常に完了しました。
ResumeServiceFailed	サービス再開に失敗しました。

を使用した App Runner API コールのログ記録 AWS CloudTrail

App Runner は AWS CloudTrail、App Runner のユーザー、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています。CloudTrail は、App Runner のすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、App Runner コンソールからの呼び出しと App Runner API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、App Runner のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、App Runner に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail の App Runner 情報

CloudTrail は、アカウントの作成 AWS アカウント 時に で有効になります。App Runner でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。で最近のイベントを表示、検索、ダウンロードできます AWS アカウント。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

App Runner のイベントなど AWS アカウント、 のイベントの継続的な記録については、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベントデータをさらに分析して処理するように他の AWS サービスを設定できます。詳細については、次を参照してください:

- 証跡の作成のための概要
- CloudTrail がサポートするサービスと統合
- CloudTrail 用 Amazon SNS 通知の構成
- 複数のリージョンから CloudTrail ログファイルを受け取るおよび複数のアカウントから CloudTrail ログファイルを受け取る

すべての App Runner アクションは CloudTrail によってログに記録され、 AWS App Runner API リファレンスに記載されています。たとえ

ば、CreateService、DeleteConnection、StartDeployment の各アクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

App Runner ログファイルエントリについて

「トレイル」は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルには、ログエントリが 1 つ以上あります。イベントは、あらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、およびリクエストパラメータに関する情報が含まれています。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下の例は、CreateService アクションを示す CloudTrail ログエントリです。

> **Note**
>
> セキュリティ上の理由から、一部のプロパティ値はログで編集され、テキスト に置き換えられますHIDDEN_DUE_TO_SECURITY_REASONS。これにより、シークレット情報が意図せず公開されるのを防ぐことができます。ただし、これらのプロパティがリクエストで渡されたか、レスポンスで返されたことがわかります。

CreateService App Runner アクションの CloudTrail ログエントリの例

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/aws-user",
```

```
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "aws-user"
  },
  "eventTime": "2020-10-02T23:25:33Z",
  "eventSource": "apprunner.amazonaws.com",
  "eventName": "CreateService",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/77.0.3865.75 Safari/537.36",
  "requestParameters": {
    "serviceName": "python-test",
    "sourceConfiguration": {
      "codeRepository": {
        "repositoryUrl": "https://github.com/github-user/python-hello",
        "sourceCodeVersion": {
          "type": "BRANCH",
          "value": "main"
        },
        "codeConfiguration": {
          "configurationSource": "API",
          "codeConfigurationValues": {
            "runtime": "python3",
            "buildCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
            "startCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
            "port": "8080",
            "runtimeEnvironmentVariables": "HIDDEN_DUE_TO_SECURITY_REASONS"
          }
        }
      },
      "autoDeploymentsEnabled": true,
      "authenticationConfiguration": {
        "connectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/your-
connection/e7656250f67242d7819feade6800f59e"
      }
    },
    "healthCheckConfiguration": {
      "protocol": "HTTP"
    },
    "instanceConfiguration": {
      "cpu": "256",
      "memory": "1024"
    }
```

```
  },
  "responseElements": {
    "service": {
        "serviceName": "python-test",
        "serviceId": "dfa2b7cc7bcb4b6fa6c1f0f4efff988a",
        "serviceArn": "arn:aws:apprunner:us-east-2:123456789012:service/python-test/
dfa2b7cc7bcb4b6fa6c1f0f4efff988a",
        "serviceUrl": "generated domain",
        "createdAt": "2020-10-02T23:25:32.650Z",
        "updatedAt": "2020-10-02T23:25:32.650Z",
        "status": "OPERATION_IN_PROGRESS",
        "sourceConfiguration": {
            "codeRepository": {
                "repositoryUrl": "https://github.com/github-user/python-hello",
                "sourceCodeVersion": {
                    "type": "Branch",
                    "value": "main"
                },
                "sourceDirectory": "/",
                "codeConfiguration": {
                    "codeConfigurationValues": {
                        "configurationSource": "API",
                        "runtime": "python3",
                        "buildCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
                        "startCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
                        "port": "8080",
                        "runtimeEnvironmentVariables": "HIDDEN_DUE_TO_SECURITY_REASONS"
                    }
                }
            },
            "autoDeploymentsEnabled": true,
            "authenticationConfiguration": {
                "connectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/
your-connection/e7656250f67242d7819feade6800f59e"
            }
        },
        "healthCheckConfiguration": {
            "protocol": "HTTP",
            "path": "/",
            "interval": 5,
            "timeout": 2,
            "healthyThreshold": 3,
            "unhealthyThreshold": 5
        },
```

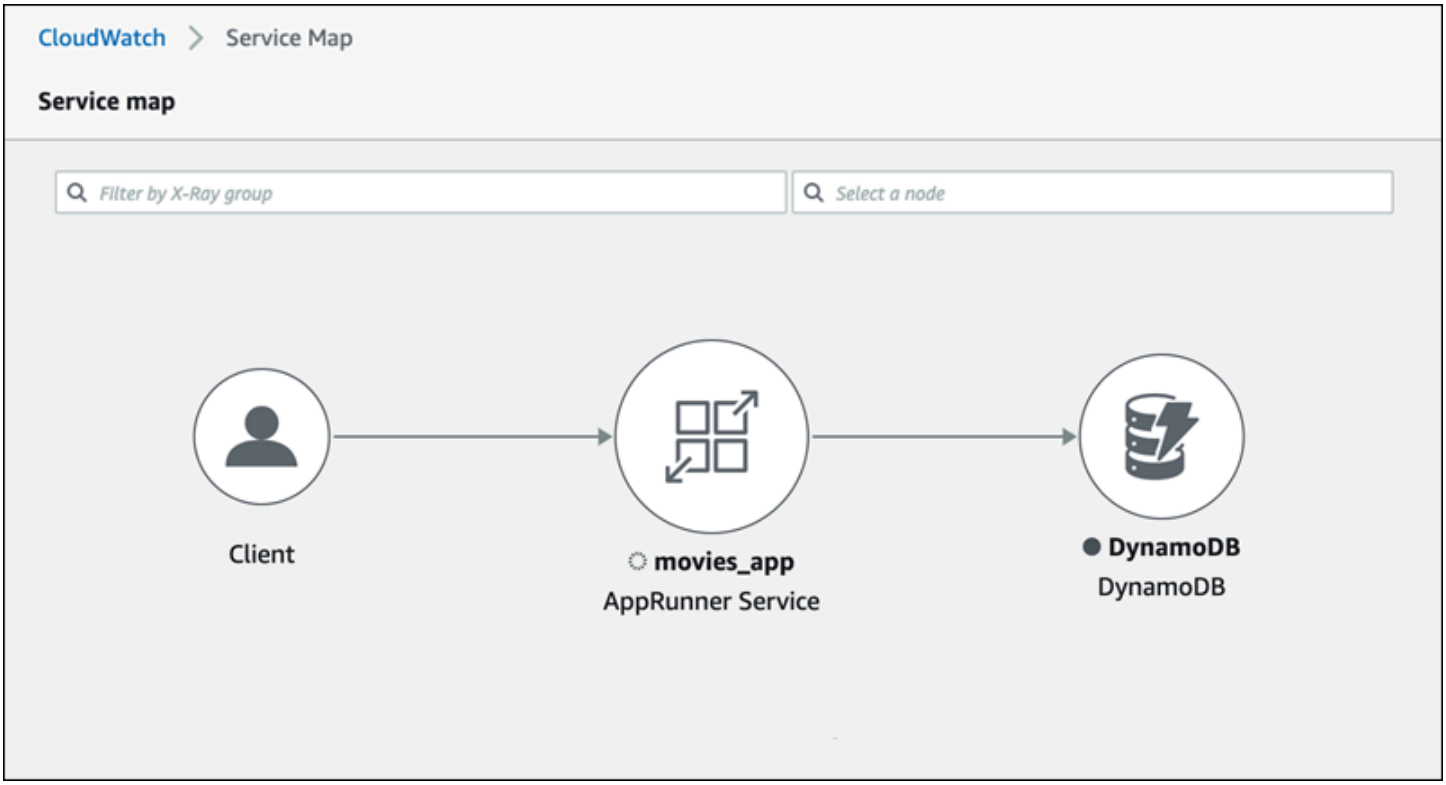
```
        "instanceConfiguration": {
            "cpu": "256",
            "memory": "1024"
        },
        "autoScalingConfigurationSummary": {
            "autoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
            "autoScalingConfigurationName": "DefaultConfiguration",
            "autoScalingConfigurationRevision": 1
        }
    }
},
  "requestID": "1a60af60-ecf5-4280-aa8f-64538319ba0a",
  "eventID": "e1a3f623-4d24-4390-a70b-bf08a0e24669",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

X-Ray を使用した App Runner アプリケーションのトレース

AWS X-Ray は、アプリケーションが処理するリクエストに関するデータを収集するサービスであり、そのデータを表示、フィルタリング、インサイトを取得して、問題や最適化の機会を特定するために使用できるツールを提供します。アプリケーションへのトレースされたリクエストについては、リクエストとレスポンスだけでなく、アプリケーションがダウンストリーム AWS リソース、マイクロサービス、データベース、HTTP ウェブ APIs に対して行う呼び出しに関する詳細情報も確認できます。

X-Ray は、クラウドアプリケーションを強化する AWS リソースからのトレースデータを使用して、詳細なサービスグラフを生成します。サービスグラフには、フロントエンドサービスが呼び出してリクエストを処理しデータを維持するクライアント、フロントエンドサービス、バックエンドサービスが表示されます。サービスグラフを使用して、ボトルネック、レイテンシーのスパイク、その他の問題を識別して解決し、アプリケーションのパフォーマンスを向上させます。

X-Ray の詳細については、「AWS X-Ray デベロッパーガイド」を参照してください。



トレース用にアプリケーションを計測する

ポータブルテレメトリ仕様である OpenTelemetry を使用して、App Runner サービスアプリケーションをトレース用に計測します。現時点では、App Runner は AWS Distro for OpenTelemetry (ADOT) をサポートしています。ADOT は、 AWS サービスを使用してテレメトリ情報を収集して表示する OpenTelemetry 実装です。X-Ray はトレースコンポーネントを実装します。

アプリケーションで使用している特定の ADOT SDK に応じて、ADOT は自動と手動の 2 つの計測アプローチをサポートします。SDK を使用した計測の詳細については、ADOT ドキュメントを参照し、ナビゲーションペインで SDK を選択します。

ランタイム設定

App Runner サービスアプリケーションをトレース用に計測するための一般的なランタイムセットアップ手順は次のとおりです。

ランタイムのトレースを設定するには

1. AWS Distro for OpenTelemetry (ADOT) のランタイムに用意されている手順に従って、アプリケーションを計測します。

2. ソースコードリポジトリを使用している場合は `apprunner.yaml`ファイルの `build`セクションに、コンテナイメージを使用している場合は Dockerfile に必要なOTEL依存関係をインストールします。
3. ソースコードリポジトリを使用している場合は `apprunner.yaml` ファイルで、コンテナイメージを使用している場合は Dockerfile で環境変数を設定します。

   Example環境変数

> **Note**
>
> 次の の例では、 `apprunner.yaml` ファイルに追加する重要な環境変数を一覧表示します。コンテナイメージを使用している場合は、これらの環境変数を Dockerfile に追加します。ただし、各ランタイムは独自の特異度を持つことができ、次のリストにさらに環境変数を追加する必要がある場合があります。ランタイム固有の手順とランタイム用にアプリケーションを設定する方法の例の詳細については、AWS 「 Distro for OpenTelemetry」を参照して、「開始方法」の「ランタイム」を参照してください。

```
env:
    - name: OTEL_PROPAGATORS
      value: xray
    - name: OTEL_METRICS_EXPORTER
      value: none
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://localhost:4317
    - name: OTEL_RESOURCE_ATTRIBUTES
      value: 'service.name=example_app'
```

> **Note**
>
> `OTEL_METRICS_EXPORTER=none` App Runner Otel コレクターはメトリクスのログ記録を受け入れないため、 は App Runner にとって重要な環境変数です。メトリクストレースのみを受け入れます。

ランタイムセットアップの例

次の例は、ADOT Python SDK を使用してアプリケーションを自動計測する方法を示しています。SDK は、Python コードを 1 行追加することなく、アプリケーションの Python フレームワークで使用される値を記述するテレメトリデータを含むスパンを自動的に生成します。2 つのソースファイルで数行のみを追加または変更する必要があります。

まず、次の例に示すように、いくつかの依存関係を追加します。

Example requirements.txt

```
opentelemetry-distro[otlp]>=0.24b0
opentelemetry-sdk-extension-aws~=2.0
opentelemetry-propagator-aws-xray~=1.0
```

次に、アプリケーションを計測します。これを行う方法は、ソースイメージまたはソースコードなどのサービスソースによって異なります。

Source image

サービスソースがイメージの場合、コンテナイメージの構築とイメージ内のアプリケーションの実行を制御する Dockerfile を直接計測できます。次の例は、Python アプリケーション用に実装された Dockerfile を示しています。計測の追加は太字で強調されます。

Example Dockerfile

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install python3.7 -y && curl -O https://bootstrap.pypa.io/get-pip.py &&
 python3 get-pip.py && yum update -y
COPY . /app
WORKDIR /app
RUN pip3 install -r requirements.txt
RUN opentelemetry-bootstrap --action=install
ENV OTEL_PYTHON_DISABLED_INSTRUMENTATIONS=urllib3
ENV OTEL_METRICS_EXPORTER=none
ENV OTEL_RESOURCE_ATTRIBUTES='service.name=example_app'
CMD OTEL_PROPAGATORS=xray OTEL_PYTHON_ID_GENERATOR=xray opentelemetry-instrument
 python3 app.py
EXPOSE 8080
```

Source code repository

サービスソースがアプリケーションソースを含むリポジトリである場合は、App Runner 設定ファイル設定を使用してイメージを間接的に計測します。これらの設定は、App Runner が生成してアプリケーションのイメージを構築するために使用する Dockerfile を制御します。次の例は、Python アプリケーションの計測された App Runner 設定ファイルを示しています。計測の追加は太字で強調されます。

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install -r requirements.txt
      - opentelemetry-bootstrap --action=install
run:
  command: opentelemetry-instrument python app.py
  network:
    port: 8080
  env:
    - name: OTEL_PROPAGATORS
      value: xray
    - name: OTEL_METRICS_EXPORTER
      value: none
    - name: OTEL_PYTHON_ID_GENERATOR
      value: xray
    - name: OTEL_PYTHON_DISABLED_INSTRUMENTATIONS
      value: urllib3
    - name: OTEL_RESOURCE_ATTRIBUTES
      value: 'service.name=example_app'
```

App Runner サービスインスタンスロールに X-Ray アクセス許可を追加する

App Runner サービスで X-Ray トレースを使用するには、サービスのインスタンスに X-Ray サービスとやり取りするためのアクセス許可を付与する必要があります。これを行うには、インスタンスロールをサービスに関連付け、X-Ray アクセス許可を持つ管理ポリシーを追加します。App Runner インスタンスロールの詳細については、「」を参照してくださいthe section called “インスタンス

ロール”。AWSXRayDaemonWriteAccess 管理ポリシーをインスタンスロールに追加し、作成時にサービスに割り当てます。

App Runner サービスの X-Ray トレースを有効にする

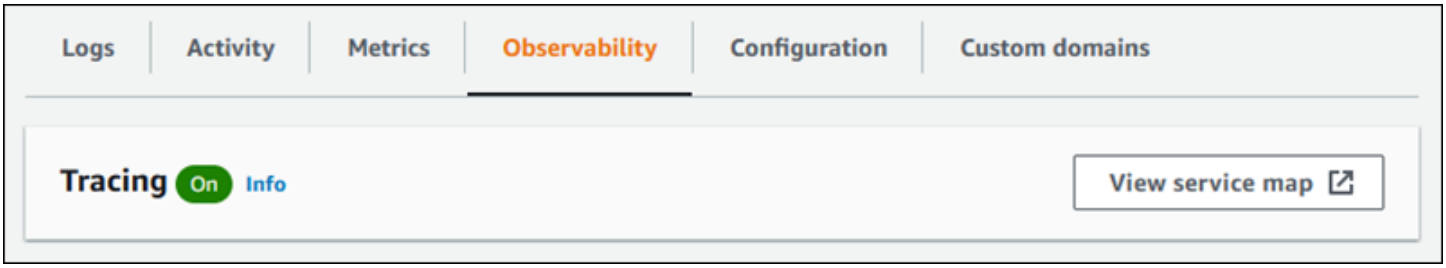
サービスを作成すると、App Runner はデフォルトでトレースを無効にします。オブザーバビリティの設定の一環として、サービスの X-Ray トレースを有効にできます。詳細については、「the section called “オブザーバビリティの管理”」を参照してください。

App Runner API または を使用する場合 AWS CLI、ObservabilityConfiguration リソースオブジェクト内の TraceConfiguration オブジェクトにはトレース設定が含まれます。ObservabilityConfiguration トレースを無効にしたままにするには、 TraceConfiguration オブジェクトを指定しないでください。

コンソールと API の両方のケースで、前のセクションで説明したインスタンスロールを App Runner サービスと関連付けてください。

App Runner サービスの X-Ray トレースデータを表示する

App Runner コンソールのサービスダッシュボードページのオブザーバビリティタブで、サービスマップを表示を選択して Amazon CloudWatch コンソールに移動します。



Amazon CloudWatch コンソールを使用して、アプリケーションが処理するリクエストのサービスマップとトレースを表示します。サービスマップには、リクエストのレイテンシーや、他のアプリケーションや AWS サービスとのやり取りなどの情報が表示されます。コードに追加するカスタム注釈を使用すると、トレースを簡単に検索できます。詳細については、Amazon CloudWatch ユーザーガイド」のServiceLens を使用してアプリケーションの状態をモニタリングする」を参照してください。

AWS WAF ウェブ ACL をサービスに関連付ける

AWS WAF は、App Runner サービスを保護するために使用できるウェブアプリケーションファイアウォールです。 AWS WAF ウェブアクセスコントロールリスト (ウェブ ACLs) を使用すると、一般的なウェブエクスプロイトや不要なボットから App Runner サービスエンドポイントを保護できます。

ウェブ ACL を使用すると、App Runner サービスへのすべての受信ウェブリクエストをきめ細かく制御できます。ウェブ ACL でルールを定義して、ウェブトラフィックを許可、ブロック、またはモニタリングし、承認された正当なリクエストのみがウェブアプリケーションと APIs に到達するようにすることができます。特定のビジネスニーズとセキュリティニーズに基づいて、ウェブ ACL ルールをカスタマイズできます。ネットワーク ACLs「ネットワークトラフィックの制御」を参照してください。

> **Important**
>
> WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。

受信ウェブリクエストフロー

AWS WAF ウェブ ACL が App Runner サービスに関連付けられている場合、受信ウェブリクエストは次のプロセスを実行します。

1. App Runner はオリジンリクエストの内容を に転送します AWS WAF。
2. AWS WAF はリクエストを検査し、その内容をウェブ ACL で指定したルールと比較します。
3. その検査に基づいて、 は App Runner に `allow`または `block`レスポンス AWS WAF を返します。
   - `allow` レスポンスが返された場合、App Runner はリクエストをアプリケーションに転送します。

- `block` レスポンスが返されると、App Runner はリクエストがウェブアプリケーションに到達するのをブロックします。`block` レスポンスを からアプリケーションに転送 AWS WAF します。

> **Note**
>
> デフォルトでは、応答が返されない場合、App Runner はリクエストをブロックします AWS WAF。

AWS WAF ウェブ ACLs「ウェブアクセスコントロールリスト (ウェブ ACLs」を参照してください。 AWS WAF

> **Note**
>
> 標準 AWS WAF 料金が発生します。App Runner サービスに AWS WAF ウェブ ACLs を使用しても、追加料金は発生しません。
> 料金の詳細については、AWS WAF 「料金表」を参照してください。

WAF ウェブ ACLs App Runner サービスに関連付ける

以下は、 AWS WAF ウェブ ACL を App Runner サービスに関連付ける大まかなプロセスです。

1. AWS WAF コンソールでウェブ ACL を作成します。詳細については、「 デベロッパーガイド」の「ウェブ ACL の作成」を参照してください。 AWS WAF
2. の AWS Identity and Access Management (IAM) アクセス許可を更新します AWS WAF。詳細については、アクセス許可を参照してください。
3. 次のいずれかの方法を使用して、ウェブ ACL を App Runner サービスに関連付けます。
   - App Runner コンソール: App Runner サービスを作成または更新するときに、App Runner コンソールを使用して既存のウェブ ACL を関連付けます。手順については、AWS WAF 「ウェブ ACLs」を参照してください。
   - AWS WAF コンソール: 既存の App Runner サービスの AWS WAF コンソールを使用してウェブ ACL を関連付けます。詳細については、「AWS WAF デベロッパーガイド」の「ウェブ ACL と AWS リソースの関連付けまたは関連付け解除」を参照してください。

- AWS CLI: AWS WAF パブリック APIs。 AWS WAF パブリック APIsAssociateWebACL」を参照してください。 AWS WAF

考慮事項

- WAF ウェブ ACLs に関連付けられている App Runner プライベートサービスのソース IP ルールは、IP ベースのルールに準拠していません。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。
- App Runner サービスは、1 つのウェブ ACL にのみ関連付けることができます。ただし、1 つのウェブ ACL を複数の App Runner サービスおよび複数の AWS リソースに関連付けることができます。例としては、Amazon Cognito ユーザープールや Application Load Balancer リソースなどがあります。
- ウェブ ACL を作成すると、ウェブ ACL が完全に伝播されて App Runner で使用できるようになるまでに、わずかな時間がかかります。伝播時間は数秒から数分までです。 は、完全に伝播される前にウェブ ACL を関連付けようとWAFUnavailableEntityExceptionすると、 AWS WAF を返します。

  ウェブ ACL が完全に伝播される前にブラウザを更新したり、App Runner コンソールから移動したりすると、関連付けは実行されません。ただし、App Runner コンソール内を移動することはできます。
- AWS WAF は、無効な状態にある App Runner サービスに対して次のいずれかの AWS WAF APIs を呼び出すと、WAFNonexistentItemException工ラーを返します。
  - AssociateWebACL
  - DisassociateWebACL
  - GetWebACLForResource

  App Runner サービスの無効な状態は次のとおりです。
  - CREATE_FAILED
  - DELETE_FAILED
  - DELETED
  - OPERATION_IN_PROGRESS

> **Note**
>
> OPERATION_IN_PROGRESS 状態は、App Runner サービスが削除されている場合にのみ無効になります。

- リクエストにより、 が検査 AWS WAF できる の制限を超えるペイロードが発生する可能性があります。が App Runner からのオーバーサイズリクエスト AWS WAF を処理する方法の詳細については、 AWS WAF デベロッパーガイドの「オーバーサイズリクエストコンポーネントの処理」を参照して、 が App Runner からのオーバーサイズリクエスト AWS WAF を処理する方法を確認してください。
- 適切なルールを設定しない場合やトラフィックパターンが変更された場合、ウェブ ACL はアプリケーションの保護にそれほど効果的ではない可能性があります。

アクセス許可

でウェブ ACL を使用するには AWS App Runner、次の IAM アクセス許可を追加します AWS WAF。

- `apprunner:ListAssociatedServicesForWebAcl`
- `apprunner:DescribeWebAclForService`
- `apprunner:AssociateWebAcl`
- `apprunner:DisassociateWebAcl`

IAM アクセス許可の詳細については、IAM ユーザーガイドの「IAM のポリシーとアクセス許可」を参照してください。

以下は、 の更新された IAM ポリシーの例です AWS WAF。この IAM ポリシーには、App Runner サービスを操作するために必要なアクセス許可が含まれています。

Example

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "wafv2:ListResourcesForWebACL",
            "wafv2:GetWebACLForResource",
            "wafv2:AssociateWebACL",
            "wafv2:DisassociateWebACL",
            "apprunner:ListAssociatedServicesForWebAcl",
            "apprunner:DescribeWebAclForService",
            "apprunner:AssociateWebAcl",
            "apprunner:DisassociateWebAcl"
         ],
         "Resource":"*"
      }
   ]
}
```

> **Note**
>
> IAM アクセス権限を付与する必要がありますが、リストされているアクションは権限のみで、API オペレーションには対応していません。

AWS WAF ウェブ ACLs の管理

次のいずれかの方法を使用して、App Runner サービスの AWS WAF ウェブ ACLs を管理します。

- the section called “App Runner コンソール”
- the section called “AWS CLI”

App Runner コンソール

App Runner コンソールでサービスを作成したり、既存のサービスを更新したりすると、 AWS WAF ウェブ ACL の関連付けまたは関連付け解除ができます。

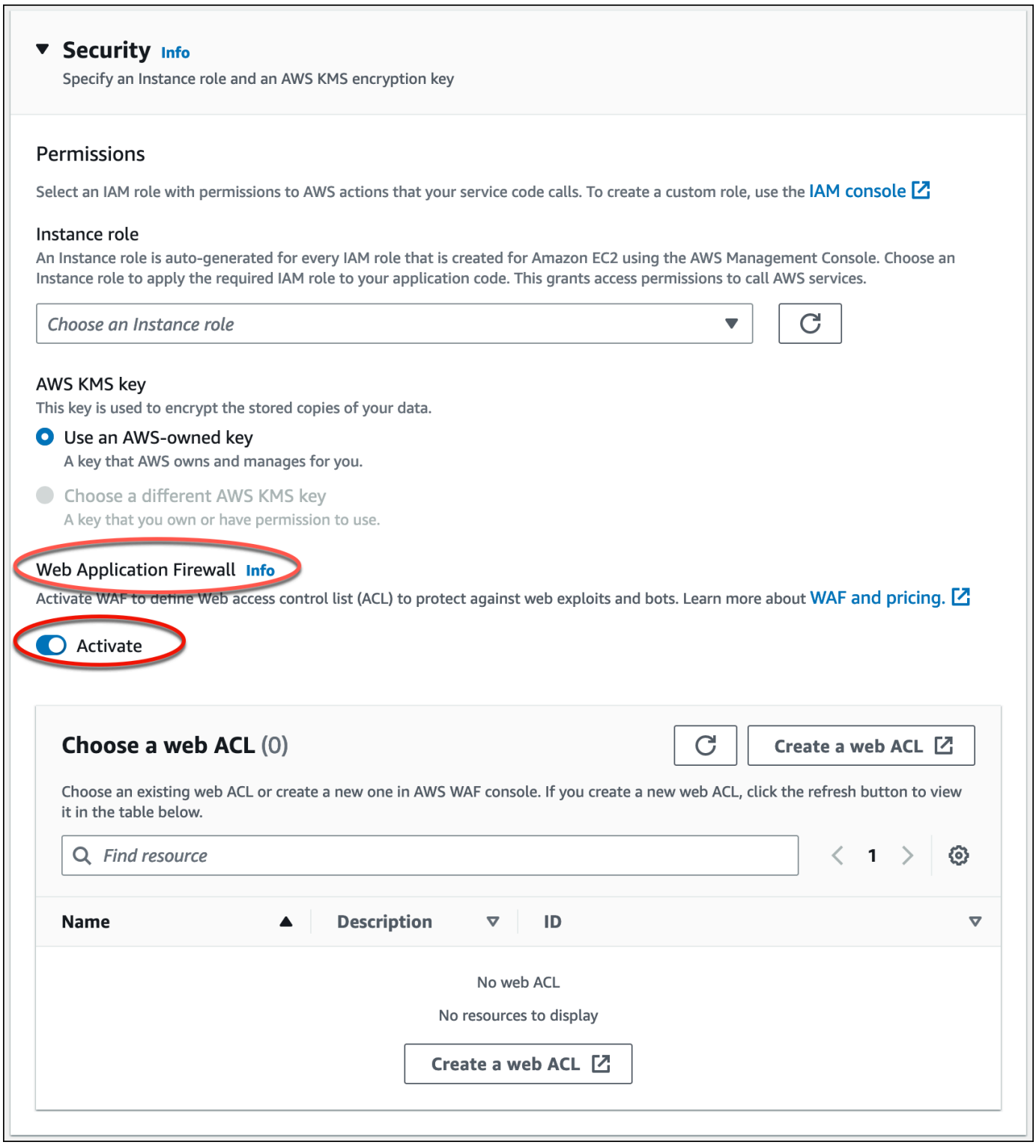
> **Note**
>
> - App Runner サービスは、1 つのウェブ ACL にのみ関連付けることができます。ただし、他の AWS リソースに加えて、1 つのウェブ ACL を複数の App Runner サービスに関連付けることができます。
> - ウェブ ACL を関連付ける前に、必ず IAM アクセス許可を更新してください AWS WAF。詳細については、「アクセス許可」を参照してください。

AWS WAF ウェブ ACL の関連付け

> **Important**
>
> WAF ウェブ ACLsルールは、IP ベースのルールに準拠していません。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。

AWS WAF ウェブ ACL を関連付けるには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. サービスを作成または更新するかどうかに基づいて、次のいずれかのステップを実行します。
   - 新しいサービスを作成する場合は、App Runner サービスの作成を選択し、サービスの設定に移動します。
   - 既存のサービスを更新する場合は、設定タブを選択し、サービスの設定で編集を選択します。
3. セキュリティでウェブアプリケーションファイアウォールに移動します。
4. トグルを有効にするボタンを選択すると、オプションが表示されます。



5. 次のいずれかのステップを実行します。

   - 既存のウェブ ACL を関連付けるには: App Runner サービスに関連付けるウェブ ACL の選択 テーブルから必要なウェブ ACL を選択します。

- 新しいウェブ ACL を作成するには: AWS WAF コンソールを使用して新しいウェブ ACL を作成するには、ウェブ ACL の作成を選択します。詳細については、「 デベロッパーガイド」の「ウェブ ACL の作成」を参照してください。 AWS WAF
  1. 更新ボタンを選択すると、新しく作成されたウェブ ACL がウェブ ACL の選択テーブルに表示されます。
  2. 必要なウェブ ACL を選択します。

6. 新しいサービスを作成する場合は次へ、既存のサービスを更新する場合は変更を保存するを選択します。選択したウェブ ACL は App Runner サービスに関連付けられています。
7. ウェブ ACL の関連付けを確認するには、サービスの設定タブを選択し、サービスの設定に移動します。Security の下にあるウェブアプリケーションファイアウォールにスクロールして、サービスに関連付けられたウェブ ACL の詳細を表示します。

> **Note**
>
> ウェブ ACL を作成すると、ウェブ ACL が完全に伝播され、App Runner で使用できるようになるまでに少し時間がかかります。伝達時間は数秒から数分までです。ウェブ ACL が完全に伝達される前に関連付けようとWAFUnavailableEntityExceptionすると、 は AWS WAF を返します。
> ウェブ ACL が完全に伝播される前にブラウザを更新したり、App Runner コンソールから移動したりすると、関連付けは失敗します。ただし、App Runner コンソール内を移動することはできます。

AWS WAF ウェブ ACL の関連付けを解除する

App Runner サービスを更新することで、不要になった AWS WAF ウェブ ACI の関連付けを解除できます。

AWS WAF ウェブ ACI の関連付けを解除するには

1. App Runner コンソールを開き、リージョンリストで を選択します AWS リージョン。
2. 更新するサービスの設定タブに移動し、サービスの設定で編集を選択します。
3. セキュリティでウェブアプリケーションファイアウォールに移動します。
4. トグルの有効化ボタンを無効にします。削除を確認するメッセージが表示されます。
5. [確認] を選択してください。ウェブ ACL は App Runner サービスとの関連付けが解除されます。

> **Note**
>
> - サービスを別のウェブ ACL に関連付ける場合は、ウェブ ACL の選択 テーブルからウェブ ACL を選択します。App Runner は、現在のウェブ ACL の関連付けを解除し、選択したウェブ ACL に関連付けるプロセスを開始します。
> - 関連付け解除されたウェブ ACL を使用する App Runner サービスまたはリソースが他にない場合は、ウェブ ACL の削除を検討してください。それ以外の場合は、引き続きコストが発生します。料金の詳細については、「AWS WAF 料金」を参照してください。ウェブ ACL を削除する方法については、 AWS WAF API リファレンスのDeleteWebACL」を参照してください。
> - 他のアクティブな App Runner サービスや他のリソースに関連付けられているウェブ ACL は削除できません。

AWS CLI

AWS WAF パブリック APIs を使用して、 AWS WAF ウェブ ACL の関連付けまたは関連付け解除を行うことができます。ウェブ ACL の関連付けまたは関連付け解除を行う App Runner サービスは、有効な状態である必要があります。

AWS WAF は、無効な状態の App Runner サービスに対して次のいずれか AWS WAF APIs を呼び出すと、`WAFNonexistentItemException`エラーを返します。

- `AssociateWebACL`
- `DisassociateWebACL`
- `GetWebACLForResource`

App Runner サービスの無効な状態は次のとおりです。

- `CREATE_FAILED`
- `DELETE_FAILED`
- `DELETED`
- `OPERATION_IN_PROGRESS`

> **Note**
>
> OPERATION_IN_PROGRESS 状態は、App Runner サービスが削除されている場合にのみ無効になります。

AWS WAF パブリック APIsAWS WAF 「 API リファレンスガイド」を参照してください。

> **Note**
>
> の IAM アクセス許可を更新します AWS WAF。詳細については、「 アクセス許可」を参照してください。

を使用した AWS WAF ウェブ ACL の関連付け AWS CLI

> **Important**
>
> WAF ウェブ ACLsルールは、IP ベースのルールに準拠していません。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送をサポートしていないためです。App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。

AWS WAF ウェブ ACL を関連付けるには

1. への優先ルールアクションのセットAllowまたはサービスへの AWS WAF ウェブリクエストを使用して、サービスの Block ウェブ ACL を作成します。 AWS WAF APIs「 API AWS WAF リファレンスガイド」の「CreateWebACL」を参照してください。

   Exampleウェブ ACL の作成 - リクエスト

   ```
   aws wafv2
   create-web-acl
   --region <region>
   --name <web-acl-name>
   --scope REGIONAL
   ```

```
--default-action Allow={}
--visibility-config <file-name.json>
# This is the file containing the WAF web ACL rules.
```

2. `associate-web-acl` AWS WAF パブリック API を使用して、作成したウェブ ACL を App Runner サービスに関連付けます。 AWS WAF APIs「 API AWS WAF リファレンスガイド」のAssociateWebACL」を参照してください。

> **Note**
>
> ウェブ ACL を作成すると、ウェブ ACL が完全に伝播され、App Runner で使用できるようになるまでに少し時間がかかります。伝達時間は数秒から数分までです。ウェブ ACL が完全に伝達される前に関連付けようと`WAFUnavailableEntityException`すると、 は AWS WAF を返します。
> ウェブ ACL が完全に伝播される前にブラウザを更新したり、App Runner コンソールから移動したりすると、関連付けは失敗します。ただし、App Runner コンソール内を移動することはできます。

Exampleウェブ ACL の関連付け - リクエスト

```
aws wafv2 associate-web-acl
--resource-arn <apprunner_service_arn>
--web-acl-arn <web_acl_arn>
--region <region>
```

3. `get-web-acl-for-resource` AWS WAF パブリック API を使用して、ウェブ ACL が App Runner サービスに関連付けられていることを確認します。 AWS WAF APIsAWS WAF 「 API リファレンスガイド」のGetWebACLForResource」を参照してください。

Exampleリソースのウェブ ACL の検証 - リクエスト

```
aws wafv2 get-web-acl-for-resource
--resource-arn <apprunner_service_arn>
--region <region>
```

サービスに関連付けられたウェブ ACLs がない場合は、空白のレスポンスが表示されます。

を使用した AWS WAF ウェブ ACL の削除 AWS CLI

App Runner サービスに関連付けられているウェブ AWS WAF ACL は削除できません。

AWS WAF ウェブ ACL を削除するには

1. `disassociate-web-acl` AWS WAF パブリック API を使用して、ウェブ ACL と App Runner サービスの関連付けを解除します。 AWS WAF APIs「 API AWS WAF リファレンスガイド」のDisassociateWebACL」を参照してください。

   Exampleウェブ ACL の関連付け解除 - リクエスト

   ```
   aws wafv2 disassociate-web-acl
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

2. `get-web-acl-for-resource` AWS WAF パブリック API を使用して、ウェブ ACL と App Runner サービスの関連付けが解除されていることを確認します。

   Exampleウェブ ACL の関連付けが解除されていることを確認する - リクエスト

   ```
   aws wafv2 get-web-acl-for-resource
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

   App Runner サービスに対して関連付けが解除されたウェブ ACL は表示されません。サービスに関連付けられたウェブ ACLs がない場合は、空白のレスポンスが表示されます。

3. `delete-web-acl` AWS WAF パブリック API を使用して、関連付けを解除したウェブ ACL を削除します。 AWS WAF APIs「 API AWS WAF リファレンスガイド」のDeleteWebACL」を参照してください。

   Exampleウェブ ACL の削除 - リクエスト

   ```
   aws wafv2 delete-web-acl
   --name <web_acl_name>
   --scope REGIONAL
   --id <web_acl_id>
   --lock-token <web_acl_lock_token>
   --region <region>
   ```

4. `list-web-acl` AWS WAF パブリック API を使用してウェブ ACL が削除されていることを確認します。 AWS WAF APIsAWS WAF 「 API リファレンスガイド」のListWebACLs」を参照してください。

   Exampleウェブ ACL が削除されたことを確認する - リクエスト

   ```
   aws wafv2 list-web-acls
   --scope REGIONAL
   --region <region>
   ```

   削除されたウェブ ACL は表示されなくなります。

   > **Note**
   >
   > ウェブ ACL が他のアクティブな App Runner サービスまたは Amazon Cognito ユーザープールなどの他のリソースに関連付けられている場合、ウェブ ACL は削除できません。

ウェブ ACL に関連付けられている App Runner サービスの一覧表示

ウェブ ACL は、複数の App Runner サービスやその他のリソースに関連付けることができます。`list-resources-for-web-acl` AWS WAF パブリック API を使用して、ウェブ ACL に関連付けられた App Runner サービスを一覧表示します。 AWS WAF APIsAWS WAF 「 API リファレンスガイド」のListResourcesForWebACL」を参照してください。

Exampleウェブ ACL に関連付けられた App Runner サービスを一覧表示する - リクエスト

```
aws wafv2 list-resources-for-web-acl
--web-acl-arn <WEB_ACL_ARN>
--resource-type APP_RUNNER_SERVICE
--region <REGION>
```

Exampleウェブ ACL に関連付けられた App Runner サービスを一覧表示する - レスポンス

次の例は、ウェブ ACL に関連付けられている App Runner サービスがない場合のレスポンスを示しています。

```
{
  "ResourceArns": []
```

```
}
```

Exampleウェブ ACL に関連付けられた App Runner サービスを一覧表示する - レスポンス

次の例は、ウェブ ACL に関連付けられている App Runner サービスがある場合のレスポンスを示しています。

```
{
  "ResourceArns": [
    "arn:aws:apprunner:<region>:<aws_account_id>:service/<service_name>/<service_id>"
  ]
}
```

AWS WAF ウェブ ACLsテストとログ記録

ウェブ ACL でルールアクションをカウントに設定すると、 はルールに一致するリクエストの数にリクエスト AWS WAF を追加します。App Runner サービスでウェブ ACL をテストするには、ルールアクションをカウントに設定し、各ルールに一致するリクエストの量を考慮します。例えば、通常のユーザートラフィックであると判断した多数のリクエストに一致するBlockアクションのルールを設定します。その場合は、ルールの再設定が必要になる場合があります。詳細については、「 AWS WAF デベロッパーガイド」のAWS WAF 「 保護のテストとチューニング」を参照してください。

リクエストヘッダーを Amazon CloudWatch Logs ロググループ、Amazon Simple Storage Service (Amazon S3) バケット、または Amazon Data Firehose に記録する AWS WAF ように を設定することもできます。詳細については、AWS WAF デベロッパーガイドの「ウェブ ACL トラフィックのログ記録」を参照してください。

App Runner サービスに関連付けられているウェブ ACL に関連するログにアクセスするには、次のログフィールドを参照してください。

- `httpSourceName`: を含む `APPRUNNER`
- `httpSourceId`: を含む `customeraccountid-apprunnerserviceid`

詳細については、「 AWS WAF デベロッパーガイド」の「ログの例」を参照してください。

> **Important**
>
> WAF ウェブ ACLsルールは、IP ベースのルールに準拠していません。これは、現在、リクエストソース IP データの WAF に関連付けられた App Runner プライベートサービスへの転送

をサポートしていないためです。App Runner アプリケーションにソース IP/CIDR 受信トラフィックコントロールルールが必要な場合は、WAF ウェブ ACLs の代わりにプライベートエンドポイントのセキュリティグループルールを使用する必要があります。

設定ファイルを使用した App Runner サービスオプションの設定

> **Note**
>
> 設定ファイルは、ソースコードに基づくサービスにのみ適用されます。イメージベースのサービスでは設定ファイルは使用できません。

ソースコードリポジトリを使用して AWS App Runner サービスを作成する場合、 はサービスの構築と開始に関する情報 AWS App Runner を必要とします。この情報は、App Runner コンソールまたは API を使用してサービスを作成するたびに指定できます。または、設定ファイルを使用してサービスオプションを設定することもできます。ファイルで指定したオプションはソースリポジトリの一部となり、これらのオプションへの変更は、ソースコードへの変更の追跡方法と同様に追跡されます。App Runner 設定ファイルを使用して、API がサポートするよりも多くのオプションを指定できます。API がサポートする基本オプションのみが必要な場合は、設定ファイルを提供する必要はありません。

App Runner 設定ファイルは、アプリケーションのリポジトリのソースディレクトリ`apprunner.yaml`に という名前の YAML ファイルです。サービスのビルドオプションとランタイムオプションを提供します。このファイルの値は、サービスを構築して開始する方法を App Runner に指示し、ネットワーク設定や環境変数などのランタイムコンテキストを提供します。

App Runner 設定ファイルには、CPU やメモリなどの操作設定は含まれません。

App Runner 設定ファイルの例については、「」を参照してくださいthe section called “例”。完全なリファレンスガイドについては、「」を参照してくださいthe section called “リファレンス”。

トピック

- App Runner 設定ファイルの例
- App Runner 設定ファイルリファレンス

App Runner 設定ファイルの例

> **Note**
>
> 設定ファイルは、ソースコードに基づくサービスにのみ適用されます。イメージベースのサービスでは設定ファイルは使用できません。

次の例は、 AWS App Runner 設定ファイルを示しています。一部の は最小限で、必要な設定のみが含まれます。その他は、すべての設定ファイルセクションを含めて完了しています。App Runner 設定ファイルの概要については、「」を参照してくださいApp Runner 設定ファイル。

設定ファイルの例

最小設定ファイル

最小限の設定ファイルでは、App Runner は次のことを前提としています。

- ビルドまたは実行時にカスタム環境変数は必要ありません。
- 最新のランタイムバージョンが使用されます。
- デフォルトのポート番号とポート環境変数が使用されます。

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install pipenv
      - pipenv install
run:
  command: python app.py
```

完全な設定ファイル

この例では、マネージドランタイムですべての設定キーをapprunner.yaml元の形式で使用しています。

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip install pipenv
      - pipenv install
    post-build:
      - python manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.7.7
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

完全な設定ファイル — (改訂されたビルドを使用)

この例では、 のすべての設定キーをマネージドランタイムapprunner.yamlで使用する方法を示します。

pre-run パラメータは、改訂された App Runner ビルドでのみサポートされます。アプリケーションが元の App Runner ビルドでサポートされているランタイムバージョンを使用している場合は、このパラメータを設定ファイルに挿入しないでください。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

> **Note**
>
> この例は Python 3.11 用であるため、 コマンドpip3と python3 コマンドを使用します。詳細については、Python プラットフォームトピック特定のランタイムバージョンのコールアウトの「」を参照してください。

Example apprunner.yaml

```
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip3 install pipenv
      - pipenv install
    post-build:
      - python3 manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.11
  pre-run:
    - pip3 install pipenv
    - pipenv install
    - python3 copy-global-files.py
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
```

```
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

特定のマネージドランタイム設定ファイルの例については、 の特定のランタイムサブトピックを参照してくださいコードベースのサービス。

App Runner 設定ファイルリファレンス

> **Note**
>
> 設定ファイルは、ソースコードに基づくサービスにのみ適用されます。イメージベースのサービスでは設定ファイルは使用できません。

このトピックは、 AWS App Runner 設定ファイルの構文とセマンティクスに関する包括的なリファレンスガイドです。App Runner 設定ファイルの概要については、「」を参照してくださいApp Runner 設定ファイル。

App Runner 設定ファイルは YAML ファイルです。に名前を付けapprunner.yaml、アプリケーションのリポジトリのソースディレクトリに配置します。

構造の概要

App Runner 設定ファイルは YAML ファイルです。に名前を付けapprunner.yaml、アプリケーションのリポジトリのソースディレクトリに配置します。

App Runner 設定ファイルには、次の主要部分が含まれています。

- トップセクション – トップレベルキーが含まれます
- ビルドセクション – ビルドステージを設定します
- 実行セクション – ランタイムステージを設定します

上部セクション

ファイルの上部にあるキーは、ファイルとサービスランタイムに関する一般的な情報を提供します。次のキーを使用できます。

- `version` – 必須。App Runner 設定ファイルのバージョン。最新バージョンを使用するのが理想的です。

  [Syntax] (構文)

  ```
  version: version
  ```

  Example

  ```
  version: 1.0
  ```

- `runtime` – 必須。アプリケーションが使用するランタイムの名前。App Runner が提供するさまざまなプログラミングプラットフォームで使用可能なランタイムについては、「」を参照してくださいコードベースのサービス。

  > **Note**
  >
  > マネージドランタイムの命名規則は *<language-name><major-version>* です。

  [Syntax] (構文)

  ```
  runtime: runtime-name
  ```

  Example

  ```
  runtime: python3
  ```

ビルドセクション

ビルドセクションでは、App Runner サービスデプロイのビルドステージを設定します。ビルドコマンドと環境変数を指定できます。ビルドコマンドが必要です。

セクションは `build:`キーで始まり、次のサブキーがあります。

- `commands` – 必須。さまざまなビルドフェーズで App Runner が実行するコマンドを指定します。次のサブキーが含まれます。
  - `pre-build` – オプション。ビルドの前に App Runner が実行するコマンド。たとえば、npm依存関係をインストールしたり、ライブラリをテストしたりします。
  - `build` – 必須。App Runner がアプリケーションを構築するために実行するコマンド。たとえば、 を使用しますpipenv。
  - `post-build` – オプション。App Runner がビルド後に実行するコマンド。例えば、Maven を使用してビルドアーティファクトを JAR または WAR ファイルにパッケージ化したり、テストを実行したりできます。

  [Syntax] (構文)

```
build:
  commands:
    pre-build:
      - command
      - …
    build:
      - command
      - …
    post-build:
      - command
      - …
```

  Example

```
build:
  commands:
    pre-build:
      - yum install openssl
    build:
      - pip install -r requirements.txt
    post-build:
      - python manage.py test
```

- `env` – オプション。ビルドステージのカスタム環境変数を指定します。名前と値のスカラーマッピングとして定義されます。これらの変数は、ビルドコマンドで名前で参照できます。

> **Note**
>
> この設定ファイルには、2 つの異なる場所に 2 つの異なるenvエントリがあります。1 つのセットはビルドセクションにあり、もう 1 つのセットは実行セクションにあります。
>
> - ビルドセクションの envセットは、ビルドプロセス中に pre-build、buildpost-build、、および pre-run コマンドで参照できます。
>
> 重要 - pre-run コマンドは、ビルドセクションで定義されている環境変数にのみアクセスできるにもかかわらず、このファイルの Run セクションにあることに注意してください。
>
> - Run セクションの envセットは、ランタイム環境の run コマンドで参照できます。

[Syntax] (構文)

```
build:
  env:
    - name: name1
      value: value1
    - name: name2
      value: value2
    - …
```

Example

```
build:
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
```

実行セクション

実行セクションでは、App Runner アプリケーションデプロイのコンテナ実行ステージを設定します。ランタイムバージョン、実行前コマンド (改訂された形式のみ) 、スタートコマンド、ネットワークポート、環境変数を指定できます。

セクションは `run:`キーで始まり、次のサブキーがあります。

- `runtime-version` – オプション。App Runner サービス用にロックするランタイムバージョンを指定します。

  デフォルトでは、メジャーバージョンのみがロックされます。App Runner は、デプロイまたはサービスの更新ごとにランタイムで使用できる最新のマイナーバージョンとパッチバージョンを使用します。メジャーバージョンとマイナーバージョンを指定すると、どちらもロックされ、App Runner はパッチバージョンのみを更新します。メジャーバージョン、マイナーバージョン、パッチバージョンを指定すると、サービスは特定のランタイムバージョンでロックされ、App Runner によって更新されることはありません。

  [Syntax] (構文)

  ```
  run:
    runtime-version: major[.minor[.patch]]
  ```

  > **Note**
  >
  > 一部のプラットフォームのランタイムには、異なるバージョンコンポーネントがあります。詳細については、特定のプラットフォームトピックを参照してください。

  Example

  ```
  runtime: python3
  run:
    runtime-version: 3.7
  ```

- `pre-run` – オプション。 ビルド使用法の改訂のみ。ビルドイメージから実行イメージにアプリケーションをコピーした後に App Runner が実行するコマンドを指定します。ここにコマンドを入力して、 `/app` ディレクトリの外部で実行イメージを変更できます。たとえば、 `/app` ディレクトリの外部にある追加のグローバル依存関係をインストールする必要がある場合は、このサブセクションに必要なコマンドを入力します。App Runner ビルドプロセスの詳細については、「」を参照してくださいマネージドランタイムバージョンと App Runner ビルド。

> **Note**
>
> - 重要 – `pre-run` コマンドは Run セクションにリストされていますが、この設定ファイルの Build セクションで定義されている環境変数のみを参照できます。この実行セクションで定義されている環境変数を参照することはできません。
> - `pre-run` パラメータは、改訂された App Runner ビルドでのみサポートされます。アプリケーションが元の App Runner ビルドでサポートされているランタイムバージョンを使用している場合は、このパラメータを設定ファイルに挿入しないでください。詳細については、「マネージドランタイムバージョンと App Runner ビルド」を参照してください。

[Syntax] (構文)

```
run:
  pre-run:
      - command
      - …
```

- `command` – 必須。App Runner がアプリケーションビルドの完了後にアプリケーションを実行するために使用するコマンド。

  [Syntax] (構文)

  ```
  run:
    command: command
  ```

- `network` – オプション。アプリケーションがリッスンするポートを指定します。この情報には以下が含まれます。
  - `port` – オプション。指定した場合、これはアプリケーションがリッスンするポート番号です。デフォルト: `8080`。
  - `env` – オプション。指定した場合、App Runner はデフォルトの環境変数 で同じポート番号を渡す (代わりに) ことに加えて、この環境変数のコンテナにポート番号を渡します`PORT`。つまり、を指定すると`env`、App Runner は 2 つの環境変数でポート番号を渡します。

  [Syntax] (構文)

  ```
  run:
  ```

```
  network:
    port: port-number
    env: env-variable-name
```

Example

```
run:
  network:
    port: 8000
    env: MY_APP_PORT
```

- env – オプション。実行ステージのカスタム環境変数の定義。名前と値のスカラーマッピングとして定義されます。ランタイム環境では、これらの変数を名前で参照できます。

> **Note**
>
> この設定ファイルには、2 つの異なる場所に 2 つの異なるenvエントリがあります。1 つのセットはビルドセクションにあり、もう 1 つのセットは実行セクションにあります。
>
> - ビルドセクションの envセットは、ビルドプロセス中に pre-build、buildpost-build、、および pre-run コマンドで参照できます。
>
> 重要 - pre-run コマンドは、ビルドセクションで定義されている環境変数にのみアクセスできるにもかかわらず、このファイルの Run セクションにあることに注意してください。
>
> - Run セクションの envセットは、ランタイム環境の run コマンドで参照できます。

[Syntax] (構文)

```
run:
  env:
    - name: name1
      value: value1
    - name: name2
      value: value2
  secrets:
    - name: name1
      value-from: arn:aws:secretsmanager:region:aws_account_id:secret:secret-id
    - name: name2
      value-from: arn:aws:ssm:region:aws_account_id:parameter/parameter-name
```

```
    - …
```

Example

```
run:
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-
S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

App Runner API

AWS App Runner アプリケーションプログラミングインターフェイス (API) は、App Runner サービスにリクエストを行うための RESTful API です。API を使用して、 で App Runner リソースを作成、一覧表示、説明、更新、削除できます AWS アカウント。

API はアプリケーションコードで直接呼び出すことも、いずれかの AWS SDKsを使用することもできます。

API リファレンスの詳細については、AWS App Runner 「 API リファレンス」を参照してください。

AWS 開発者ツールの詳細については、 「構築するツール AWS」を参照してください。

トピック

- AWS CLI を使用して App Runner を操作する
- AWS CloudShell を使用して を操作する AWS App Runner

AWS CLI を使用して App Runner を操作する

コマンドラインスクリプトの場合は、 AWS CLI を使用して App Runner サービスを呼び出します。詳細な AWS CLI リファレンス情報については、AWS CLI 「 コマンドリファレンス」の「apprunner」を参照してください。

AWS CloudShell を使用すると、 AWS CLI 開発環境に をインストールせずに、 AWS マネジメントコンソール 代わりに で使用できます。インストールの回避に加えて、認証情報を設定する必要も、リージョンを指定する必要はありません。 AWS マネジメントコンソール セッションはこのコンテキストを に提供します AWS CLI。CloudShell の詳細と使用例については、「」を参照してくださいthe section called “の使用 AWS CloudShell”。

AWS CloudShell を使用して を操作する AWS App Runner

AWS CloudShell はブラウザベースの事前認証済みシェルで、 から直接起動できます AWS マネジメントコンソール。任意のシェル (Bash、PowerShell、または Z シェル AWS App Runner) を使用して、 AWS サービス (を含む) に対して AWS CLI コマンドを実行できます。この手順は、コマンドラインツールのダウンロードもインストールも不要です。

AWS CloudShell から を起動 AWS マネジメントコンソールすると、コンソールへのサインインに使用した AWS 認証情報が新しいシェルセッションで自動的に利用可能になります。 AWS CloudShell ユーザーのこの事前認証により、 AWS CLI バージョン 2 (シェルのコンピューティング環境にプリインストール済み) を使用して App Runner などの AWS サービスとやり取りするときに、認証情報の設定をスキップできます。

トピック

- の IAM アクセス許可の取得 AWS CloudShell
- を使用した App Runner の操作 AWS CloudShell
- を使用した App Runner サービスの検証 AWS CloudShell

の IAM アクセス許可の取得 AWS CloudShell

が提供するアクセス管理リソースを使用すると AWS Identity and Access Management、管理者は IAM ユーザーにアクセス許可を付与して、環境の機能にアクセスして AWS CloudShell 使用できます。

管理者がユーザーにアクセス権を付与する最も簡単な方法は、 AWS 管理ポリシーを使用することです。AWS マネージドポリシーは、 AWSが作成および管理するスタンドアロンポリシーです。CloudShell の次の AWS 管理ポリシーを IAM ID にアタッチできます。

- `AWSCloudShellFullAccess`: すべての機能へのフルアクセス AWS CloudShell で を使用するアクセス許可を付与します。

IAM ユーザーが実行できるアクションの範囲を制限する場合は AWS CloudShell、`AWSCloudShellFullAccess`管理ポリシーをテンプレートとして使用するカスタムポリシーを作成できます。CloudShell でユーザーが使用できるアクションの制限の詳細については、AWS CloudShell 「 ユーザーガイド」の「IAM ポリシーによる AWS CloudShell アクセスと使用状況の管理」を参照してください。

> **Note**
>
> IAM ID には、App Runner を呼び出すアクセス許可を付与するポリシーも必要です。詳細については、「the section called “App Runner と IAM”」を参照してください。

を使用した App Runner の操作 AWS CloudShell

AWS CloudShell から を起動すると AWS マネジメントコンソール、コマンドラインインターフェイスを使用して App Runner の操作をすぐに開始できます。

次の例では、 AWS CLI CloudShell の を使用して App Runner サービスの 1 つに関する情報を取得します。

> **Note**
>
> AWS CLI で を使用する場合 AWS CloudShell、追加のリソースをダウンロードまたはインストールする必要はありません。さらに、ユーザーはシェル内で既に認証されているので、呼び出しを行う前に認証情報を設定する必要はありません。

Exampleを使用した App Runner サービス情報の取得 AWS CloudShell

1. から AWS マネジメントコンソール、ナビゲーションバーで使用できる以下のオプションを選択して CloudShell を起動できます。
   - CloudShell アイコンを選択します。
   - **`cloudshell`** 検索ボックスに入力を開始し、検索結果に表示されたら CloudShell オプションを選択します。
2. コンソールセッションの AWS リージョンにある AWS アカウント内のすべての現在の App Runner サービスを一覧表示するには、CloudShell コマンドラインに次のコマンドを入力します。

   ```
   $ aws apprunner list-services
   ```

   出力には、 サービスの概要情報が一覧表示されます。

   ```
   {
     "ServiceSummaryList": [
       {
         "ServiceName": "my-app-1",
         "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
         "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-
   app-1/8fe1e10304f84fd2b0df550fe98a71fa",
         "ServiceUrl": "psbqam834h.us-east-1.awsapprunner.com",
   ```

```
      "CreatedAt": "2020-11-20T19:05:25Z",
      "UpdatedAt": "2020-11-23T12:41:37Z",
      "Status": "RUNNING"
    },
    {
      "ServiceName": "my-app-2",
      "ServiceId": "ab8f94cfe29a460fb8760afd2ee87555",
      "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-app-2/
ab8f94cfe29a460fb8760afd2ee87555",
      "ServiceUrl": "e2m8rrrx33.us-east-1.awsapprunner.com",
      "CreatedAt": "2020-11-06T23:15:30Z",
      "UpdatedAt": "2020-11-23T13:21:22Z",
      "Status": "RUNNING"
    }
  ]
}
```

3. 特定の App Runner サービスの詳細な説明を取得するには、前のステップで取得した ARNs のいずれかを使用して、CloudShell コマンドラインに次のコマンドを入力します。

```
$ aws apprunner describe-service --service-arn arn:aws:apprunner:us-
east-2:123456789012:service/my-app-1/8fe1e10304f84fd2b0df550fe98a71fa
```

出力には、指定したサービスの詳細な説明が一覧表示されます。

```
{
  "Service": {
    "ServiceName": "my-app-1",
    "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-
app-1/8fe1e10304f84fd2b0df550fe98a71fa",
    "ServiceUrl": "psbqam834h.us-east-1.awsapprunner.com",
    "CreatedAt": "2020-11-20T19:05:25Z",
    "UpdatedAt": "2020-11-23T12:41:37Z",
    "Status": "RUNNING",
    "SourceConfiguration": {
      "CodeRepository": {
        "RepositoryUrl": "https://github.com/my-account/python-hello",
        "SourceCodeVersion": {
          "Type": "BRANCH",
          "Value": "main"
        },
        "CodeConfiguration": {
```

```
          "CodeConfigurationValues": {
            "BuildCommand": "[pip install -r requirements.txt]",
            "Port": "8080",
            "Runtime": "PYTHON_3",
            "RuntimeEnvironmentVariables": [
              {
                "NAME": "Jane"
              }
            ],
            "StartCommand": "python server.py"
          },
          "ConfigurationSource": "API"
        }
      },
      "AutoDeploymentsEnabled": true,
      "AuthenticationConfiguration": {
        "ConnectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/my-
github-connection/e7656250f67242d7819feade6800f59e"
      }
    },
    "InstanceConfiguration": {
      "CPU": "1 vCPU",
      "Memory": "3 GB"
    },
    "HealthCheckConfiguration": {
      "Protocol": "TCP",
      "Path": "/",
      "Interval": 10,
      "Timeout": 5,
      "HealthyThreshold": 1,
      "UnhealthyThreshold": 5
    },
    "AutoScalingConfigurationSummary": {
      "AutoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
      "AutoScalingConfigurationName": "DefaultConfiguration",
      "AutoScalingConfigurationRevision": 1
    }
  }
}
```

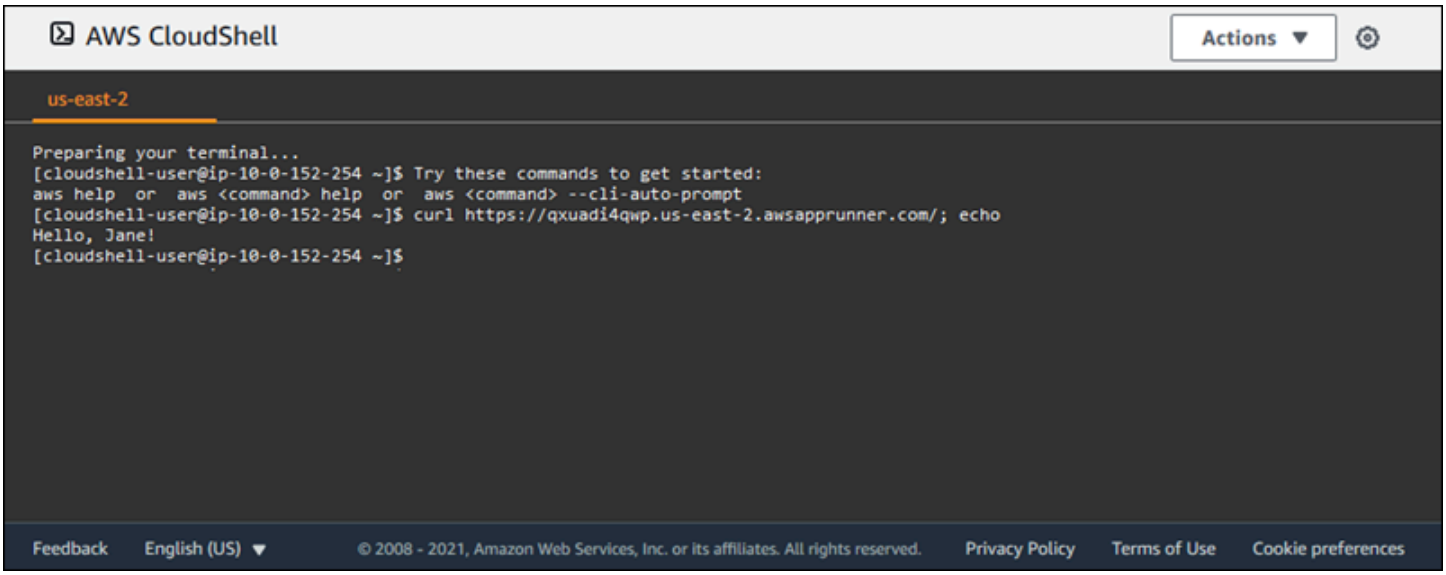
を使用した App Runner サービスの検証 AWS CloudShell

App Runner サービスを作成すると、App Runner はサービスのウェブサイトのデフォルトドメインを作成し、コンソールに表示します (または API コール結果で返します)。CloudShell を使用してウェブサイトを呼び出し、正しく動作していることを確認できます。

たとえば、 の説明に従って App Runner サービスを作成したら開始方法、CloudShell で次のコマンドを実行します。

```
$ curl https://qxuadi4qwp.us-east-2.awsapprunner.com/; echo
```

出力には、予想されるページの内容が表示されます。



トラブルシューティング

この章では、 AWS App Runner サービスの使用中に発生する可能性がある一般的なエラーや問題のトラブルシューティング手順について説明します。エラーメッセージは、サービスページのコンソール、API、またはログタブに表示されます。

トラブルシューティングに関するアドバイス、およびサポートへの一般的な質問に対する回答については、 ナレッジセンターにアクセスしてください。

トピック

サービスの作成に失敗した場合

App Runner サービスの作成に失敗すると、サービスは `CREATE_FAILED` ステータスになります。このステータスは、コンソールで作成に失敗しましたと表示されます。次のいずれかに関連する問題が原因で、サービスの作成に失敗することがあります。

- アプリケーションコード
- ビルドプロセス
- 設定
- リソースクォータ
- サービス AWS のサービス が使用する基盤となる の一時的な問題

サービスの作成に失敗したトラブルシューティングを行うには、以下を実行することをお勧めします。

1. サービスイベントとログを読み、サービスの作成に失敗した原因を確認します。
2. コードまたは設定に必要な変更を加えます。

3. サービスクォータに達した場合は、1 つ以上のサービスを削除します。
4. 別のリソースクォータに達した場合は、調整可能な場合は増やすことができます。
5. 上記のすべてのステップを完了したら、サービスの再構築を再試行してください。サービスを再構築する方法については、「」を参照してくださいthe section called “失敗したサービスを再構築する”。

> **Note**
>
> 問題の原因となっている可能性のある調整可能なリソースクォータの 1 つは、Fargate オンデマンド vCPU リソースです。
> vCPU リソース数は、App Runner がサービスに提供できるインスタンスの数を決定します。これは、 AWS Fargate サービスに存在する Fargate オンデマンド vCPU リソース数の調整可能なクォータ値です。アカウントの vCPU クォータ設定を表示したり、クォータの引き上げをリクエストしたりするには、 の Service Quotas コンソールを使用します AWS マネジメントコンソール。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「サービスAWS Fargate クォータ」を参照してください。

> **Important**
>
> 失敗したサービスの最初の作成試行以降に追加料金は発生しません。失敗したサービスは使用できませんが、サービスクォータにカウントされます。App Runner は失敗したサービスを自動的に削除しないため、障害の分析が終了したら削除してください。

カスタムドメイン名

このセクションでは、カスタムドメインにリンクするときに発生する可能性のあるさまざまなエラーをトラブルシューティングして解決する方法について説明します。

> **Note**
>
> App Runner アプリケーションのセキュリティを強化するために、*.awsapprunner.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化するために、App Runner アプリケーションのデフォルトのドメイン名で機密 Cookie を設定する必要がある場合は、__Host-プレフィックス付きの Cookie を使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメイン

を防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

カスタムドメインの作成失敗エラーの取得

- このエラーが CAA レコードの問題によるものであるかどうかを確認します。DNS ツリーのどこに CAA レコードがない場合、メッセージ を受け取りfail open、カスタムドメインを検証するための証明書 AWS Certificate Manager を発行します。これにより、App Runner はカスタムドメインを受け入れることができます。DNS レコードで CAA 証明書を使用している場合は、少なくとも 1 つのドメインの CAA レコードに が含まれていることを確認してくださいamazon.com。それ以外の場合、ACM は証明書の発行に失敗します。その結果、App Runner のカスタムドメインは作成されません。

  次の例では、DNS ルックアップツール DiG を使用して、必要なエントリがない CAA レコードを表示します。この例では、カスタムドメインexample.comとして を使用します。この例で次のコマンドを実行して、CAA レコードを確認します。

```
...
;; QUESTION SECTION:
;example.com.            IN  CAA

;; ANSWER SECTION:
example.com.        7200    IN  CAA 0 iodef "mailto:hostmaster@example.com"
example.com.        7200    IN  CAA 0 issue "letsencrypt.org"
...note absence of "amazon.com" in any of the above CAA records...
```

- ドメインレコードを修正し、少なくとも 1 つの CAA レコードに が含まれていることを確認しますamazon.com。
- カスタムドメインと App Runner のリンクを再試行してください。

CAA エラーを解決する方法については、以下を参照してください。

- 認証機関認可 (CAA) の問題
- ACM 証明書の発行または更新に関する CAA エラーを解決するにはどうすればよいですか?

カスタムドメインの DNS 証明書検証保留中エラーの取得

- カスタムドメイン設定の重要なステップをスキップしたかどうかを確認します。さらに、DiG などの DNS ルックアップツールを使用して DNS レコードを誤って設定したかどうかを確認します。特に、次の間違いがないか確認してください。
  - 欠落したステップ。
  - DNS レコードの二重引用符などのサポートされていない文字。
- 間違いを修正します。
- カスタムドメインと App Runner のリンクを再試行してください。

CAA 検証エラーを解決する方法については、以下を参照してください。

- DNS 検証
- the section called “カスタムドメイン名”

基本的なトラブルシューティングコマンド

- サービスが見つかったことを確認します。

```
aws apprunner list-services
```

- サービスを記述し、そのステータスを確認します。

```
aws apprunner describe-service --service-arn
```

- カスタムドメインのステータスを確認します。

```
aws apprunner describe-custom-domains --service-arn
```

- 進行中のすべてのオペレーションを一覧表示します。

```
aws apprunner list-operations --service-arn
```

カスタムドメイン証明書の更新

サービスにカスタムドメインを追加すると、App Runner は DNS サーバーに追加する一連の CNAME レコードを提供します。これらの CNAME レコードには証明書レコードが含まれます。App Runner は AWS Certificate Manager (ACM) を使用してドメインを検証します。App Runner は、これらの DNS レコードを検証して、このドメインの所有権が継続していることを確認します。DNS ゾーンから CNAME レコードを削除すると、App Runner は DNS レコードを検証できなくなり、カスタムドメイン証明書は自動的に更新されません。

このセクションでは、以下のカスタムドメイン証明書の更新の問題を解決する方法について説明します。

- the section called “CNAME が DNS サーバーから削除される”.
- the section called “証明書の有効期限が切れています”.

CNAME が DNS サーバーから削除される

- DescribeCustomDomains API を使用するか、App Runner コンソールのカスタムドメイン設定から CNAME レコードを取得します。保存された CNAMEs「 CertificateValidationRecords」を参照してください。
- 証明書検証 CNAME レコードを DNS サーバーに追加します。App Runner は、ドメインを所有していることを検証できます。CNAME レコードを追加した後、DNS レコードが伝播されるまでに最大 30 分かかることがあります。App Runner と ACM が証明書の更新プロセスを再試行するまでに数時間かかることもあります。CNAME レコードを追加する方法については、「」を参照してくださいthe section called “カスタムドメインを管理する”。

証明書の有効期限が切れています

- App Runner コンソールまたは API を使用して、App Runner サービスのカスタムドメインの関連付けを解除 (リンク解除) してから、関連付け (リンク) します。App Runner は、新しい証明書検証 CNAME レコードを作成します。
- 新しい証明書検証 CNAME レコードを DNS サーバーに追加します。

カスタムドメインの関連付けを解除 (リンク解除) および関連付け (リンク) する方法については、「」を参照してくださいthe section called “カスタムドメインを管理する”。

証明書が正常に更新されたことを確認する方法

証明書レコードのステータスをチェックして、証明書が正常に更新されたことを確認できます。証明書のステータスは、curl などのツールを使用して確認できます。

証明書の更新の詳細については、次のリンクを参照してください。

- ACM 証明書が更新対象外とマークされるのはなぜですか?
- ACM 証明書のマネージド更新
- DNS 検証

HTTP/HTTPS リクエストのルーティングエラー

このセクションでは、App Runner サービスエンドポイントに HTTP/HTTPS トラフィックをルーティングするときに発生する可能性のあるエラーをトラブルシューティングして解決する方法について説明します。

404 App Runner サービスエンドポイントに HTTP/HTTPS トラフィックを送信するときにエラーが見つからない

- App Runner `Host Header`がホストヘッダー情報を使用してリクエストをルーティングするため、 が HTTP リクエストのサービス URL を指していることを確認します。cURL、、ウェブブラウザなどのほとんどのクライアントは、ホストヘッダーを自動的にサービス URL にポイントします。クライアントがサービス URL を として設定しない場合`Host Header`、`404 Not Found`エラーが発生します。

  Exampleホストヘッダーが正しくない

```
$ ~ curl -I -H "host: foobar.com" https://testservice.awsapprunner.com/
HTTP/1.1 404 Not Found
transfer-encoding: chunked
```

Example正しいホストヘッダー

```
$ ~ curl -I -H "host: testservice.awsapprunner.com" https://
testservice.awsapprunner.com/
HTTP/1.1 200 OK
content-length: 11772
content-type: text/html; charset=utf-8
```

- クライアントがパブリックサービスまたはプライベートサービスへのリクエストルーティングのサーバー名インジケータ (SNI) を正しく設定していることを確認します。TLS の終了とリクエストのルーティングの場合、App Runner は HTTPS 接続で設定された SNI を使用します。

サービスが Amazon RDS またはダウンストリームサービスに接続できない場合

Amazon RDS データベースや他のダウンストリームアプリケーションやサービスへの接続に失敗すると、サービスにネットワーク設定の問題が発生する可能性があります。このトピックでは、ネットワーク設定に問題があるかどうかを判断する手順と、それらを修正するオプションについて説明します。App Runner のアウトバウンドトラフィック設定の詳細については、「」を参照してください送信トラフィックの VPC アクセスの有効化。

> **Note**
>
> VPC Connector の設定を表示するには、App Runner コンソールの左側のナビゲーションペインから、ネットワーク設定を選択します。次に、送信トラフィックタブを選択します。VPC コネクタを選択します。次のページには、VPC Connector の詳細が表示されます。このページから、VPC を使用するサブネット、セキュリティグループ、App Runner サービスを表示およびドリルダウンできます。

アプリケーションが別のダウンストリームサービスに接続できない原因を絞り込むには

1. VPC Connector で使用されるサブネットがプライベートサブネットであることを確認します。コネクタがパブリックサブネットで設定されている場合、各サブネットの基盤となる Hyperplane ENIs (エラスティックネットワークインターフェイス) にパブリック IP スペースがないため、サービスでエラーが発生します。

VPC コネクタがパブリックサブネットを使用している場合は、この設定を修正するための以下のオプションがあります。

a. 新しいプライベートサブネットを作成し、VPC Connector のパブリックサブネットの代わりに使用します。詳細については、「Amazon VPC ユーザーガイド」の「VPC のサブネット」を参照してください。

b. 既存のパブリックサブネットを NAT ゲートウェイ経由でルーティングします。詳細については、「Amazon VPC ユーザーガイド」の「NAT ゲートウェイ」を参照してください。

2. VPC Connector のセキュリティグループの入出力ルールが正しいことを確認します。App Runner コンソールの左側のナビゲーションペインから、ネットワーク設定 > 送信トラフィックを選択します。リストから VPC Connector を選択します。次のページには、検査するために選択できるセキュリティグループが一覧表示されます。

3. 接続しようとしている RDS インスタンスまたはその他のダウンストリームサービスに対して、セキュリティグループのインバウンドルールとアウトバウンドルールが正しいことを確認します。詳細については、App Runner アプリケーションが接続しようとしているダウンストリームサービスのサービスガイドを参照してください。

4. App Runner 設定の外部で他のタイプのネットワークセットアップの問題がないことを確認するには、App Runner の外部で RDS またはダウンストリームサービスに接続してみてください。

a. 同じ VPC 内の Amazon EC2 インスタンスから、RDS インスタンスまたはサービスに接続してみてください。

b. サービス VPC エンドポイントに接続しようとしている場合は、同じ VPC 内の EC2 インスタンスから同じエンドポイントにアクセスして接続を確認します。

5. ステップ 4 の接続テストのいずれかが失敗した場合、おそらく App Runner 設定の外部で AWS アカウント内の別のリソースに問題がある可能性があります。 AWS サポートに連絡して、他のネットワーク設定の問題をさらに分離して修正してください。

6. ステップ 4 の手順を実行して RDS インスタンスまたはダウンストリームサービスに正常に接続した場合は、このステップの手順に進みます。Hyperplane ENI フローログを有効にして検査することで、トラフィックが ENI に入り込んでいるかどうかを確認します。

> **Note**
>
> これらのステップを完了し、必要な ENI フローログ情報を取得するには、App Runner サービスが正常に起動した後に RDS またはダウンストリームサービスへの接続試行を行う必要があります。アプリケーションは、実行中状態のときに RDS またはダウンス

トリームサービスへの接続オペレーションを実行する必要があります。それ以外の場合、ENIsは App Runner のロールバックワークフローの一部としてクリーンアップできます。このアプローチにより、ENIsをさらに調査できます。

a. AWS コンソールから EC2 コンソールを起動します。

b. 左側のナビゲーションペインのネットワークとセキュリティのグループで、ネットワークインターフェイスを選択します。

c. インターフェイスタイプ列と説明列にスクロールして、VPC コネクタに関連付けられたサブネット内の ENIs を見つけます。次の命名パターンがあります。

- インターフェイスタイプ: fargate
- 説明: で始まる AWSAppRunner ENI (例: AWSAppRunner ENI - abcde123-abcd-1234-1234-abcde1233456)

d. 行の先頭にあるチェックボックスを使用して、適用する ENIsを選択します。

e. アクションメニューからフローログの作成を選択します。

f. プロンプトに情報を入力し、ページの下部にあるフローログの作成を選択します。

g. 生成されたフローログを検査します。

- 接続のテスト中にトラフィックが ENI に入っていた場合、問題は ENI のセットアップとは関係ありません。App Runner サービス以外の AWS アカウント内の別のリソースでネットワーク設定の問題が発生する可能性があります。サポート AWS にお問い合わせください。
- 接続のテスト中にトラフィックが ENI に入らなかった場合は、 AWS サポートに連絡して、Fargate サービスに既知の問題があるかどうか確認することをお勧めします。

h. ネットワーク Reachability Analyzer ツールを使用します。このツールは、仮想ネットワークパスのソースに到達できない場合にブロックコンポーネントを特定することで、ネットワークの設定ミスを判断するのに役立ちます。詳細については、Amazon VPC Reachability Analyzer ガイドの「Reachability Analyzer とは」を参照してください。

App Runner ENI をソースとして、RDS ENI を送信先として入力します。

7. 問題をさらに絞り込むことができない場合、または前のステップを完了した後も RDS またはダウンストリームサービスに接続できない場合は、 AWS サポートに連絡してサポートを受けることをお勧めします。

インスタンスの起動またはスケーリングに十分な IP アドレスがない場合

> **Note**
>
> パブリックサービスの場合、App Runner は VPCs に Elastic Network Interface (ENI) を作成しないため、パブリックサービスはこの変更の影響を受けません。

このガイドは、送信トラフィックの VPC アクセスが有効になっている App Runner サービスで発生する可能性のある IP 枯渇エラーを解決するのに役立ちます。

App Runner は、VPC コネクタに関連付けられたサブネットでインスタンスを起動します。App Runner は、インスタンスが起動されるサブネットにインスタンスごとに 1 つの ENI を作成します。各 ENI は、そのサブネットでプライベート IP を使用します。サブネットには、そのサブネットに関連付けられた CIDR ブロックに応じて、使用可能な IPs が一定数あります。App Runner が ENI を作成するのに十分な IPs を持つサブネット (複数可) を見つけられない場合、App Runner サービスの新しいインスタンスを起動できません。これにより、サービスのスケールアップに関する問題が発生する可能性があります。このような場合、App Runner イベントログが表示され、App Runner が使用可能な IPs を持つサブネットを見つけることができないことが示されます。以下の手順でサービスを更新して、このようなエラーを解決できます。

サービスを更新して利用可能な IPsを増やす方法

サブネットで使用可能な IP アドレスの数は、そのサブネットに関連付けられた CIDR ブロックに基づいています。サブネットに関連付けられた CIDR ブロックは、作成後に更新することはできません。App Runner VPC コネクタは、作成後に更新することもできません。送信トラフィックの VPC アクセスを有効にして App Runner サービスにより多くの IPs を提供するには:

1. より大きな CIDR ブロックを使用して新しいサブネット (複数可) を作成します。
2. 新しいサブネット (複数可) を使用して新しい VPC コネクタを作成します。
3. App Runner サービスを更新して、新しい VPC コネクタを使用します。

サービスに必要な IPsの計算

より大きな CIDR ブロックを使用して新しいサブネット (複数可) を作成する前に、App Runner サービス全体で必要な IPs の数を決定します。コネクタに必要な IPs の数は、次のように計算することをお勧めします。

1. 送信トラフィックに対する VPC アクセスが有効になっているサービスごとに、自動スケーリング設定の最大サイズ (最大インスタンス) を書き留めます。
2. すべてのサービスで値を合計します。
3. ブルー/グリーンデプロイ中に起動された新しいインスタンスを考慮して、この合計を 2 倍にします。

例

同じ VPC コネクタを使用する 2 つのサービス A と B を検討してください。

1. サービス A の最大サイズは 25 に設定されています。
2. サービス B の最大サイズは 15 に設定されています。

必要な IPs = 2 × (25 + 15) = 80

サブネットに少なくとも 80 IPs が結合されていることを確認します。

新しいサブネットを作成する (複数可)

1. この式を使用して IPv4 に必要な CIDR ブロックサイズを決定します (5 つの IPs は AWS によって予約されていることに注意してください: サブネットのサイズ設定)。

```
Number of available IP addresses = 2^(32 - prefix length) - 5
```

```
Example :
For 192.168.1.0/24:
Prefix length is 24
Number of available IP addresses = 2^(32 - 24) - 5 = 2^8-5 = 251 IP addresses

For 10.0.0.0/16:
Prefix length is 16
Number of available IP addresses = 2^(32 - 16) - 5 = 2^16-5 = 65,531 IP addresses
```

```
Quick reference:
/24 = 251 IP addresses
/16 = 65,531 IP addresses
```

2. AWS EC2 CLI を使用して新しいサブネットを作成します。

```
aws ec2 create-subnet --vpc-id <my-vpc-id> --cidr-block <cidr-block>
```

例 (4,096 IPs を持つサブネットを作成します) 。

```
aws ec2 create-subnet --vpc-id my-vpc-id --cidr-block 10.0.0.0/20
```

3. 新しい VPC コネクタを作成します。「」を参照: VPC アクセスの管理
4. この新しい VPC コネクタを使用するように VPC への送信トラフィックを有効にしてサービスを更新します。サービスが更新されると、App Runner は新しいサブネットの使用を開始します。

> **Note**
>
> VPCsは、CIDR ブロックによってサブネットに割り当てることができる使用可能な IPs の数にも制限されます。CIDR ブロックが大きいサブネットを作成できない場合は、新しいサブネット (複数可) を作成する前に VPC をセカンダリ CIDR ブロックで更新する必要がある場合があります。

VPC へのセカンダリ CIDR ブロックのアタッチ

セカンダリ CIDR ブロックをこの VPC に関連付けます。

```
aws ec2 associate-vpc-cidr-block --vpc-id <my-vpc-id> --cidr-block <cidr-block>
```

例 :

```
aws ec2 associate-vpc-cidr-block --vpc-id my-vpc-id --cidr-block 10.1.0.0/16
```

検証

サービスを更新したら、以下を使用して、修正の検証を実行できます。

1. イベントログのモニタリング: App Runner サービスイベントログをモニタリングして、新しい IP または ENI 使用不可エラーが表示されないことを検証します。
2. サービススケーリングを確認する:
    1. 自動スケーリング設定で最小インスタンス数を変更してサービスを完全にスケールアップする
    2. IP 関連のエラーなしですべての新しいインスタンスが起動されることを確認する
    3. 複数のスケーリングイベントをモニタリングして、一貫したパフォーマンスを確保する
3. コンソールバナー: AWS マネジメントコンソールを使用している場合は、App Runner に IPs 不足に関するバナー警告が表示されないことを確認します。
4. VPC とサブネット IP 使用率:
    1. VPC ダッシュボードまたは CLI コマンドを使用して、新しいサブネットの IP アドレス使用率を確認します。
    2. サービスのスケールアップ後も使用可能な IPs の正常なマージンがあることを確認します。

一般的な落とし穴

App Runner サービスで IP の枯渇に対処するときは、以下の潜在的な問題に注意してください。

1. IP アドレス計画が不十分: 将来の IP ニーズを過小評価すると、枯渇の問題が繰り返し発生する可能性があります。潜在的なサービスの増加とピーク時の使用シナリオを考慮して、徹底的なキャパシティプランニングを実施します。
2. VPC 全体の IP 使用状況の無視: 同じ VPC 内の他の AWS のサービスも IP アドレスを消費することに注意してください。VPC とサブネットの設定を計画するときは、すべてのサービスの IP 要件を考慮してください。
3. サービスの更新を無視する: 新しいサブネットまたは VPC コネクタを作成したら、新しい設定を使用するように App Runner サービスを更新してください。そうしないと、使い果たされた IP 範囲が引き続き使用されます。
4. CIDR ブロックの重複の誤解: VPC にセカンダリ CIDR ブロックを追加するときは、既存のブロックと重複しないようにしてください。CIDR ブロックが重複すると、ルーティングの競合や IP アドレスのあいまいさが発生する可能性があります。

5. VPC の制限を超える: VPC は最大 5 つの CIDR ブロック (1 つのプライマリと 4 つのセカンダリ) を持つことができることに注意してください。これらの制約内で IP アドレス空間の拡張を計画します。
6. サブネット AZ ディストリビューションを無視する: 新しいサブネットを作成するときは、高可用性と耐障害性を確保するために、複数のアベイラビリティーゾーンに分散されていることを確認します。
7. ENI の制限を無視する: インスタンスにアタッチできる ENIs の数には制限があることに注意してください。AWS アカウントの制限が、計画されたネットワークインターフェイスの使用と一致していることを確認します。

これらの落とし穴を認識することで、VPC リソースをより効果的に管理し、App Runner サービスの IP 枯渇の問題を回避できます。

その他のリソース

1. AWS VPC ドキュメント
2. CIDR ブロックについて
3. App Runner VPC コネクタ

用語集

1. ENI:Elastic Network Interface、AWS の仮想ネットワークインターフェイス。
2. CIDR:Classless Inter-Domain Routing、IP アドレスを割り当てる方法。
3. VPC コネクタ: App Runner が VPC に接続できるようにするリソース。

App Runner のセキュリティ

でのクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS お客様とお客様の間の責任共有です。責任共有モデルではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS は、 で AWS サービスを実行するインフラストラクチャを保護する責任を担います AWS クラウド。 は、お客様が安全に使用できるサービス AWS も提供します。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。が適用されるコンプライアンスプログラムの詳細については AWS App Runner、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、App Runner を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。以下のトピックでは、セキュリティとコンプライアンスの目的を達成するように App Runner を設定する方法について説明します。また、App Runner リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

App Runner でのデータ保護

責任 AWS 共有モデル、 でのデータ保護に適用されます AWS App Runner。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM アイデンティティセンターまたは AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して App Runner AWS CLIまたは他の AWS のサービス を使用する場合も同様です。AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

トピック

- 暗号化を使用したデータの保護
- ネットワーク間のトラフィックのプライバシー

暗号化を使用したデータの保護

AWS App Runner は、指定したリポジトリからアプリケーションソース (ソースイメージまたはソースコード) を読み取り、サービスへのデプロイ用に保存します。詳細については、「アーキテクチャと概念」を参照してください。

データ保護とは、転送中 (App Runner との間で送受信されるデータ) と保管中 (AWS データセンターに格納されるデータ) のデータを保護することです。

データ保護の詳細については、「the section called “データ保護”」を参照してください。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

転送中の暗号化

転送中のデータ保護は、Transport Layer Security (TLS) を使用して接続を暗号化するか、クライアント側の暗号化 (送信前にオブジェクトが暗号化される) を使用するという 2 つの方法で実現できます。どちらの方法も、アプリケーションデータを保護するために有効です。接続を保護するには、アプリケーション、その開発者と管理者、およびそのエンドユーザーがオブジェクトを送受信するたびに、TLS を使用して暗号化します。App Runner は、TLS 経由でトラフィックを受信するようにアプリケーションを設定します。

クライアント側の暗号化は、デプロイのために App Runner に提供するソースイメージまたはコードを保護する有効な方法ではありません。App Runner は暗号化できないように、アプリケーションソースにアクセスする必要があります。したがって、開発環境またはデプロイ環境と App Runner 間の接続は必ず保護してください。

保管時の暗号化とキー管理

アプリケーションの保管中のデータを保護するために、App Runner はアプリケーションソースイメージまたはソースバンドルのすべての保存済みコピーを暗号化します。App Runner サービスを作成するときに、 を指定できます AWS KMS key。指定した場合、App Runner は指定されたキーを使

用してソースを暗号化します。指定しない場合、App Runner は AWS マネージドキー 代わりに を使用します。

App Runner サービス作成パラメータの詳細については、CreateService」を参照してください。AWS Key Management Service (AWS KMS) の詳細については、「 AWS Key Management Service デベロッパーガイド」を参照してください。

ネットワーク間のトラフィックのプライバシー

App Runner は Amazon Virtual Private Cloud (Amazon VPC) を使用して、App Runner アプリケーション内のリソース間の境界を作成し、それらのリソース、オンプレミスネットワーク、インターネット間のトラフィックを制御します。Amazon VPC セキュリティの詳細については、「Amazon VPC ユーザーガイド」の「Amazon VPC でのインターネットワークトラフィックのプライバシー」を参照してください。

App Runner アプリケーションをカスタム Amazon VPC に関連付ける方法については、「」を参照してくださいthe section called “送信トラフィック”。

VPC エンドポイントを使用して App Runner へのリクエストを保護する方法については、「」を参照してくださいthe section called “VPC エンドポイント”。

データ保護の詳細については、「the section called “データ保護”」を参照してください。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

App Runner の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に App Runner リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

トピック

- オーディエンス
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理

- App Runner と IAM の連携方法
- App Runner アイデンティティベースのポリシーの例
- App Runner のサービスにリンクされたロールの使用
- AWS の マネージドポリシー AWS App Runner
- App Runner の ID とアクセスのトラブルシューティング

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「App Runner の ID とアクセスのトラブルシューティング」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「App Runner と IAM の連携方法」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「App Runner アイデンティティベースのポリシーの例」を参照)

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルート

ユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスすることを人間 AWS のユーザーに要求する」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、ID またはリソースに関連付けられたときにアクセス許可を定義します。 は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイド の アクセスコントロールリスト (ACL) の概要 を参照してください。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の上限を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。

- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

App Runner と IAM の連携方法

IAM を使用して へのアクセスを管理する前に AWS App Runner、App Runner で使用できる IAM 機能を理解しておく必要があります。App Runner およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

トピック

- App Runner アイデンティティベースのポリシー
- App Runner リソースベースのポリシー
- App Runner タグに基づく認可
- App Runner ユーザーアクセス許可
- App Runner IAM ロール

App Runner アイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは許可または拒否するアクションとリソース、またアクションを許可または拒否する条件を指定できます。App Runner は、特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素のリファレンス」を参照してください。

アクション

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

App Runner のポリシーアクションは、アクションの前にプレフィックス を使用しますapprunner:。たとえば、 Amazon EC2 RunInstances API オペレーションで Amazon EC2 インスタンスを実行するためのアクセス許可をユーザーに付与するには、ポリシーに `ec2:RunInstances` アクションを含めます。ポリシーステートメントには、`Action` または `NotAction` エレメントを含める必要があります。App Runner は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": [
   "apprunner:CreateService",
   "apprunner:CreateConnection"
]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、`Describe` という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "apprunner:Describe*"
```

App Runner アクションのリストを確認するには、「サービス認可リファレンス」の「 で定義されるアクション AWS App Runner」を参照してください。

リソース

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

App Runner リソースには、次の ARN 構造があります。

```
arn:aws:apprunner:region:account-id:resource-type/resource-name[/resource-id]
```

ARN の形式の詳細については、の「Amazon リソースネーム (ARNs) と AWS サービス名前空間」を参照してくださいAWS 全般のリファレンス。

たとえば、 ステートメントで my-serviceサービスを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:apprunner:us-east-1:123456789012:service/my-service"
```

特定のアカウントに属するすべてのサービスを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:apprunner:us-east-1:123456789012:service/*"
```

リソースを作成するためのアクションなど、一部の App Runner アクションは、特定のリソースで実行できません。このような場合はワイルドカード *を使用する必要があります。

```
"Resource": "*"
```

App Runner リソースタイプとその ARNs「 で定義されるリソース AWS App Runner」を参照してください。 どのアクションで各リソースの ARN を指定できるかについては、「AWS App Runnerで定義されるアクション」を参照してください。

条件キー

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

App Runner は、いくつかのグローバル条件キーの使用をサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドのAWS 「グローバル条件コンテキストキー」を参照してください。

App Runner は、サービス固有の条件キーのセットを定義します。さらに、App Runner は、条件キーを使用して実装されるタグベースのアクセスコントロールをサポートしています。詳細については、「the section called “App Runner タグに基づく認可”」を参照してください。

App Runner 条件キーのリストを確認するには、「サービス認可リファレンス」の「 の条件キー AWS App Runner」を参照してください。条件キーを使用できるアクションとリソースについては、「 で定義されるアクション AWS App Runner」を参照してください。

例

App Runner アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいApp Runner アイデンティティベースのポリシーの例。

App Runner リソースベースのポリシー

App Runner はリソースベースのポリシーをサポートしていません。

App Runner タグに基づく認可

App Runner リソースにタグをアタッチするか、App Runner へのリクエストでタグを渡すことができます。タグに基づいてアクセスを管理するには、`apprunner:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。App Runner リソースのタグ付けの詳細については、「」を参照してくださいthe section called “設定”。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグに基づく App Runner サービスへのアクセスの制御」を参照してください。

App Runner ユーザーアクセス許可

App Runner を使用するには、IAM ユーザーに App Runner アクションへのアクセス許可が必要です。ユーザーにアクセス許可を付与する一般的な方法は、IAM ユーザーまたはグループにポリシーをアタッチすることです。ユーザーアクセス許可の管理の詳細については、IAM ユーザーガイドの「IAM ユーザーのアクセス許可の変更」を参照してください。

App Runner には、ユーザーにアタッチできる 2 つの管理ポリシーが用意されています。

- `AWSAppRunnerReadOnlyAccess` – App Runner リソースの詳細を一覧表示および表示するアクセス許可を付与します。
- `AWSAppRunnerFullAccess`– すべての App Runner アクションにアクセス許可を付与します。

ユーザーアクセス許可をより詳細に制御するには、カスタムポリシーを作成してユーザーにアタッチします。詳細については、IAM ユーザーガイドの「IAM ポリシーの作成」を参照してください。

ユーザーポリシーの例については、「」を参照してくださいthe section called “ユーザーポリシー”。

App Runner IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のエンティティです。

サービスリンクロール

サービスにリンクされたロールを使用すると、 AWS サービスは他の サービスのリソースにアクセスして、ユーザーに代わってアクションを実行できます。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

App Runner は、サービスにリンクされたロールをサポートしています。App Runner サービスにリンクされたロールの作成または管理については、「」を参照してくださいthe section called “サービスにリンクされたロールの使用”。

サービス役割

この機能により、ユーザーに代わってサービスがサービスロールを引き受けることが許可されます。この役割により、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを

完了することが許可されます。サービスロールはIAM アカウントに表示され、アカウントによって所有されます。つまり、IAM ユーザーはこのロールのアクセス許可を変更できます。ただし、それにより、サービスの機能が損なわれる場合があります。

App Runner は、いくつかのサービスロールをサポートしています。

アクセスロール

アクセスロールは、App Runner がアカウントの Amazon Elastic Container Registry (Amazon ECR) 内のイメージにアクセスするために使用するロールです。Amazon ECR 内のイメージにアクセスする必要があり、Amazon ECR Public では必須ではありません。

Amazon ECR のイメージに基づいてサービスを作成する前に、IAM を使用してサービスロールを作成します。サービスロール`AWSAppRunnerServicePolicyForECRAccess`で 管理ポリシーを使用します。その後、SourceConfigurationConfiguration パラメータの AuthenticationConfiguration メンバーで CreateService API を呼び出すとき、または App Runner コンソールを使用してサービスを作成するときに、このロールを App Runner に渡すことができます。

> **Note**
>
> アクセスロールに独自のカスタムポリシーを作成する場合は、必ず `ecr:GetAuthorizationToken` アクション`"Resource": "*"`に を指定してください。トークンを使用して、アクセスできる任意の Amazon ECR レジストリにアクセスできます。

アクセスロールを作成するときは、App Runner サービスプリンシパルを信頼されたエンティティ`build.apprunner.amazonaws.com`として宣言する信頼ポリシーを必ず追加してください。

アクセスロールの信頼ポリシー

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": "build.apprunner.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

App Runner コンソールを使用してサービスを作成する場合、コンソールは自動的にアクセスロールを作成し、新しいサービス用に選択することができます。コンソールには、アカウントの他のロールも一覧表示されます。必要に応じて別のロールを選択できます。

インスタンスロール

インスタンスロールは、App Runner がサービスのコンピューティングインスタンスに必要な AWS サービスアクションにアクセス許可を付与するために使用するオプションのロールです。アプリケーションコードが AWS アクション (APIs) を呼び出す場合は、App Runner にインスタンスロールを提供する必要があります。必要なアクセス許可をインスタンスロールに埋め込むか、独自のカスタムポリシーを作成してインスタンスロールで使用します。コードが使用する呼び出しを予測する方法はありません。したがって、この目的のために管理ポリシーは提供しません。

App Runner サービスを作成する前に、IAM を使用して、必要なカスタムポリシーまたは埋め込みポリシーでサービスロールを作成します。その後、InstanceConfiguration パラメータ`InstanceRoleArn`のメンバーで CreateService API を呼び出すとき、または App Runner コンソールを使用してサービスを作成するときに、このロールをインスタンスロールとして App Runner に渡すことができます。 InstanceConfiguration

インスタンスロールを作成するときは、App Runner サービスプリンシパルを信頼されたエンティティ`tasks.apprunner.amazonaws.com`として宣言する信頼ポリシーを必ず追加してください。

インスタンスロールの信頼ポリシー

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": "tasks.apprunner.aws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

App Runner コンソールを使用してサービスを作成する場合、コンソールにはアカウントのロールが一覧表示され、この目的のために作成したロールを選択できます。

サービスの作成の詳細については、「」を参照してくださいthe section called “作成”。

App Runner アイデンティティベースのポリシーの例

デフォルトでは、IAM ユーザーとロールには AWS App Runner リソースを作成または変更するアクセス許可はありません。また、 AWS マネジメントコンソール、 AWS CLI、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらの権限が必要な IAM ユーザーまたはグループにそのポリシーをアタッチする必要があります。

JSON ポリシードキュメントのこれらの例を使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

トピック

- ポリシーに関するベストプラクティス
- ユーザーポリシー
- タグに基づく App Runner サービスへのアクセスの制御

ポリシーに関するベストプラクティス

ID ベースのポリシーは、アカウント内で App Runner リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションでは、 AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能のAWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

ユーザーポリシー

App Runner コンソールにアクセスするには、IAM ユーザーに最小限のアクセス許可が必要です。これらのアクセス許可により、 の App Runner リソースの詳細を一覧表示および表示できます AWS ア

カウント。最低限必要なアクセス許可よりも制限の厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つユーザーに対してコンソールは意図したとおりに機能しません。

App Runner には、ユーザーにアタッチできる 2 つの管理ポリシーが用意されています。

- `AWSAppRunnerReadOnlyAccess` – App Runner リソースの詳細を一覧表示および表示するアクセス許可を付与します。
- `AWSAppRunnerFullAccess` – すべての App Runner アクションにアクセス許可を付与します。

ユーザーが App Runner コンソールを使用できるようにするには、少なくとも `AWSAppRunnerReadOnlyAccess`管理ポリシーをユーザーにアタッチします。代わりに `AWSAppRunnerFullAccess`管理ポリシーをアタッチするか、特定のアクセス許可を追加して、ユーザーがリソースを作成、変更、削除できるようにします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、ユーザーに実行を許可する API オペレーションに一致するアクションにのみアクセスを許可します。

次の例は、カスタムユーザーポリシーを示しています。独自のカスタムユーザーポリシーを定義するための出発点として使用できます。この例をコピーし、アクションの削除、リソースのスコープダウン、条件の追加を行います。

例: コンソールと接続管理のユーザーポリシー

このポリシー例では、コンソールアクセスを有効にし、接続の作成と管理を許可します。App Runner サービスの作成と管理は許可されません。これは、ソースコードアセットへの App Runner サービスアクセスを管理するロールを持つユーザーにアタッチできます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apprunner:List*",
        "apprunner:Describe*",
```

```
        "apprunner:CreateConnection",
        "apprunner:DeleteConnection"
      ],
      "Resource": "*"
    }
  ]
}
```

例: 条件キーを使用するユーザーポリシー

このセクションの例では、一部のリソースプロパティまたはアクションパラメータに依存する条件付きアクセス許可を示します。

このポリシー例では、App Runner サービスの作成を有効にしますが、 という名前の接続の使用を拒否しますprod。

JSON

```
  { "Version":"2012-10-17",
  "Statement":
     [ { "Sid": "AllowCreateAppRunnerServiceWithNonProdConnections",
         "Effect": "Allow",
         "Action": "apprunner:CreateService",
         "Resource": "*",
         "Condition":
            { "ArnNotLike":
               {"apprunner:ConnectionArn":"arn:aws:apprunner:*:*:connection/prod/
*"}
            }
       }
     ]
   }
```

このポリシー例では、 という名前の自動スケーリング設定preprodでのみ という名前の App Runner サービスを更新できますpreprod。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUpdatePreProdAppRunnerServiceWithPreProdASConfig",
            "Effect": "Allow",
            "Action": "apprunner:UpdateService",
            "Resource": "arn:aws:apprunner:*:*:service/preprod/*",
            "Condition": {
                "ArnLike": {
                    "apprunner:AutoScalingConfigurationArn":
 "arn:aws:apprunner:us-east-1:*:autoscalingconfiguration/preprod/*"
                }
              }
         }
     ]
}
```

タグに基づく App Runner サービスへのアクセスの制御

アイデンティティベースのポリシーの条件を使用して、タグに基づいて App Runner リソースへのアクセスを制御できます。この例では、App Runner サービスの削除を許可するポリシーを作成する方法を示します。ただし、アクセス許可は、サービスタグ `Owner` にそのユーザーのユーザー名の値がある場合のみ、付与されます。このポリシーでは、このアクションをコンソールで実行するために必要なアクセス許可も付与します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ListServicesInConsole",
      "Effect": "Allow",
      "Action": "apprunner:ListServices",
      "Resource": "*"
    },
```

```
    {
      "Sid": "DeleteServiceIfOwner",
      "Effect": "Allow",
      "Action": "apprunner:DeleteService",
      "Resource": "arn:aws:apprunner:us-east-1:*:service/*",
      "Condition": {
        "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
      }
    }
  ]
}
```

このポリシーをアカウントの IAM ユーザーにアタッチできます。という名前のユーザーが App Runner サービスを削除richard-roeしようとする場合は、サービスに Owner=richard-roeまたは のタグを付ける必要がありますowner=richard-roe。それ以外の場合、アクセスは拒否されます。条件キー名では大文字と小文字が区別されないため、条件タグキー Owner は Owner と owner の両方に一致します。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素：条件」を参照してください。

App Runner のサービスにリンクされたロールの使用

AWS App Runner は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、App Runner に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは App Runner によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

トピック

- 管理にロールを使用する
- ネットワークにロールを使用する

管理にロールを使用する

AWS App Runner は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、App Runner に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは App Runner によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、App Runner の設定が簡単になります。App Runner は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されている場合を除き、App Runner のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、最初に関連リソースを削除する必要があります。これにより、リソースへのアクセス許可が誤って削除されないため、App Runner リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連携するAWS サービス」を参照して、サービスにリンクされたロール列がはいになっているサービスを見つけてください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

App Runner のサービスにリンクされたロールのアクセス許可

App Runner は、AWSServiceRoleForAppRunner という名前のサービスにリンクされたロールを使用します。

このロールにより、App Runner は次のタスクを実行できます。

- Amazon CloudWatch Logs Logs ロググループにログをプッシュします。
- Amazon Elastic Container Registry (Amazon ECR) イメージプッシュをサブスクライブする Amazon CloudWatch Events ルールを作成します。
- トレース情報を に送信します AWS X-Ray。

AWSServiceRoleForAppRunner サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `apprunner.amazonaws.com`

AWSServiceRoleForAppRunner サービスにリンクされたロールのアクセス許可ポリシーには、App Runner がユーザーに代わってアクションを実行するために必要なすべてのアクセス許可が含まれています。

- 管理ポリシー `AppRunnerServiceRolePolicy`
- X-Ray トレースのポリシー – 次のポリシーの内容を参照してください。

X-Ray トレースのポリシー

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords",
        "xray:GetSamplingRules",
        "xray:GetSamplingTargets",
        "xray:GetSamplingStatisticSummaries"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの許可」を参照してください。

App Runner のサービスにリンクされたロールの作成

サービスリンクロールを手動で作成する必要はありません。 AWS マネジメントコンソール、、AWS CLIまたは AWS API で App Runner サービスを作成すると、App Runner によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。App Runner サービスを作成すると、App Runner によってサービスにリンクされたロールが再度作成されます。

App Runner のサービスにリンクされたロールの編集

App Runner では、AWSServiceRoleForAppRunner サービスにリンクされたロールを編集することはできません。サービスリンクロールの作成後は、さまざまなエンティティがロールを参照する可能

性があるため、ロール名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

App Runner のサービスにリンクされたロールの削除

サービスリンクロールを必要とする機能やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

サービスリンク役割のクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

App Runner では、これはアカウント内のすべての App Runner サービスを削除することを意味します。App Runner サービスの削除については、「」を参照してくださいthe section called “削除”。

> **Note**
>
> リソースを削除しようとしたときに App Runner サービスがロールを使用している場合、削除が失敗する可能性があります。失敗した場合は数分待ってから操作を再試行してください。

サービスにリンクされたロールを手動で削除する

IAM コンソール、 AWS CLI、または AWS API を使用して、AWSServiceRoleForAppRunner サービスにリンクされたロールを削除します。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

App Runner サービスにリンクされたロールでサポートされているリージョン

App Runner は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートしています。詳細については、AWS 全般のリファレンスの「AWS App Runner エンドポイントとクォータ」を参照してください。

ネットワークにロールを使用する

AWS App Runner は AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、App Runner に直接リンクされた一意のタイプの

IAM ロールです。サービスにリンクされたロールは App Runner によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、App Runner の設定が簡単になります。App Runner は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されている場合を除き、App Runner のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、最初に関連リソースを削除する必要があります。これにより、リソースへのアクセス許可が誤って削除されないため、App Runner リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連携するAWS サービス」を参照して、サービスにリンクされたロール列がはいになっているサービスを見つけてください。サービスにリンクされた役割に関するドキュメントをサービスで表示するには[はい] リンクを選択してください。

App Runner のサービスにリンクされたロールのアクセス許可

App Runner は、AWSServiceRoleForAppRunnerNetworking という名前のサービスにリンクされたロールを使用します。

このロールにより、App Runner は次のタスクを実行できます。

- VPC を App Runner サービスにアタッチし、ネットワークインターフェイスを管理します。

AWSServiceRoleForAppRunnerNetworking サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `networking.apprunner.amazonaws.com`

という名前のロールアクセス許可ポリシー`AppRunnerNetworkingServiceRolePolicy`には、App Runner がユーザーに代わってアクションを実行するために必要なすべてのアクセス許可が含まれています。

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可を設定する必要があります。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの許可」を参照してください。

App Runner のサービスにリンクされたロールの作成

サービスリンクロールを手動で作成する必要はありません。 AWS マネジメントコンソール、、AWS CLIまたは AWS API で VPC コネクタを作成すると、App Runner によってサービスにリンクされたロールが作成されます。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。VPC コネクタを作成すると、App Runner によってサービスにリンクされたロールが再度作成されます。

App Runner のサービスにリンクされたロールの編集

App Runner では、AWSServiceRoleForAppRunnerNetworking サービスにリンクされたロールを編集することはできません。サービスリンクロールの作成後は、さまざまなエンティティがロールを参照する可能性があるため、ロール名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

App Runner のサービスにリンクされたロールの削除

サービスリンクロールを必要とする機能やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除する前に、サービスリンク役割をクリーンアップする必要があります。

サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされた役割を削除するには最初に、その役割で使用されているリソースをすべて削除する必要があります。

App Runner では、これは、アカウント内のすべての App Runner サービスから VPC コネクタの関連付けを解除し、VPC コネクタを削除することを意味します。詳細については、「the section called “送信トラフィック”」を参照してください。

> **Note**
>
> リソースを削除しようとしたときに App Runner サービスがロールを使用している場合、削除が失敗する可能性があります。失敗した場合は数分待ってから操作を再試行してください。

サービスリンクロールを手動で削除する

IAM コンソール、 AWS CLI、または AWS API を使用して、AWSServiceRoleForAppRunnerNetworking サービスにリンクされたロールを削除します。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

App Runner サービスにリンクされたロールでサポートされているリージョン

App Runner は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートしています。詳細については、AWS 全般のリファレンスの「AWS App Runner エンドポイントとクォータ」を参照してください。

AWS の マネージドポリシー AWS App Runner

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、多くの一般的なユースケースにアクセス許可を付与するように設計されているため、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できます。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS 管理ポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。 AWS は、新しい AWS のサービス が起動されたとき、または既存のサービスで新しい API オペレーションが利用可能になったときに、 AWS 管理ポリシーを更新する可能性が最も高くなります。

詳細については「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS マネージドポリシーに対する App Runner の更新

App Runner の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更に関する自動通知については、App Runner ドキュメント履歴ページの RSS フィードにサブスクライブしてください。

変更	説明	日付
AWSAppRunnerReadOnlyAccess – 新しいポリシー	App Runner は、ユーザーが App Runner リソースの詳細を一覧表示および表示できるようにする新しいポリシーを追加しました。	2022 年 2 月 24 日
AWSAppRunnerFullAccess – 既存のポリシーの更新	App Runner は、 `iam:CreateServiceLinkedRole` アクションのリソースリストを更新して、`AWSServiceRoleForAppRunnerNetworking` サービスにリンクされたロールを作成できるようにしました。	2022 年 2 月 8 日
AppRunnerNetworkingServiceRolePolicy – 新しいポリシー	App Runner は、App Runner が Amazon Virtual Private Cloud を呼び出して VPC を App Runner サービスにアタッチし、App Runner サービスに代わってネットワークインターフェイスを管理できるようにする新しいポリシーを追加しました。ポリシーは、`AWSServiceRoleForAppRunnerNetworking` サービスにリンクされたロールで使用されます。	2022 年 2 月 8 日
AWSAppRunnerFullAccess – 新しいポリシー	App Runner は、ユーザーがすべての App Runner アクションを実行できるようにする新しいポリシーを追加しました。	2022 年 1 月 10 日
AppRunnerServiceRolePolicy – 新しいポリシー	App Runner は、App Runner が App Runner サービスに代わって Amazon CloudWatch Logs と Amazon CloudWatch Events を呼び出すことを許可する	2021 年 3 月 1 日

変更	説明	日付
	新しいポリシーを追加しました。ポリシーは、AWSServiceRoleForAppRunner サービスにリンクされたロールで使用されます。	
AWSAppRunnerServicePolicyForECRAccess – 新しいポリシー	App Runner は、App Runner がアカウントの Amazon Elastic Container Registry (Amazon ECR) イメージにアクセスできるようにする新しいポリシーを追加しました。	2021 年 3 月 1 日
App Runner が変更の追跡を開始しました	App Runner が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 3 月 1 日

App Runner の ID とアクセスのトラブルシューティング

次の情報は、 および IAM の使用時に発生する可能性がある一般的な問題の診断 AWS App Runner と修正に役立ちます。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

トピック

- App Runner でアクションを実行する権限がありません
- 自分の 以外のユーザーに App Runner リソース AWS アカウント へのアクセスを許可したい

App Runner でアクションを実行する権限がありません

がアクションを実行する権限がないと AWS マネジメントコンソール 通知した場合は、管理者に連絡してサポートを依頼してください。管理者は、 AWS サインイン認証情報を提供したユーザーです。

次の例のエラーは、 という IAM ユーザーがコンソールを使用して App Runner marymajor サービスの詳細を表示しようとしているが、 アクセスapprunner:DescribeService許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 apprunner:DescribeService on resource: my-example-service
```

この場合、Mary は管理者にポリシーを更新して、 apprunner:DescribeServiceアクションを使用して*my-example-service*リソースにアクセスすることを許可するよう依頼します。

自分の 以外のユーザーに App Runner リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- App Runner がこれらの機能をサポートしているかどうかを確認するには、「」を参照してくださいApp Runner と IAM の連携方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

App Runner でのログ記録とモニタリング

モニタリングは、 AWS App Runner サービスの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。 AWS ソリューションのすべての部分からモニタリングデータを収集することで、障害が発生した場合に障害をより簡単にデバッグできます。App Runner は、App Runner サービスをモニタリングし、潜在的なインシデントに対応するためのいくつかの AWS ツールと統合されています。

Amazon CloudWatch アラーム

Amazon CloudWatch アラームを使用すると、指定した期間にわたってサービスメトリクスを監視できます。メトリクスが特定の期間に特定のしきい値を超えた場合は、通知を受け取ります。

App Runner は、サービス全体と、ウェブサービスを実行するインスタンス (スケーリングユニット) に関するさまざまなメトリクスを収集します。詳細については、「メトリクス (CloudWatch)」を参照してください。

アプリケーションログ

App Runner はアプリケーションコードの出力を収集し、Amazon CloudWatch Logs にストリーミングします。この出力の内容はユーザー次第です。たとえば、ウェブサービスに対して行われたリクエストの詳細なレコードを含めることができます。これらのログレコードは、セキュリティとアクセスの監査に役立つ場合があります。詳細については、「ログ (CloudWatch Logs)」を参照してください。

AWS CloudTrail アクションログ

App Runner は AWS CloudTrail、App Runner のユーザー、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています。CloudTrail は、App Runner のすべての API コールをイベントとしてキャプチャします。CloudTrail コンソールで最新のイベントを表示でき、証跡を作成して、Amazon Simple Storage Service (Amazon S3) バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。詳細については、「API アクション (CloudTrail)」を参照してください。

App Runner のコンプライアンス検証

サードパーティーの監査者は、複数のコンプライアンスプログラムの一環として AWS App Runner のセキュリティと AWS コンプライアンスを評価します。これらのプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、「コンプライアンスAWS のサービス プログラムによるスコープ」の「コンプライアンス」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

App Runner の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。 は、低レイテンシー、高スループット、高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーン AWS リージョン を提供します。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

AWS リージョン およびアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

AWS App Runner は、ユーザーに代わって AWS グローバルインフラストラクチャの使用を管理および自動化します。App Runner を使用すると、 AWS が提供する可用性と耐障害性メカニズムを活用できます。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

のインフラストラクチャセキュリティ AWS App Runner

マネージドサービスである AWS App Runner は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティ手順で保護されています。

AWS 公開された API コールを使用して、ネットワーク経由で App Runner を管理および運用します。App Runner APIs を呼び出すクライアントは、Transport Layer Security (TLS) 1.2 以降をサポートする必要があります。また、一時的ディフィー・ヘルマン Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは Java 7 以降

など、ほとんどの最新システムでサポートされています。これらの要件は、App Runner アプリケーションのエンドポイントには適用されません。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

VPC エンドポイントでの App Runner の使用

AWS アプリケーションは、Amazon Amazon Virtual Private Cloud VPC) の VPC で AWS のサービス 実行される他の AWS App Runner サービスとサービスを統合する場合があります。アプリケーションの一部は、VPC 内から App Runner にリクエストを行う場合があります。たとえば、 AWS CodePipeline を使用して App Runner サービスに継続的にデプロイできます。アプリケーションのセキュリティを向上させる 1 つの方法は、VPC エンドポイント経由でこれらの App Runner リクエスト (および他の へのリクエスト AWS のサービス) を送信することです。

VPC エンドポイントを使用すると、 を使用するサポートされている AWS のサービス および VPC エンドポイントサービスに VPC をプライベートに接続できます AWS PrivateLink。インターネットゲートウェイ、NAT デバイス、VPN 接続、または Direct Connect 接続は必要ありません。

VPC 内のリソースは、パブリック IP アドレスを使用して App Runner リソースとやり取りしません。VPC と App Runner 間のトラフィックは Amazon ネットワークを離れません。VPC エンドポイントの詳細については、「 AWS PrivateLink ガイド」の「VPC エンドポイント」を参照してください。

> **Note**
>
> デフォルトでは、App Runner サービスのウェブアプリケーションは、App Runner が提供および設定する VPC で実行されます。この VPC はパブリックです。つまり、インターネットに接続されています。オプションで、アプリケーションをカスタム VPC に関連付けることができます。詳細については、「the section called “送信トラフィック”」を参照してください。
>
> サービスが VPC に接続されている場合でも、APIsを含む AWS インターネットにアクセスするようにサービスを設定できます。VPC アウトバウンドトラフィックのパブリックインターネットアクセスを有効にする方法については、「」を参照してくださいthe section called “サブネットを選択する際の考慮事項”。

App Runner は、アプリケーションの VPC エンドポイントの作成をサポートしていません。

App Runner の VPC エンドポイントの設定

VPC で App Runner サービスのインターフェイス VPC エンドポイントを作成するには、「 AWS PrivateLink ガイド」の「インターフェイスエンドポイントの作成」の手順に従います。[Service Name] (サービス名)には `com.amazonaws.region.apprunner` を選択します。

VPC ネットワークプライバシーに関する考慮事項

Important

App Runner に VPC エンドポイントを使用しても、VPC からのすべてのトラフィックがインターネット外に留まるとは限りません。VPC はパブリックである可能性があります。さらに、ソリューションの一部では、VPC エンドポイントを使用して AWS API コールを実行しない場合があります。たとえば、パブリックエンドポイントを使用して他の サービスを呼び出す AWS のサービス ことができます。VPC のソリューションにトラフィックプライバシーが必要な場合は、このセクションをお読みください。

VPC 内のネットワークトラフィックのプライバシーを確保するには、次の点を考慮してください。

- DNS 名を有効にする – アプリケーションの一部は、`apprunner.region.amazonaws.com`パブリックエンドポイントを使用してインターネット経由で App Runner にリクエストを送信する場合があります。VPC がインターネットアクセスで設定されている場合、これらのリクエストはユーザーには通知されずに成功します。これを防ぐには、エンドポイントの作成時に DNS 名を有効にするを有効にします。デフォルトでは、true に設定されます。これにより、パブリックサービスエンドポイントをインターフェイス VPC エンドポイントにマップする DNS エントリが VPC に追加されます。
- 追加サービス用に VPC エンドポイントを設定する – ソリューションは他の にリクエストを送信する場合があります AWS のサービス。例えば、 は にリクエストを送信する AWS CodePipeline 場合があります AWS CodeBuild。これらのサービスの VPC エンドポイントを設定し、これらのエンドポイントで DNS 名を有効にします。
- プライベート VPC を設定する – 可能な場合 (ソリューションがインターネットアクセスをまったく必要としない場合) 、VPC をプライベートとして設定します。これは、インターネット接続が

ないことを意味します。これにより、VPC エンドポイントが欠落しているとエラーが表示されるため、欠落しているエンドポイントを追加できます。

エンドポイントポリシーを使用して VPC エンドポイントでアクセスを制御する

VPC エンドポイントポリシーは App Runner でサポートされています。デフォルトでは、インターフェイスエンドポイントを介して App Runner へのフルアクセスが許可されます。VPC エンドポイントポリシーを使用して、App Runner エンドポイントにアクセスできる AWS プリンシパルを制御できます。または、セキュリティグループをエンドポイントネットワークインターフェイスに関連付けて、インターフェイスエンドポイントを介して App Runner へのトラフィックを制御することもできます。

インターフェイスエンドポイントとの統合

App Runner は、App Runner へのプライベート接続を提供し AWS PrivateLink、インターネットへのトラフィックの露出を排除する、 をサポートしています。を使用してアプリケーションが App Runner にリクエストを送信できるようにするには AWS PrivateLink、インターフェイスエンドポイントと呼ばれる VPC エンドポイントのタイプを設定します。詳細については、「AWS PrivateLink ガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

App Runner の設定と脆弱性の分析

AWS と のお客様は、高いレベルのソフトウェアコンポーネントのセキュリティとコンプライアンスを達成する責任を共有します。詳細については、 AWS 「 責任共有モデル」を参照してください。

パッチコンテナイメージ

コンテナイメージにパッチを適用することは、共有セキュリティモデルにおけるお客様の責任の一部です。イメージ所有者は、コンテナイメージを更新して定期的にパッチを適用する責任があります。コンテナイメージの更新を確認して適用するための定期的なスケジュールを確立することをお勧めします。イメージの脆弱性をスキャンする方法の詳細については、AWS App Runner ドキュメントを参照してください。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

App Runner のセキュリティのベストプラクティス

AWS App Runner には、独自のセキュリティポリシーを開発および実装する際に考慮すべきいくつかのセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションに相当するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、絶対的な解決策ではなく、役に立つ情報としてお考えください。

その他の App Runner セキュリティトピックについては、「」を参照してくださいセキュリティ。

予防的セキュリティのベストプラクティス

予防的セキュリティ管理では、インシデントが発生する前に防ぐことを試みます。

最小特権アクセスの実装

App Runner は、IAM ユーザーとアクセスロールに AWS Identity and Access Management (IAM) 管理ポリシーを提供します。 ???これらの管理ポリシーは、App Runner サービスの正しいオペレーションに必要なすべてのアクセス許可を指定します。

アプリケーションで、管理ポリシーのすべてのアクセス権限が必要とは限りません。カスタマイズして、ユーザーと App Runner サービスがタスクを実行するために必要なアクセス許可のみを付与できます。これは特に、ユーザーロールごとに異なるアクセス権限のニーズを持つユーザーポリシーに関連します。最小特権アクセスの実装は、セキュリティリスクと、エラーや悪意によってもたらされる可能性のある影響の低減における基本です。

イメージをスキャンして脆弱性がないか調べる

Amazon ECR の APIs を使用して、コンテナイメージ内のソフトウェアの脆弱性を特定できます。詳細については、Amazon ECR ドキュメントを参照してください。

セキュリティ問題の検出ベストプラクティス

セキュリティコントロールの検出により、セキュリティ違反が発生した後に識別されます。セキュリティ上の脅威やインシデントの検出に役立ちます。

モニタリングを実装する

モニタリングは、App Runner ソリューションの信頼性、セキュリティ、可用性、パフォーマンスを維持する上で重要な部分です。 は、 AWS サービスのモニタリングに役立ついくつかのツールとサービス AWS を提供します。

以下は、モニタリングする項目のいくつかの例です。

- App Runner の Amazon CloudWatch メトリクス – 主要な App Runner メトリクスとアプリケーションのカスタムメトリクスのアラームを設定します。詳細については、「メトリクス (CloudWatch)」を参照してください。
- AWS CloudTrail エントリ – `PauseService`や など、可用性に影響を与える可能性のあるアクションを追跡します`DeleteConnection`。詳細については、API アクション (CloudTrail) を参照してください

AWS 用語集

最新の AWS 用語については、 AWS の用語集 リファレンスのAWS 用語集を参照してください。